Universität Bielefeld

Technische Fakultät
Bioinformatics Resource Facility
Center for Biotechnology (CeBiTec)

BRIDGE

A Bioinformatics Software Platform for the
Integration of heterogeneous Data
from Genomic Explorations

Zur Erlangung des akademischen Grades eines Doktors der
Naturwissenschaften der Universität Bielefeld vorgelegte
Dissertation

von

Alexander Goesmann

March 11, 2004

Contents

List of Figures

List of Tables

Acknowledgements

This PhD thesis was carried out from June 2000 until March 2004 at the Center for Genome Research at Bielefeld University. The work was funded by a scholarship of the NRW Graduiertenkolleg Bioinformatik from October 2000 until September 2001. Since October 2001 I was employed in the BMB+F network for “Genome Research on Bacteria Relevant for Agriculture, Environment and Biotechnology”. This work would not have been possible without the extensive support of many people.

First of all, I would like to thank Prof. Dr. Robert Giegerich and Prof. Dr. Alfred Pühler for giving me the opportunity to work in this exciting area of research. Furthermore, I wish to thank Dr. Folker Meyer and Dr. Jörn Kalinowski for their scientific guidance.

Special thanks to all people of the bioinformatics group at the Center for Genome Research (now CeBiTec Bioinformatics Resource Facility) for their excellent cooperation and team work. I very much appreciate your personal commitment.

I am also especially grateful to Daniela Bartels, Torsten Kasch, Jörn Clausen, Dr. Olaf Kaiser, Peter Serocka, and many other people for numerous corrections and improvements of the text.

Ganz besonders herzlich möchte ich mich an dieser Stelle aber auch bei meinen Eltern für die vielfältige Unterstützung während meines Studiums und während der Promotion bedanken.

Nicht zuletzt gilt mein ganz persönlicher Dank an dieser Stelle meiner Partnerin Anke für ihr Verständnis und die unermüdliche Unterstützung während der Promotion.

CHAPTER 1

Motivation and overview

The flood of data acquired from the increasing number of publicly available genomes leads to new demands for bioinformatics software. With the growing amount of information resulting from high-throughput experiments, new questions arise that often focus on the comparison of genes, genomes, and their expression profiles. Inferring new knowledge by combining different kinds of “post genomics” data necessitates the development of new approaches that allow the integration of variable data sources into a flexible framework.

1.1. Motivation

Today, roughly 50–60% of all genes in a newly sequenced bacterial genome can be classified automatically based on sequence similarity [FES00]. A *functional annotation* can be assigned by using well established tools like BLAST [AMS$^+$97], HMMer [Edd98], InterPro [AAB$^+$01], and many others. For the remaining 40–50% it is still a laborious task to identify their function. These new genes encoding some special features of the organism are often among the most interesting ones for scientific progress or commercial purposes. Hence, it should always be worthwhile to spend some time (and money) for their detailed analysis.

The advent of high-throughput sequencing techniques (approximately 1–2 Mbp per day and sequencer) has decreased the time needed to obtain the complete genomic DNA sequence of an organism by several orders of magnitude. In particular, the analysis of prokaryotic genomes which are smaller and simpler than those of eukaryotes has become a standard task for many research groups and already lead to numerous inventions and novel scientific results. While the first years of genomic explorations concentrated on the analysis of single genes, the focus is now changing towards identifying the remaining genes of so far unknown functions and towards unraveling relationships between genes and their regulation at growing levels of complexity [BH02]. New techniques developed in the past few years allow the simultaneous measurement of all mRNAs or proteins in a cell which is essential for the identification of complex interactions and co-regulated genes. These information can then be used to construct gene regulatory networks and model metabolic pathways.

As an example, ambitious programs like the "Genomes to Life" project [FJT$^+$03], funded by the US Department of Energy[1] with $103 million for five years, show that today the analysis of a genome involves many areas of genomics and post-genomics research.

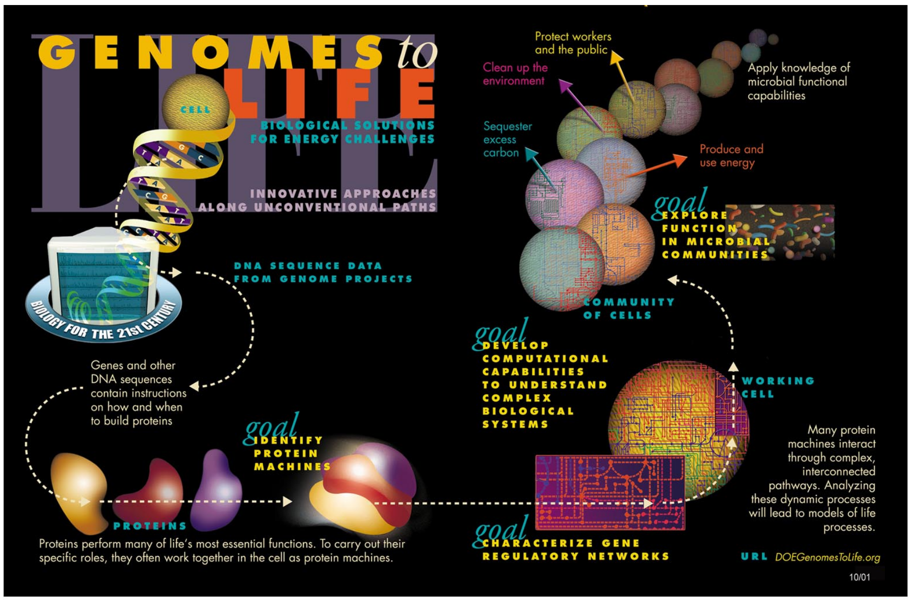

GENOMES to LIFE
BIOLOGICAL SOLUTIONS FOR ENERGY CHALLENGES
INNOVATIVE APPROACHES ALONG UNCONVENTIONAL PATHS
CELL
BIOLOGY FOR THE 21ST CENTURY
DNA SEQUENCE DATA FROM GENOME PROJECTS
Genes and other DNA sequences contain instructions on how and when to build proteins
goal IDENTIFY PROTEIN MACHINES
PROTEINS
Proteins perform many of life's most essential functions. To carry out their specific roles, they often work together in the cell as protein machines.
goal CHARACTERIZE GENE REGULATORY NETWORKS
goal DEVELOP COMPUTATIONAL CAPABILITIES TO UNDERSTAND COMPLEX BIOLOGICAL SYSTEMS
WORKING CELL
Many protein machines interact through complex, interconnected pathways. Analyzing these dynamic processes will lead to models of life processes.
COMMUNITY OF CELLS
goal EXPLORE FUNCTION IN MICROBIAL COMMUNITIES
Protect workers and the public
Clean up the environment
Apply knowledge of microbial functional capabilities
Sequester excess carbon
Produce and use energy
URL DOEGenomesToLife.org
10/01

Figure 1.1.: Overview of the "Genomes to Life" project (GTL) funded by the US Department of Energy. All research in this program is focused on the identification and analysis of genes, protein machines, and regulatory networks in complex biological systems and microbial communities.

The goals described for such programs (see figure 1.1) clarify that sophisticated data repositories are the essential basis for all further research towards understanding complex biolo-

[1] *http://doegenomestolife.org/*

gical systems. While the amount of data from high-throughput experiments increases exponentially, reliable and well structured storage of strongly connected information becomes a task of top priority. Each experiment itself may of course help to gain new insights but more complex relationships and regulatory networks will only be understood if the results of various experiments are combined, and analyzed together.

The huge amounts of data aquired from such experiments can only be handled with intensive bioinformatics support that has to provide an adequate infrastructure for storing and analyzing these data. For a detailed scientific analysis, quite individual questions are more often than not in the focus of the researchers interest. Thus, bioinformatics has to deliver tools as well as hardware and software solutions for answering such questions. This also includes the development of software toolkits that allow the implementation of special algorithms for specific tasks. As an example, it should be possible to implement typical workflows (e.g. for identifying co-regulated gene clusters that belong to a specific pathway) as described in algorithm 1 in a very simple and abstract manner (see next page).

This simple workflow already involves quite a few different data types and data sources: the information about genes that encode special enzymes acting in a selected pathway is coupled with expression data. The location of genes and their functional classification is used to identify co-regulated gene clusters.

The increasing number of applications of high-throughput methods for the simultaneous analysis of hundreds or thousands of genes in a single experiment leads to the demand for software solutions that allow the flexible integration of heterogeneous data types and data sources into an extensible platform. Such a system should not only be able to cope with high dimensional data but also provide different (meta-) views on the data that therefore have to be cross-linked. Furthermore, the software envisioned here should support higher level query and programming languages which allow a customizable exploration of different data sources. Instead of navigating through different databases and repositories by clicking on hyperlinks, the data could then be explored automatically according to individual requirements.

Although there are many software packages available that can be used for the analysis of data from one of the research domains described above (see chapter 3 for details about existing systems), there is no open source system known to the author that features the complete integration of different data sources **and** their corresponding applications.

The BRIDGE system (**B**ioinformatics **R**esource for the **I**ntegration of heterogeneous **D**ata from **G**enomic **E**xplorations) presented here describes a concept for the integration of heterogeneous data into a common framework. The implemented system includes a higher-level programming environment and it provides a comfortable and easy to learn interface for writing individual scripts in order to explore the flood of data. Thus, exactly tailored programs or specific algorithms can be implemented for data mining and visualization.

Finally, some sample applications illustrate the usability of this approach as a platform for systems biology.

Algorithm 1 A typical example for a simple workflow in pseudo code. The sample code shown here implements a simplified approach for finding clusters or operons of co-regulated genes.

```
Require: name of pathway and expression experiment
  for all genes in given experiment do
    if gene is significantly up regulated then
      if gene is already annotated then
        get the annotation
        get the gene name and gene product
        get the functional category
        get the EC number
      else
        get the amino acid sequence of the gene
        get the best 10 homologous sequences from SwissProt
        search for EC number in annotation of each homolog
      end if
      if an EC number was found in the annotation or one of the homologs then
        if EC number is in given pathway then
          store gene in list L
        end if
      end if
    end if
  end for
  for all genes in list L do
    get the start and stop position
    evaluate the distance of the genes
    if distance < 10000 bp then
      put gene into cluster / operon
    end if
  end for
```

1.2. Organization of the text

Chapter 2 reviews some fundamental knowledge about genomics, transcriptomics, and metabolic pathways as important areas of research for genome analysis.

In chapter 3 we describe a number of existing systems that represent state-of-the-art applications in the field of genome research. Furthermore, we present some recently developed approaches for the integration of heterogeneous data.

Chapter 4 focusses on a detailed specification analysis as a basis for the design of a new platform for systems biology.

In chapter 5 we discuss existing tools that meet some of the requirements and evaluate solutions for building a platform with respect to special requirements for data integration.

Based on conclusions derived from the previous chapters, chapter 6 illustrates the general design of the BRIDGE system.

Chapter 7 describes three components for building the platform. Parts of the GenDB and EMMA chapters were derived from [MGM$^+$03] and [DGB$^+$03].

Chapter 8 explains the implementation of a *General Project-Management System* and we present detailed solutions for integrating the specialized components into a platform for systems biology. Parts of the BRIDGE chapter were adapted from [GLR$^+$03].

Chapter 9 presents a number of successful applications of the BRIDGE architecture for the analysis of microbial genomes. Parts of these results were already described in [DGB$^+$03] and [GLR$^+$03]

Chapter 10 summarizes the basic aspects of this work. We evaluate and discuss the system presented here with respect to the obtained results and compare it to other approaches. Finally, we illustrate some ideas for further development and future directions.

The appendix contains selected topics of the source code and details about the implementation of special components.

CHAPTER 2

State of the art in genome research

In the past 20 years, genome research has become an important domain for the study of organisms which includes genome mapping, sequencing, and functional analysis. Biologists and other researchers are trying to understand the global regulatory mechanisms behind the transcription of genes into mRNA and their translation into protein sequences. Thereby, the function of proteins, interactions between them, and their role in complex biochemical networks is of major interest. The frequently used suffix *-omics* has become a common term that denotes the study of the entire set of something:

- *genomics*: study of all genes
- *transcriptomics*: study of all mRNA transcripts
- *proteomics*: study of all proteins
- *metabolomics*: study of all (“non polymeric”) metabolites in a cell

While genomics is often regarded as the study of more or less static properties of a genome, transcriptomics, proteomics, and metabolomics research analyzes dynamic features of an organism. The changing focus from static to dynamic analyses characterizes the *post genomics* aera with its somewhat misleading name. Since gene function experiments can be performed on a genome-wide scale, the term *functional genomics* can be defined as “the study of genes, their resulting proteins, and the role played by the proteins" in biochemical processes.[1] As an example, such analyses can be focused towards identifying the key mechanisms for the production of certain amino-acids or towards understanding the function of disease related genes. For unraveling such complex processes in detail, only a combined analysis at the sequence, mRNA, protein, and metabolite level is likely to reveal the true nature of such mechanisms. This is also reflected in figure 2.1 that illustrates life's complexity.

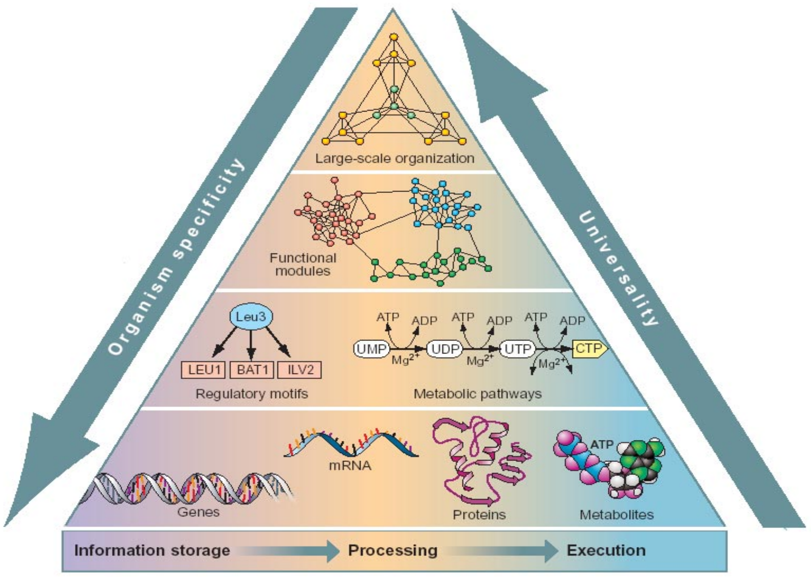


Figure 2.1.: Life's complexity pyramid: based on simple principles, genetic information is stored and translated into small functional units such as proteins and metabolites. These are the main building blocks that form functional modules that consist of regulatory motifs or metabolic pathways. On top of these units, large scale organizations implement the characteristic features of an organism. While the universality of certain modules increases from the bottom to the top, the organism specificity is mostly conserved in the DNA sequence and the encoded genes [OB02].

Although the basic principle for storing genetic information is quite simple, evolution has borne complex functional modules and well-structured large-scale organizations. While the precise repertoire of components – genes, proteins, metabolites – is unique to each organism, the key properties of larger functional modules are shared across most species (organism specificity vs. universality). To complete the list of definitions, *structural genomics* is de-

[1] *http://www.hyperdictionary.com/*

fined as the analysis of DNA and protein structures while *systems biology* describes the study of complex biological systems and biochemical networks.[2]

The following sections briefly introduce the most relevant topics of functional genomics as an essential basis for the development of a software platform for systems biology. After describing the basic principles of genome sequence analysis and microarray-technology, some fundamental concepts concerning the analysis of metabolic pathways are explained.

2.1. Genomics

After James D. Watson and Francis H. C. Crick described the structure of the DNA helix in 1953 [WC53], the basic mechanisms of DNA replication and recombination, protein synthesis, and gene expression were rapidly unravelled. Technological advances like the invention of the polymerase chain reaction (PCR) [SGS$^+$88] and automated DNA sequencing methods [SNC77, SSK$^+$86] have progressed to the point that today the entire genomic sequence of any organism can be obtained in a snatch. As of this writing, the GOLD database[3] reports more than 900 organisms, including completely sequenced genomes and genomes for which sequencing is in progress. For more than 800 genomes the (partial) sequence is already available in the NCBI databases[4].

2.1.1. Genome sequence analysis

All efforts for a complete analysis of almost every genome start by reading the DNA sequence of the whole organism. Ideally, the complete correct order of the four base pairs A, T, G, and C has to be determined before any further research can be initiated (i.e. the complete and correct DNA sequence is vital for a correct gene prediction based on characteristic DNA features, [FEN$^+$02]). Nowadays, whole genome sequencing is either done by a hierarchical (map based) sequencing approach (see figure 2.2) or by whole genome shotgun sequencing (see figure 2.3) [FF97, Gre01, KBB$^+$03a].

While the hierarchical approach first splits up the genomic DNA into a set of clones which have to be ordered based on their overlapping ends along the minimal tiling path, the shotgun approach simply cuts the whole genome into a large number of small fragments which are then sequenced and re-assembled.

[2] *http://www.ornl.gov/sci/techresources/Human_Genome/publicat/primer2001/glossary.shtml*
[3] *http://www.genomesonline.org/*
[4] *http://www.ncbi.nlm.nih.gov/About/tools/index.html*

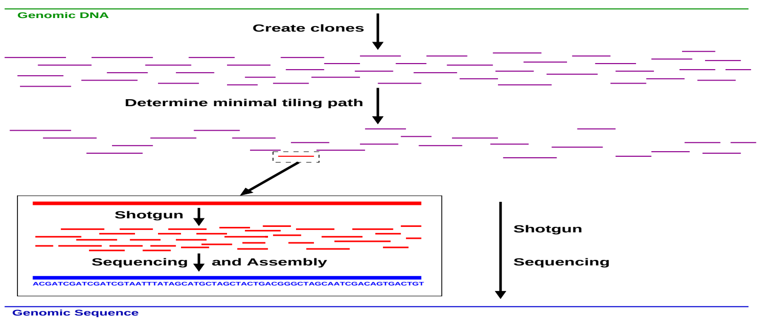

Genomic DNA
Create clones
Determine minimal tiling path
Shotgun
Sequencing
and Assembly
ACGATCGATCGATCGTAATTTATAGCATGCTAGCTACTGACGGGCTAGCAATCGACAGTGACTGT
Shotgun
Sequencing
Genomic Sequence

Figure 2.2.: The hierachical sequencing strategy first splits the genome into pieces of approximately 40 to 200 kb. These pieces are then cloned into *large insert libraries* (e.g. BACs, YACs, cosmids, fosmids). From the huge number of insert clones a *minimal tiling path* is created, selecting a subset of clones that cover the genome with minimal overlap between the individual clones. Since a map of clones is used, this approach is sometimes referred to as *map based shotgun*. The individual clones are sequenced using a shotgun approach for each one.

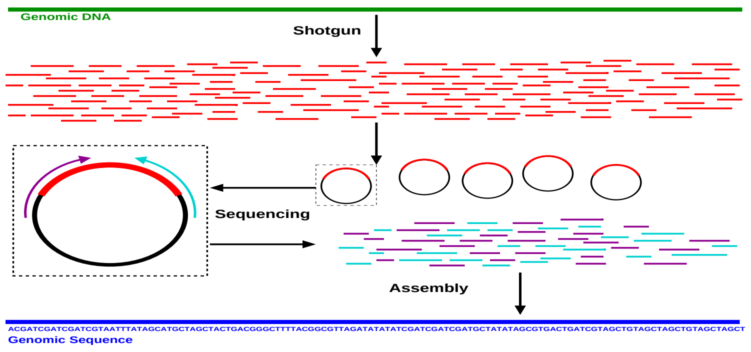

Genomic DNA
Shotgun
Sequencing
Assembly
ACGATCGATCGATCGTAATTTATAGCATGCTAGCTACTGACGGGCTTTTACGGCGTTAGATATATATCGATCGATCGATGCTATATAGCGTGACTGATCGTAGCTGTAGCTAGCTGTAGCTAGCT
Genomic Sequence

Figure 2.3.: For whole genome shotgun sequencing, the genome is split into a multitude of fragments of approximately 1 to 12 kB (shotgun phase). The resulting fragments are then cloned into sequencing vectors and transformed in bacterial cells (usually *E. coli*). The so-called vectors are small replicons that include a "multiple cloning site" where the fragments can be inserted. The fragment is thus flanked by the well known sequence of the vector and this sequence can be used to define a sequencing primer. This primer binds to the DNA of the vector. Two primers are used, yielding two sequences per "insert", a *forward* and a *reverse* sequence. Then the resulting DNA sequences can be assembled. Using overlaps between the individual sequences, an attempt is made to determine the genomic sequence from the sets of fragments.

Especially the whole genome shotgun approach depends on efficient assembling algorithms and requires considerable hard- and software support. In general, minimizing the manual effort for the shotgun approach by automated high-throughput sequencing pipelines has greatly decreased the cost for whole genome sequencing projects [FEN$^+$02]. After the sequencing and assembly phase, the obtained genomic sequence (usually a small number of contigs) has to be finished by closing the gaps between the contigs. Furthermore, the genome has to be polished in order to improve the quality of the consensus sequence. Finally, the complete genomic DNA sequence is ideally obtained in a single large contig as a basis for all further research. Although the completion of the sequencing phase in a genome project is always an important step towards understanding the genome and the basic genetic principles behind, the DNA sequence is actually just the starting point for large scale downstream analysis.

2.1.2. Finding genes – region prediction

The first step towards a detailed analysis of the DNA sequence in any genome is the identification of potentially functional regions like protein coding sequences (CDS) and other functional non-coding genes like transfer RNAs (tRNAs), ribosomal RNA genes (rRNAs), ribosomal binding sites (RBS), etc. Thereby, the prediction of such regions can be considered the most important task leading to the development of various approaches for gene prediction.

Due to their coding potential, the protein coding sequences in a bacterial genome typically exhibit certain, characteristic sequence properties which distinguish them from non-coding Open Reading Frames (ORFs) in the sequence. An additional useful property for gene identification is sequence homology of a potential coding region to genes of other organisms. *Ab initio* or *intrinsic* gene-finders exclusively use the statistical analysis of sequence properties (e.g. Hidden Markov Models) to distinguish real protein coding CDSs from ORFs. Examples for these *ab initio* gene-finders in prokaryotic sequence data are e.g. Glimmer (Gene Locator and Interpolated Context Modeller) [DHK$^+$99] or ZCURVE [GOZ03]. Programs like Critica (Coding Region Identification Tool Invoking Comparative Analysis) [BO99] and Orpheus [FMMG98] which additionally use homology-based information for gene prediction are also called *extrinsic* gene-finders.

For the prediction of other non–coding regions of interest such as tRNAs, rRNAs, signal peptides, etc. a number of tools exist at different levels of quality (tRNAscan-SE [LE97], SignalP [NEBH97], helix-turn-helix [BB90], TMHMM [SvHK98], etc.). Some of the obtained predictions are also strongly related to functional assignments for the identified regions so that it is not always possible to clearly distinguish the prediction of region and function.

An objective evaluation of the predictive accuracy of different gene-finders is difficult since an experimentally verified annotation for all genes of a bacterial genome does not yet exist (even for *E. coli*, only a few hundred genes have been verified experimentally by now).

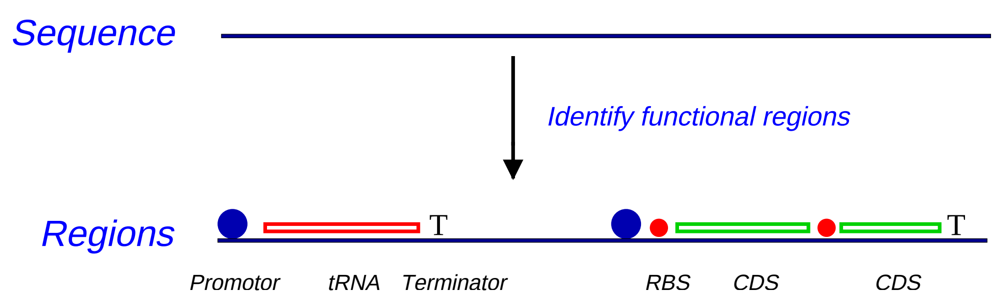


Figure 2.4.: Prediction of functional regions. Protein coding sequences (CDS) as well as other functional non–coding genes (tRNAs, rRNAs, promotors, terminators, etc.) can be identified by analyzing characteristic sequence properties.

Therefore, the current state-of-the-art is the comparison with available genome annotation data, which more or less reflects the manual annotation work of human experts. The reliability of these kinds of annotations varies, however, and depends heavily on the methods used and the manual effort involved in the annotation process. Furthermore, the state of the experimental knowledge concerning the respective organism differs quite a lot and thus reflects a certain degree of reliability for a given annotation. Nevertheless, the success of one or another gene prediction strategy can be evaluated to some degree by comparing the number of predicted genes to the number of genes found in an existing annotation and by calculating the selectivity and sensitivity for the gene numbers obtained.

2.1.3. Prediction of functions

After identifying the regions of interest in the genomic sequence, researchers find themselves confronted with the challenging task of assigning potential functions and biological meaning to more or less unimposing parts in the genomic sequence. Since the cost and manual effort for detailed wet lab experiments on each of these regions would clearly exceed the resources of every genome project, bioinformatics tools have been implemented that allow an automated prediction of potential gene functions.

Many of these tools rely on different strategies that compare unknown sequences to DNA or protein sequences that have already been determined by researchers in the past 20 years. Almost all of them have been deposited in a number of so-called *sequence databases* (from a computer scientist's point of view these are merely data collections). The most current list of these sequence repositories can be found either in the first issue of NAR (Nucleic Acid Research) each year or on the web via one of the different sequence retrieval servers (e.g. via the SRS server at *http://srs.ebi.ac.uk/*)[5].

[5]See section 3.5.4 for details about the SRS system.

While we can easily query these sequence databases for a gene with a specific name, the naming of genes is by no means consistent and each gene may have several names. So one reason for doing database searches based on sequence similarity is the chaotic state of the sequence databases.

The most important reason for performing similarity searches is the determination of putative functions for newly sequenced stretches of DNA. By comparing the new sequences to the databases of “well known” sequences and their “annotations”, we can derive a putative gene function.

If we find a database “match” for a new sequence, we can assume that the function of our new sequence may in fact be related to that of our match. This is based on a dictum by Carl Woese [Woe87] who stated that:

- Two proteins of identical function will have a similar protein structure, because protein structure determines the protein function.
- Two proteins of similar structure will have similar amino acid sequences.
- Two similar amino acid sequences will have some degree of DNA sequence similarity.
- Thus from a similar DNA or amino acid function a similar protein function might be inferred.

Although this is true for many proteins, it should be clearly stated that even small changes in the DNA sequence can render the gene product useless or completely change its function. In contrast to similarity in function, the term *homology* indicates a genetic relationship based on correspondence or relation in the type of a structure (here in the DNA or amino-acid sequence itself).

Unfortunately, a “match” in a DNA or protein database needs to be interpreted; the uninitiated may mistake a chance hit (the databases are very large) with a meaningful “match”.

Prominent and commonly applied tools like BLAST [AMS+97] or FASTA [Pea90, PL88] compare the DNA or amino-acid query sequence with huge databases of collected already known sequences by computing alignments. The results of these tools are supposed to reflect the degree of similarity between two genes in different organisms thus following the thesis that the same (or similar) gene function should have an (almost) identical underlying genomic sequence. Although these comparisons often reveal the homology among evolutionary related organisms, the results have to be interpreted carefully since they can only be as reliable as the database entry itself.[6]

[6]This refers to the fact that many database entries contain unsupervised and error prone data (e.g. GenBank [BKML+02]).

Other tools like Pfam [BBC$^+$02], Blocks [HGPH00, HHP99], iPSORT [BTM$^+$02], and PROSITE [FPB$^+$02] are based on (manually) curated motif or domain databases that allow the classification of proteins based on hidden markov models and other techniques. Recently developed tools like InterPro [AAB$^+$01] also combine the results of several other applications thus trying to compute more reliable and quite exact predictions that classify partial genomic sequences.

2.1.4. Genome annotation

Annotation is generally thought to possess best quality when performed by a human expert. The large amounts of data which have to be evaluated in any whole-genome annotation project, however, have led to the (partial) automation of the procedure. Hence, software assistance for computation, storage, retrieval, and analysis of relevant data has become essential for the success of any genome project. Genome annotation can be done automatically (e.g. by using the "best Blast hit") or manually. The latter is supposed to possess a higher quality but on the other hand takes much more time. However, to be sure about the "real biological function", each annotation of a gene would have to be confirmed by wet lab experiments.

Figure 2.5 shows the flowchart of an often employed genome annotation pipeline also displaying the interactions and dependencies between the single steps: e.g. a correct gene prediction depends heavily on the quality of the genomic sequence. Vice versa questionable predictions of regions can help to identify sequencing errors (e.g. frameshifts) that require further improvement of the sequence itself in some positions.

Another important aspect for the success of any genome annotation project is the use of a consistent nomenclature when assigning gene names. Comparing just a few existing genome annotations shows that there is no commonly used systematic naming scheme: for example, the genes coding for the enzyme *homoserine dehydrogenase* are named completely different in the corresponding SwissProt annotations for *E. coli* (*THRA* or *THRA1* or *THRA2* or *B0002*), *B. subtilis* (*HOM* or *TDM*), and *S. cerevisiae* (*HOM6* or *YJR139C* or *J2132*) as illustrated in figure 2.6.

They can only be identified as the same encoded enzyme because each database entry is additionally mapped onto the same enzyme classification number *EC 1.1.1.3* (see section 2.4.1 for further details on enzyme nomenclature). This does not only prevent simple comparisons between different organisms but also complicates the identification of genes with the same or similar function. Using a standardized vocabulary like the Gene Ontologies (see section 3.4.5) might therefore be one of the most fruitful efforts towards a unified standard for genome annotations.

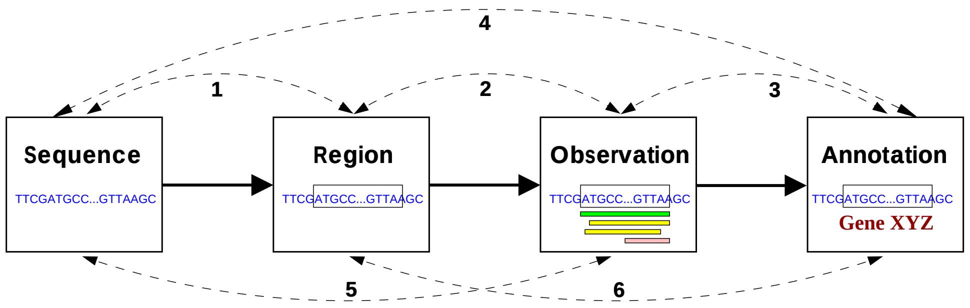
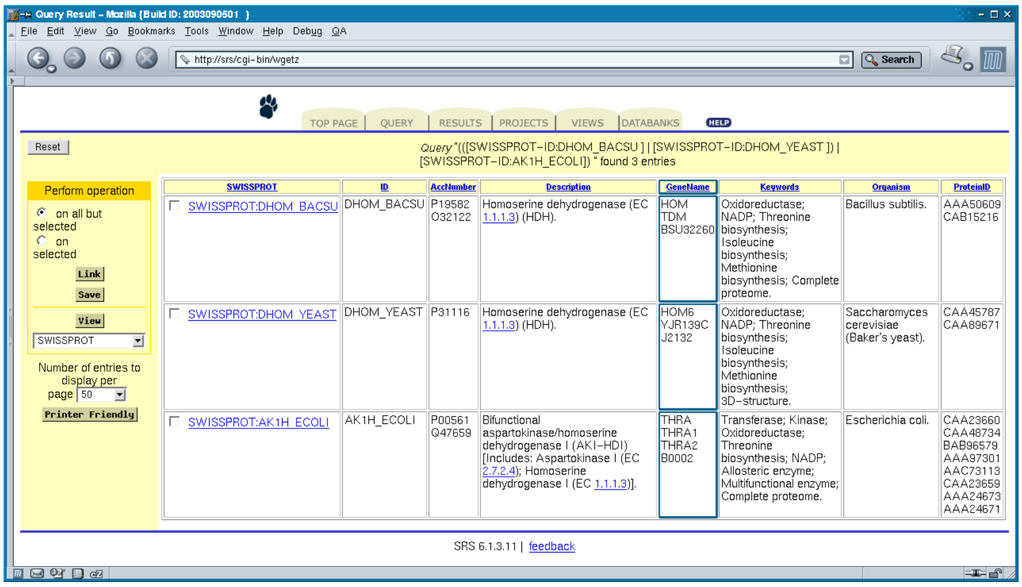


Figure 2.5.: Traditional flowchart of a genome annotation pipeline. The process of genome annotation can be defined as assigning a meaning to sequence data that would otherwise be almost devoid of information. By identifying regions of interest and defining putative functions for those areas, the genome can be understood and further research may be initiated. Since genome annotation is a dynamic process, the arrows indicate different mutual influences between the different steps. For example, the region prediction (1), the computation of observations (5), and the annotation (4) depend on the quality of the sequence (because of frameshifts etc.). On the other hand, "surprising" observations (2) or inconsistencies that were discovered during the annotation (6) may require updates of the region prediction. Changes of a region will thus produce new observations which have to be considered carefully for a novel annotation (3).



Query "(([SWISSPROT-ID:DHOM_BACSU] | [SWISSPROT-ID:DHOM_YEAST]) | [SWISSPROT-ID:AK1H_ECOLI]) " found 3 entries

SWISSPROT	ID	AccNumber	Description	GeneName	Keywords	Organism	ProteinID
☐ SWISSPROT:DHOM_BACSU	DHOM_BACSU	P19582 O32122	Homoserine dehydrogenase (EC 1.1.1.3) (HDH).	HOM TDM BSU32260	Oxidoreductase; NADP; Threonine biosynthesis; Isoleucine biosynthesis; Methionine biosynthesis; Complete proteome.	Bacillus subtilis.	AAA50609 CAB15216
☐ SWISSPROT:DHOM_YEAST	DHOM_YEAST	P31116	Homoserine dehydrogenase (EC 1.1.1.3) (HDH).	HOM6 YJR139C J2132	Oxidoreductase; NADP; Threonine biosynthesis; Isoleucine biosynthesis; Methionine biosynthesis; 3D-structure.	Saccharomyces cerevisiae (Baker's yeast).	CAA45787 CAA89671
☐ SWISSPROT:AK1H_ECOLI	AK1H_ECOLI	P00561 Q47659	Bifunctional aspartokinase/homoserine dehydrogenase I (AKI-HDI) [Includes: Aspartokinase I (EC 2.7.2.4); Homoserine dehydrogenase I (EC 1.1.1.3)].	THRA THRA1 THRA2 B0002	Transferase; Kinase; Oxidoreductase; Threonine biosynthesis; NADP; Allosteric enzyme; Multifunctional enzyme; Complete proteome.	Escherichia coli.	CAA23660 CAA48734 BAB96579 AAA97301 AAC73113 CAA23659 AAA24673 AAA24671

SRS 6.1.3.11 | feedback

Figure 2.6.: Searching for a *homoserine dehydrogenase* in the SwissProt database using the SRS system results in a number of hits for various organisms. The hits shown here illustrate that for only three organisms 9 different gene names were assigned.

2.2. Transcriptomics

A number of array-based technologies have been developed over the last years that allow the simultaneous measurements of thousands of interactions between mRNA-derived target molecules and genome-derived probes. As a high-throughput technique, microarray experiments are rapidly producing enormous amounts of raw and derived data never before encountered by biologists. These data sets consist of measured data, laboratory protocols, and experimental settings. A major challenge is the efficient storage and analysis of such large scale data sets associated with an enduring demand for good bioinformatics solutions, in particular for the (automated) evaluation of the results.

2.2.1. DNA array technology

The principle of microarray technology (see figure 2.7) is based on the differential expression of regulated genes that can be observed by simultaneously measuring the level of mRNA gene products of living cells. They allow the measurement of mRNA-abundance in cells for thousands of genes in parallel [SSDB95, DIB97, DBC$^+$99].

In its simplest sense, a DNA array is defined as an orderly arrangement of tens to hundreds of thousands of unique DNA molecules (probes) of known sequence [BH02]. These DNA probes can be either synthesized on a rigid surface (usually glass) or pre-synthesized probes (oligonucleotides or PCR products) can be attached to the array platform (usually glass or nylon membranes). The most widely used microarray flavors are the commercial Affymetrix GeneChip$^{\mathrm{TM}}$ technology described in [FRH$^+$93] and two-color cDNA microarrays developed by Pat Brown [SSDB95]. Since the latter (less expensive) method is applied at the Bielefeld Center for Genome Research, the first approach is not considered any further here. The following descriptions also focus on the production and evaluation of glass based microarrays but nevertheless most of the applied techniques are as well suited for the analysis of filters like nylon membranes.

As illustrated in figure 2.7, the basic experimental strategy in a cDNA microarray experiment is to purify RNA from two different sample materials grown under different conditions. One condition is often called “control” and the other “treatment”. More generally, the source material can arise from virtually any two different conditions or tissues. After extracting the mRNA from two strains or from different tissues or experimental conditions, the probe mRNA is transcribed into DNA by reverse transcription and thereby labeled with two fluorescent dyes respectively. Both dyes are then mixed and dispensed over the prepared slides with the single stranded target DNA molecules that can finally hybridize with the complementary probes. The intensity of the fluorescent molecules measured for the two channels by laser scanning thus reveals the amount of expressed mRNA in the original cells.

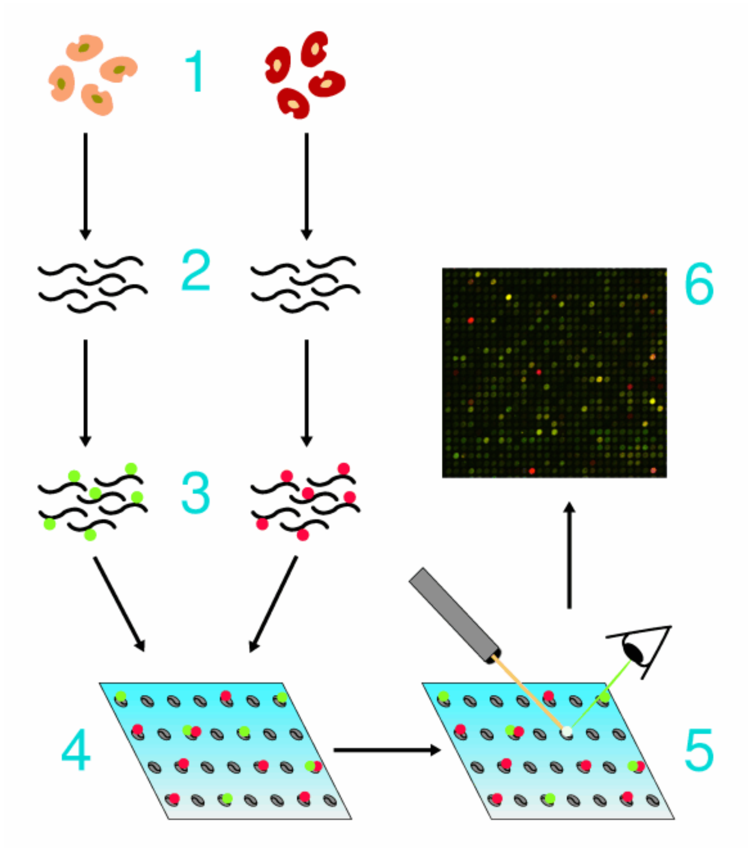


Figure 2.7.: In principle, each microarray experiment starts with a purification of RNA from two different sample materials grown under different conditions (1 & 2). The cDNA-probes are then created by reverse transcription of the RNA and labeled with two fluorophores (3). Afterwards, both probes are mixed and simultaneously hybridized to the microarray (4). During hybridization, the labeled transcripts bind to their corresponding reporter molecules in the spots. The array is scanned subsequently by a microarray scanner (5) that detects the fluorescent dyes and creates one digital image for each dye (6).

Since most microarray techniques are very sensitive and prone to changes in the environment (e.g. temperature, humidity, pressure), several *replica* are normally spotted for each gene in order to facilitate a sound statistical evaluation. These may either be *technical replica* e.g. obtained by spotting the same material several times onto different positions or *biological replica* that were produced e.g. by using different biological source materials to generate the hybridization probes.

Data acquisition

High-throughput microarray experiments produce large amounts of measured data and numerous results of further analysis steps. But also the production of the glass slides involves a number of steps where different types of data have to be stored carefully to ensure a correct evaluation of the results. A typical flowchart of microarray experiments can be divided into six production steps as displayed in figure 2.8.

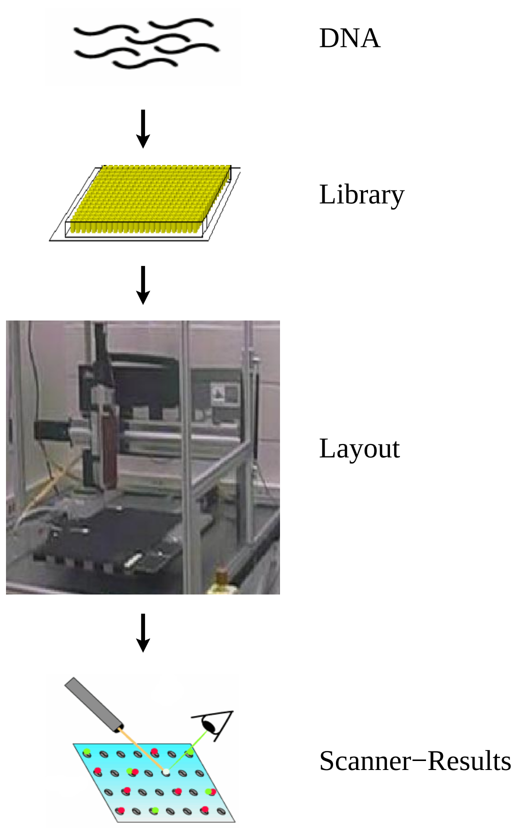


Figure 2.8.: A unique piece of DNA for each gene that should be represented on the microarray is typically stored in a set of microtiter plates called *library*. In most cases a working copy is produced from the original library. This may also include rearrangements (e.g. from 96 well plates to 384 well plates). A resulting set of *spotting plates* is typically used to define a *layout* for a series of slides, i.e. the order of the spots printed by a spotting robot during the microarray production process. Finally, the obtained scanner results have to be mapped onto the original library data, i.e. the genes.

All steps during the production of microarrays depend on detailed information, but they also generate new data that is more often than not stored in flat files.

As an initial step, a library of oligonucleotides, PCR products, or expressed sequence tags (ESTs) has to be created that contains the information about 96 or 384 well plates and their contents:

```
Plate   X  Y  ProteinID CloneID Description
NMHY-1  A  1  3159924   1616520 ug99a01.r1 Soares mous
NMHY-1  A  2  3159925   1616522 Rattus norvegicus
NMHY-1  B  1  3159958   1616568 ug99e01.r1 Soares mous
NMHY-1  B  2            1616570
NMHY-1  A  3  3159926   1616524 Rattus norvegicus
NMHY-1  A  4  3159927   1616526 ug99a04.r1 Soares mous
NMHY-1  B  3            1616572
NMHY-1  B  4  3159959   1616574 ug99e04.r1 Soares mous
NMHY-1  A  5            1616528
...
```

A layout file that can be obtained from most spotting robots contains the essential mapping table that describes the absolute spot coordinates on a slide and its corresponding content in terms of plate coordinates:

```
Plate #,Plate ID,Well #,Well Col,Well Row,Probe,Replica #,Pin #,SlideAbsX,SlideAbsY
1,AutoGen1,A1,1,1,,0,1,5.500,40.000
1,AutoGen1,A3,3,1,,0,1,5.875,40.000
1,AutoGen1,A5,5,1,,0,1,6.250,40.000
1,AutoGen1,A7,7,1,,0,1,6.625,40.000
1,AutoGen1,A9,9,1,,0,1,7.000,40.000
1,AutoGen1,A11,11,1,,0,1,7.375,40.000
...
```

Finally, all scanner results (including the measured spot positions), their background, and intensity values and various other results are obtained in tab separated lists or as spreadsheet tables.

```
ATF     1.0
4       9
"Type=GenePix results 1.3"
"PixelSize=10"
"Creator=AIM 1.2 mkatzer mussorgsky 24. August 2001"
"FileName=/vol/biochips/share/olaf/olaf1cy3_80_2.Tif
         /vol/biochips/share/olaf/olaf1cy5_80_2.Tif"
"Block" "Column" "Row" "X"  "Y" "Ratio Means" "Ratio SD" "Ratio of Medians"
1       1        1     626  584 0.9706        0.15094    0.58327
1       2        1     1002 562 1.1142        0.46408    0.83645
1       3        1     1381 568 1.002         0.066656   1.5403
1       4        1     1761 572 1.1797        0.054528   1.2084
1       5        1     2131 569 1.0027        0.050338   0.85335
1       6        1     2506 565 0.96684       0.076259   0.95631
...
```

In addition to the data described above, laboratory protocols have to be stored in order to guarantee reproducible microarray results. Comprehensive laboratory protocols are applied at all stages of the experiment. Since the data obtained is not self explanatory, missing or incomplete experimental descriptions and parameter setups can render an experiment almost impossible to reproduce or even interpret. Complete recipes and descriptions of RNA-purification, labeling, washing, and hybridization can be created and managed using a specialized laboratory inventory management system (LIMS). This data has to be linked to measured data from the analysis of the microarrays.

It becomes clear that only widely accepted standards and data exchange formats will allow the compatibility and comparison of different DNA array formats, platforms, and tools. Therefore, an efficient data management is essential thus ensuring the reproducibility of any experiment and supporting the evaluation of the measured results with exchangeable methods. Additional prerequisites that have to be fulfilled for high-throughput transcriptomics are the availability of high quality measurements, exact spot information, and automated methods for the identification and analysis of expression data.

At present, the cDNA-microarray technology is well established and routine pipelines can be set up [Bow99]. Because of the massive parallelism of microarray experiments they are often called high-throughput experiments [BH02]. Due to the large number of genes typically represented on microarrays these are still expensive and quite often there are only few replica spots available on each slide to support the large number of hypotheses that could be generated.

2.2.2. Analysis of expression data

Once a DNA array experiment has been designed and performed, the data must be extracted and analyzed. The identification of similarities and differences in gene expression at varying levels and the exploration of distinctive features between two samples depends on thorough data analysis techniques due to the high dimensional characteristics of microarray experiments.

Experimental data from microarrays have several properties which distinguish them from other biological datasets that measure RNA abundance like for example real-time quantitative PCR [GHW96, HSLW96]. Microarray data is highly prone to variation. A number of sources of systematic and non-systematic variation raises the need for normalization and calibration [KKB03, CAM$^+$99]. Whenever datasets from multiple microarrays have to be compared, normalization is required in order to correct such systematic errors. Robust methods for the analysis of the datasets are needed and have to be performed with respect to the often restricted number of replica available.

Spot detection

As a first step for the analysis of expression data, all spots that are normally arranged in *grids* have to be detected on the microarray. Therefore, a number of image analysis programs provide a semi-automatic spot detection where the user has to roughly adjust a grid before the spots can be located. After manually adjusting the spot positions, their size, and sometimes also their shape, the intensity values are computed for each dye and the background. Finally, another manual inspection of the results can be useful to check the spot detection. Thereby, weak or wrongly detected spots can be marked (*flagged*) and excluded from all further analysis.

Normalization

Before the differential gene expression profiles between two conditions can be obtained for a microarray experiment, it has to be ascertained that the data sets are comparable. Different normalization methods have been developed in the recent past that account for the systematic experimental and biological variations described above. Basically, these methods try to adjust the following variables:

- number of cells in the sample
- total RNA isolation efficiency
- mRNA isolation and labeling efficiency
- hybridization efficiency
- signal measurement sensitivity
- amount of spotted material
- saturation
- “bleaching”

The methods applied for these purposes employ a global or slide-based scaling approach, control-based methods (e.g. reference RNA or housekeeping genes) and pin-dependent normalization (for print-tip groups); but all of them have their drawbacks and advantages. More sophisticated methods perform normalization based on local regression (e.g. *lowess()* function [YDLSa]).

Statistical analysis and other approaches

Although many data analysis techniques have been applied to DNA array data, the field is still evolving and the methods have not yet reached a level of maturity [Zha99]. Gene expression array data can be analyzed on the level of single genes, multiple genes (in terms of common functionalities, interactions, co-regulation, etc.), and on the level of protein networks. The methods applied so far are ranging from simple-minded fold approaches or filters up to probabilistic Bayesian models and supervised or unsupervised clustering strategies. Among numerous statistical methods the well known Students' t-test [DYCS00] is the most frequently used approach to identify significant differentially expressed genes. Other data analysis techniques include self-organizing-maps (SOM) [Koh97], k-means clustering [YHR01], hierarchical clustering [ESBB98], and variants of principal component analysis (PCA) [Jol86]. Although many of these approaches are well suited for detailed analysis of microarray data, it is important to notice that the quality and reliability of the obtained results depend heavily on the design of the experiment (e.g. number of biological or physical replica). All evaluation of DNA array data always has to take into account the biological context and the experimental setup and therefore it is essential to preserve these information.

Since this work is focused on data integration, these and other approaches for the analysis of microarray data are not discussed in detail.

2.3. Proteomics

During the last few years the high-throughput analysis of all proteins of a cell (the proteome) has become more and more important. In this section a very short description of the proteomics approach is presented.

2.3.1. A short introduction

In general, proteomics tries to identify all proteins in an organism, tissue, or cell at a particular time. The often highly dynamic behavior of proteins can be measured by common techniques such as two-dimensional sodiumdodecylsulfate polyacrylamide gel electrophoresis (2D SDS-PAGE) for protein separation and mass spectrometry (MS) which is used for protein identification. Mass spectra obtained for a spot on a 2D gel are then analyzed by various bioinformatics tools in order to identify the protein as illustrated in figure 2.9.

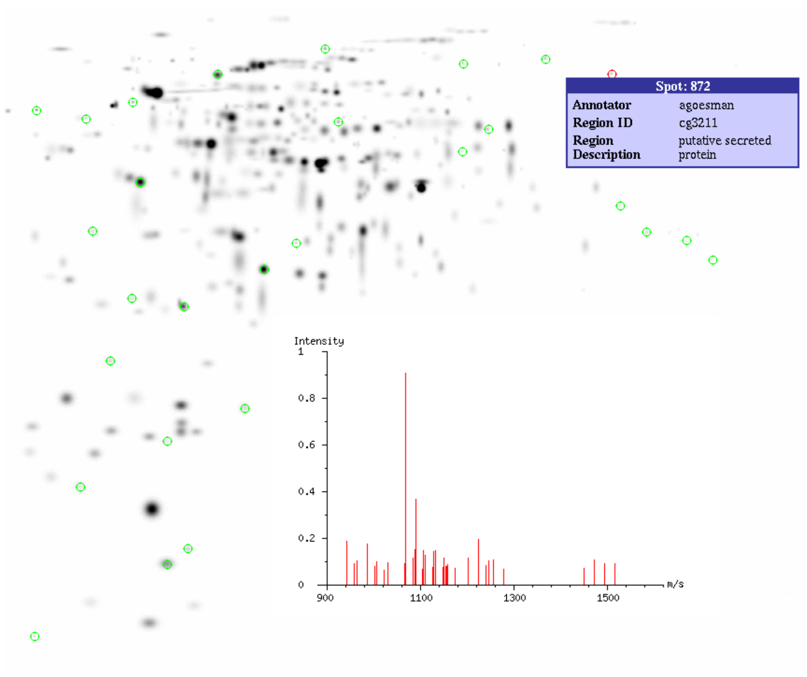


Figure 2.9.: After separating the proteins of a cell they can be analyzed using mass spectrometry. Therefore, the proteins are digested with specific agents that cut the protein sequence-specifically thus producing smaller peptides. The spots on a gel can then be identified by comparing their mass spectra with a database of all proteins of an organism.

Since this work is focused on the analysis of genome and transcriptome data, the proteomics approach is not explained in more detail here. The interested reader can find a review and an introduction into proteomics in [GH02], an overview about existing techniques and systems is also given in [WRB$^+$03].

2.4. Metabolic pathways

The advent of large scale high-throughput methodologies such as microarray and proteome analysis encourages researchers more than ever to gain insight into cellular networks of growing complexity. One major step towards understanding biological systems as a whole is the detailed analysis of enzymes and metabolic pathways. Most of the details described in this section were adapted from [Str91, Leh85, KR97].

2.4.1. Enzymes

Enzymes act as catalysts in every biological system. Microorganisms, plants, and animals control their vital metabolisms quickly, well directed and efficiently by using enzymatic reactions. With the noted exception of some small RNA molecules (ribozymes) the major part of all reactions in a cell is being influenced by proteins [Leh85]. Enzymes catalyze the chemical reactions in a cell which can be denoted together as the intermediary metabolism. The activity of most enzymes is highly specific for a certain substrate and their efficiency surpasses any synthetic catalyst (biochemical reactions can be accelerated up to 10^{12} times). Chemical enzymatic transformations take place in aqueous solution inside a cell under moderate temperature and pH conditions. By catalyzing sequences of reactions, enzymes can build or destroy metabolites, store energy in chemical compounds, or combine simple organic compounds to macro molecules. The activity of enzymes can be regulated by different mechanisms and is often controlled by the concentration of synthesized end products (*feedback inhibition*).

Enzymes like *lactate-dehydrogenase* or *malate-dehydrogenase* that can be found in a variety of molecular forms (*iso-enzymes*) are often adapted to specific tissues and can differ in their catalytic activity. Iso-enzymes play an important role in cell differentiation and for the development of various tissues.

Co-enzymes can be modified temporarily during an enzymatic reaction and revert back again into their original state. They are not specific for a single enzymatic reaction and can therefore interact with a number of enzymes. Co-enzymes can be separated into soluble co-enzymes and prosthetic groups depending on the type of the catalytic reaction.

Since all enzymes are temperature-sensitive proteins that are subject to denaturation and inactivation, the synthesis of required enzymes is vital for each cell to guarantee a continuous metabolism. Additional control of the catalytic activity can be accomplished by increasing or inhibiting the synthesis of specific enzymes dependent on the respective metabolic conditions.

Nomenclature and classification of enzymes

Enzymes can be classified into six main classes (see table 2.1) according to the recommendations of the NC-IUBMB[7] and it is important to note that each class specifies the type of reactions, not the structures of the proteins that catalyze them. Every sufficiently characterized enzyme is also described by a four-digit EC-number such as `EC A.B.C.D` where the prefix EC is an abbreviation for *Enzyme Commission* and each capital letter represents a number specifying the catalytic reaction as follows:

[7]Nomenclature Committee of the International Union of Biochemistry and Molecular Biology

- **A** denotes one of six main class (see table 2.1 for more details).
- **B** defines the chemical structures that are changed by the enzymatic reaction.
- **C** separates different kinds of co-substrates and therefore defines the properties of an enzyme.
- **D** is a serial number characterizing enzymes in more detail if they could not be separated by only assigning the first three groups.[8]

The systematic name of an enzyme consists of three parts (substrate, type of catalyzed reaction, and suffix “ase”) but most enzymes also have additional shorter and more common trivial names. The systematic naming scheme emphasizes the directed role of a catalyst thus explaining different EC-numbers for the forward and reverse reaction. At the time being (November 2003) more than 3700 enzymes have been classified by the enzyme commision. Considering the estimated number of about 25000 natural enzymes [Kin81] it becomes clear that only a very small portion of all enzymes is well known today.

Enzyme classification	Explanation
1 Oxidoreductases	catalyzing oxido-reductions
2 Transferases	transferring a group from one compound to another compound
3 Hydrolases	catalyzing the hydrolysis of various bonds
4 Lyases	cleaving C-C, C-O, C-N and other bonds by other means than by hydrolysis or oxidation
5 Isomerases	catalyzing either racemization or epimerization of a centre of chirality
6 Ligases	catalyzing the joining of two molecules with concomitant hydrolysis of the diphosphate bond in ATP or a similar triphosphate

Table 2.1.: The enzyme comission is responsible for grouping all enzymes into the six main classes and their sub groups.

Enzymatic modes of action *in vivo* and *in vitro*

In any living organism the concentration and activity of an enzyme is adjusted according to the physiological circumstances in a cell. This means that certain conditions highly influence the efficiency, speed, and direction of an enzymatic reaction. In contrast to this, *in vitro* analysis can only model some limited extracts of all enzymatic features at an improper level of detail. This is also due to the fact that enzymatic activity is often measured for constant enzyme concentrations and substrate saturation which is rarely the case *in vivo*.

As a basic principle, enzymatic reactions are reversible which means that the catalytic reaction describes the conversion from a substrate to a product and vice versa. In case of equal

[8]Enzymes from different organisms that catalyze an identical reaction may have the same EC-number although they do not share the same kinetics due to differences in their primary structure.

enzyme efficiency for both directions, the reaction is denoted *reversible* and otherwise called *directed*. Due to this fact many visualizations of metabolic pathways use edges with either one or two arrows to connect the chemical compounds thus reflecting this correlation.

2.4.2. Metabolic pathways

One of the most outstanding characteristic features of any living organism is the ability to assimilate and convert energy from the environment. Together with other substances, this energy can be used for mechanical work or to build complex structures of living cells. For most of these processes, enzymes play an important role and thus the term *metabolism* can be described as the combination of all enzymatic reactions as a whole. Basically all metabolic procedures can be divided into the production of chemical energy (e.g. storage as ATP) and its utilization (e.g. for the synthesis of cellular components or active transport).
In general, four specific functions of metabolism can be distinguished:

1. extraction of chemical energy out of organic nutrients or sunlight
2. conversion of nutrients from the environment into basic modules or pre-stages of macro-molecular components in a cell
3. assembly of these components to proteins, nucleic acids, lipids, polysaccharides and other cellular components
4. production and degradation of biomolecules for specific functions in a cell

Although the intermediary metabolism contains hundreds of different enzymatic reactions, at least the main metabolic pathways are organized in a quite simple manner and show only little differences in most organisms [Leh85].

Catabolism and anabolism

All metabolic activity that can be separated into *catabolism* and *anabolism* passes a sequence of several enzymatic reactions via a number of intermediary products (*metabolites*). The catabolism comprises all energy releasing processes where nutritive molecules (e.g. lipids, proteins) are transformed into smaller and simpler end products (e.g. lactic acid, acetic acid). Anabolism (or *biosynthesis*) can be described as the synthetic energy consuming phase where small and simple building blocks are combined to relatively high molecular components. Both catabolic and anabolic processes take place at the same time although they may be located in different compartments inside a cell. They can also be influenced independently by different enzymes allowing most flexible adaptations of specific metabolic pathways. In conclusion, almost all reactions are connected with each other since numerous cascades of reactions can be build by combining substrate-product relationships. Additionally, the metabolic activity is influenced by protein-protein interactions and metabolite channeling.

The energy cycle in cells

Since all aspects of cellular metabolism are subject to the principle of maximal efficiency, the extend of degradative reactions is not determined by the concentration of available “fuel” but by the momentary demand for energy so that the required amount of ATP is always guaranteed. Metabolic pathways can be regulated by three different mechanisms:

1. The fastest and easiest method is the adaptation via specific, allosteric, or adjustable enzymes.

2. Enzyme concentrations inside a cell can regulate the rates of degradation and biosynthesis.

3. In higher level organisms, specific metabolic activities can be inhibited or stimulated by hormones or via neural pulses.

Measuring metabolic activity

The exploration of metabolic pathways deals with the analysis of the chemical stoichiometry and regulatory mechanisms that control each reaction step. Therefore, three main methods can be applied:

1. Cell-free systems:
   Cell-free preparations extracted from cells or tissues can be used to measure the accumulation of specific metabolic intermediary products after inhibiting or in-activating particular enzymes. Determining the chemical structure of these products can finally help to identify and isolate the corresponding enzymes ideally leading to a complete *in vitro* reconstruction of the metabolic pathway.

2. Genetic defects in metabolisms of auxotrophic mutants:
   Another approach is the production of (viable) genetic mutant strains that cannot synthesize specific enzymes. In this case, an accumulation or excretion of the defect enzyme’s substrate or the absence of its product can be measured. Auxotrophic mutants can be used to analyze catabolic as well as anabolic metabolism.

3. Radioactively labeled compounds:
   Using isotopes of an element for radioactive labeling of specific metabolites is another successfully applied method for the analysis of metabolic pathways. Such marked molecules can be used to determine the speed of the enzymatic reactions or to verify a postulated chain of reactions *in vitro*. This method has also been applied to discover that molecular components in cells and tissues are subject to a *metabolic turnover* which means that all compounds reside in a *steady state* where the speed of the synthesis and degradation are balanced.

CHAPTER 3

Existing systems

This chapter briefly describes numerous existing systems for the annotation and functional analysis of (microbial) genomes. Subsequently, an overview of tools for the storage and analysis of microarray expression data and various approaches for the visualization of metabolic pathways are presented. Furthermore, different schemes for the functional classification of genes are explained. Afterwards, some recently developed approaches for the integration of heterogeneous data are described. Conclusions learned from the analysis of the existing systems are summarized in the last section of this chapter as a basis for the design of a novel approach described later in section 5.1.

For lack of space, only some outstanding features of the most important systems can be outlined here.

3.1. Genome annotation systems

The vast amount of data which has to be evaluated in any whole-genome annotation project require a (partial) automation of the procedure. Most genome annotation systems developed to date perform automated gene prediction using one of the standard gene prediction tools, function prediction based on different tool results, automatic annotation, and sometimes even a more detailed genome comparison with other already annotated organisms.

3.1.1. Comparison of existing tools

As illustrated in table 3.1, a number of genome annotation systems intended for the analysis of prokaryotic and eukaryotic organisms have been designed and presented in the last few years.

Software	**URL**
MAGPIE	*http://www.visualgenomics.ca/*
GeneQuiz	*http://jura.ebi.ac.uk:8765/ext-genequiz/*
Pedant	*http://pedant.gsf.de/*
ERGO	*http://wit.integratedgenomics.com/IGwit/*
PedantPro	*http://www.biomax.de/products/f_prod_Ped.html*
Phylosopher	*http://www.genedata.com/products.php*
BioScout	*http://www.lionbioscience.com/bioscout/*
WIT	*http://wit.mcs.anl.gov/WIT2/*
Artemis	*http://www.sanger.ac.uk/Software/Artemis/*
DAS	*http://www.biodas.org/*
Manatee	*http://manatee.sourceforge.net/*
GenDB-1	*http://gendb.genetik.uni-bielefeld.de/*

Table 3.1.: URLs of the most prominent genome annotation systems. Commercial products are listed as well as open source systems like Manatee or GenDB. As a major drawback, most of these systems do not provide a well structured interface for programmers.

The first generation of genome annotation systems was released in 1996 and consisted of MAGPIE [GS96], GeneQuiz [ABL+99], and Pedant [FAH+01]. These focused primarily on generating human readable HTML documents based on tables and sometimes in-line graphics. A number of good ideas originated from this first generation of genome annotation systems and made their way into today's systems. Examples are the intuitive visualizations or the splitting of results by significance levels to enable comparison of different tools by MAGPIE.

Since then, a second generation of mostly commercial genome annotation systems has been published, including ERGO (Integrated Genomics, Inc.) [OLW+03], Pedant-Pro (successor to Pedant, Biomax Informatics AG), Phylosopher (Gene Data, Inc.), BioScout (successor to GeneQuiz, Lion AG), WIT [OLP+02], and the open source system Artemis [RPC+00]. In particular, MAGPIE, Artemis, and Phylosopher contain extensive visualizations. ERGO also includes multiple genome comparison based annotation strategies. With the exception of Artemis, all systems provide an automatic annotation feature. In general, all systems except ERGO use a variant of “best blast hit” as their fixed, built-in annotation strategy. Only MAGPIE, Artemis, and the newer versions of Pedant allow the integration of expert knowledge through manual annotation. In contrast to most of the other tools, the recently published system Manatee is focused on different annotation strategies that employ and assign functional categories like the Gene Ontology (see section 3.4.5). Another completely different concept for annotating a genome was introduced by the Distributed Annotation System (DAS) [DJD$^+$01] which provides a concept for a decentralized client/server architecture and data exchange via XML data streams.

The substantial commercial interest in the area of genome annotation has led to a situation where, with the noted exception of Artemis, no genome annotation system was in the public domain for a very long time. Therefore, only the source code of Artemis was available for further analysis by the research community. Even in-depth technical information about commercial systems, such as details about the annotation strategy implemented are very hard to obtain. This lack of access is a major hurdle when trying to evaluate these complex systems. Together with the omission of well defined APIs, this prevents the extension of existing systems and is counter-productive for science in this area of research: the best experts in the field have no medium to contribute their experience to the cooperative evolution of better and better annotation systems. Furthermore, none of the systems has a modular architecture that allows a flexible extension at different levels which is essential for the integration of experimental and other higher level data (e.g. transcriptomics or proteomics results).

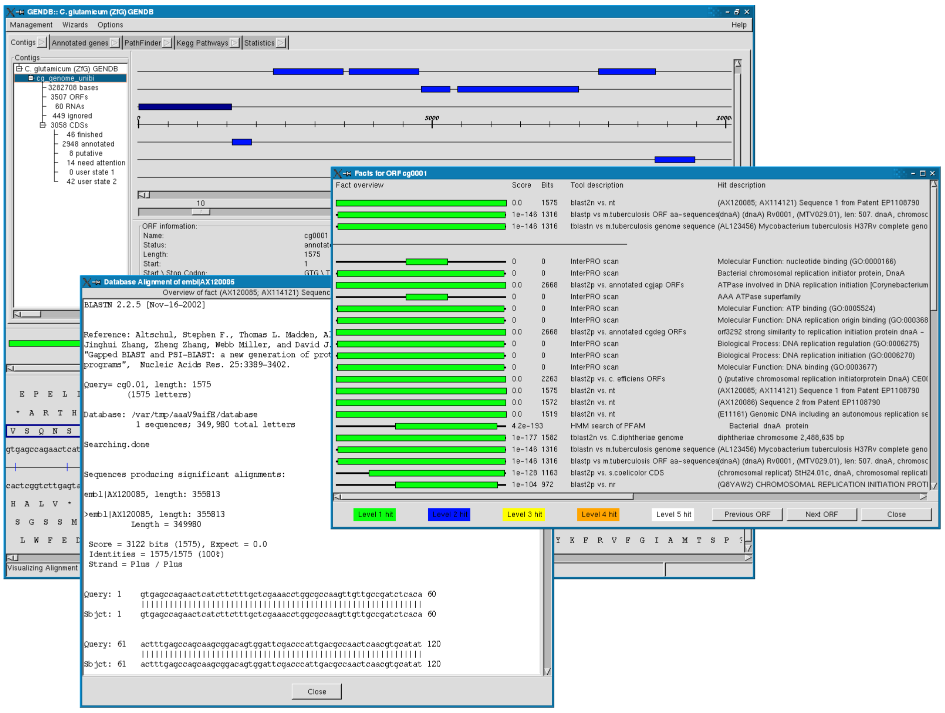
3.1.2. GenDB-1

The need for a well designed and documented open source genome annotation system led to the development of GenDB at the Center for Genome Research, Bielefeld University. GenDB is a flexible and easily extensible system and was first published (version 1.0) in the PhD thesis of Folker Meyer [Mey01]. GenDB-1 was successfully employed for the annotation of more than a dozen novel microbial genomes in world-wide projects. Nevertheless, the system had several drawbacks and limitations that hindered its application and integration.

System overview

The open source genome annotation system GenDB-1 is based on a relational database management system. Contig sequences can be imported and after predicting the coding sequences (CDS), a number of standard bioinformatics tools like BLAST, Pfam, InterPro etc. can be run on these regions as a basis for the functional (manual) annotation. By storing only a minimal required subset of each tool result (e.g. only a short description and a computed score) and recomputing the complete result (e.g. an alignment) on demand, the storage requirements can be reduced enormously.

The software has an object-oriented application programmers interface (API) implemented in Perl [Per] which has been partially generated automatically with O2DBI [Cla02]. The latter creates an object-relational mapping onto SQL tables. GenDB-1 has a web frontend and a Gtk [MKM] user interface (see figure 3.1) that creates dynamic visualizations and can be used for annotating genes based on the computed tool results.



ORF information:

Name:	cg0001
Status:	annotat
Length:	1575
Start:	1
Start \ Stop Codon:	GTG \ T

Fact overview	Score	Bits	Tool description	Hit description
🟩	0.0	1575	blast2n vs. nt	(AX120085; AX114121) Sequence 1 from Patent EP1108790
🟩	1e-146	1316	blastp vs m.tuberculosis ORF aa-sequences	(dnaA) (dnaA) Rv0001, (MTV029.01), len: 507. dnaA, chromosc
🟩	1e-146	1316	tblastn vs m.tuberculosis genome sequence	(AL123456) Mycobacterium tuberculosis H37Rv complete geno
🟩	0	0	InterPRO scan	Molecular Function: nucleotide binding (GO:0000166)
🟩	0	0	InterPRO scan	Bacterial chromosomal replication initiator protein, DnaA
🟩	0.0	2668	blast2p vs. annotated cgjap ORFs	ATPase involved in DNA replication initiation [Corynebacterium
🟩	0	0	InterPRO scan	AAA ATPase superfamily
🟩	0	0	InterPRO scan	Molecular Function: ATP binding (GO:0005524)
🟩	0	0	InterPRO scan	Molecular Function: DNA replication origin binding (GO:000368
🟩	0.0	2668	blast2p vs. annotated cgdeg ORFs	orf3292 strong similarity to replication initiation protein dnaA –
🟩	0	0	InterPRO scan	Biological Process: DNA replication regulation (GO:0006275)
🟩	0	0	InterPRO scan	Biological Process: DNA replication initiation (GO:0006270)
🟩	0	0	InterPRO scan	Molecular Function: DNA binding (GO:0003677)
🟩	0.0	2263	blast2p vs. c. efficiens ORFs	() (putative chromosomal replication initiatorprotein DnaA) CE0(
🟩	0.0	1575	blast2n vs. nt	(AX120085; AX114121) Sequence 1 from Patent EP1108790
🟩	0.0	1572	blast2n vs. nt	(AX120086) Sequence 2 from Patent EP1108790
🟩	0.0	1519	blast2n vs. nt	(E11161) Genomic DNA including an autonomous replication se
🟩	4.2e-193		HMM search of PFAM	Bacterial dnaA protein
🟩	1e-177	1582	tblast2n vs. C.diphtheriae genome	diphtheriae chromosome 2,488,635 bp
🟩	1e-146	1316	tblastn vs m.tuberculosis genome sequence	(AL123456) Mycobacterium tuberculosis H37Rv complete geno
🟩	1e-146	1316	blastp vs m.tuberculosis ORF aa-sequences	(dnaA) (dnaA) Rv0001, (MTV029.01), len: 507. dnaA, chromosc
🟩	1e-128	1163	blast2p vs. s.coelicolor CDS	(chromosomal replicat) StH24.01c, dnaA, chromosomal replicati
🟩	1e-104	972	blast2p vs. nr	(Q8YAW2) CHROMOSOMAL REPLICATION INITIATION PROTI

```
BLASTN 2.2.5 [Nov-16-2002]


Reference: Altschul, Stephen F., Thomas L. Madden, Al
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J.
"Gapped BLAST and PSI-BLAST: a new generation of prot
programs",  Nucleic Acids Res. 25:3389-3402.

Query= cg0.01, length: 1575
         (1575 letters)

Database: /var/tmp/aaaV9aifE/database
           1 sequences; 349,980 total letters

Searching.done


Sequences producing significant alignments:

embl|AX120085, length: 355813

>embl|AX120085, length: 355813
          Length = 349980

 Score = 3122 bits (1575), Expect = 0.0
 Identities = 1575/1575 (100%)
 Strand = Plus / Plus


Query: 1    gtgagccagaactcatcttctttgctcgaaacctggcgccaagttgttgccgatctcaca 60
            ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 1    gtgagccagaactcatcttctttgctcgaaacctggcgccaagttgttgccgatctcaca 60


Query: 61   actttgagccagcaagcggacagtggattcgacccattgacgccaactcaacgtgcatat 120
            ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 61   actttgagccagcaagcggacagtggattcgacccattgacgccaactcaacgtgcatat 120
```

Figure 3.1.: Screenshot of the GenDB-1 Gtk graphical user interface. The observations computed by different bioinformatics tools are listed for a selected region and the dynamically recomputed BLAST alignment is displayed for one of them.

The system features the concept of wizards which are software agents that automate complex repetitive tasks (e.g. ORF editor, frame-shift correction, contig update, etc.). In addition to a search interface, the software can generate statistical plots and includes a virtual 2D gel. The data contained in GenDB can be exported into widely used output formats like FASTA, EMBL, GFF (genome feature format[1]), and others. An integrated pathway module supports the analysis and visualization of annotated enzymes based on the KEGG [KG00] metabolic pathways and the PathFinder [GHM+02] software.

[1] *http://www.sanger.ac.uk/Software/formats/GFF*

Limitations of GenDB-1

In GenDB-1, the only built-in types of regions are contigs, supercontigs, and ORFs that can be handled by the GenDB system. Although an EMBL feature can be assigned to each ORF for further classification, it is clear that the lack of comprehensively defined region types is a major hurdle when trying to completely describe all features of a genome (promotors, tRNAs, rRNA operons, repeats, IS elements, etc.). Since all kinds of tools and their observations are represented in single database tables, storing individual tool settings or results is uncomfortable and complicated. For example, all tools have to use the `score` field of the fact table to store their (numerical) result that can be a floating point value (e.g. for SignalP) or an expectation value (e.g. for BLAST). Additionally, the configuration and computation of different bioinformatics tools is quite complicated and is therefore only a task for experienced users. Since GenDB was designed as an open platform for further extension and is continuously developed, the restriction to a Perl API as the only programmers interface is another important disadvantage that limits using the software. The system has almost no project or user management support and thus provides only inadequate access control (only *annotators* are distinguished from other users).

Using a dedicated database backend, an API, and separate frontends, the GenDB architecture itself has been modularized sufficiently for the purpose of genome annotation. Compared to other systems that do not provide such a layered architecture, GenDB seems to be best suited as a core module for handling all issues involved in the annotation of whole genomes. Nevertheless, the system is missing a general concept for the integration of additional components (e.g. for transcriptome or proteome data analysis). Instead, a module for visualizing and browsing metabolic pathways was integrated directly into the Gtk frontend. It is clear that – in particular due to the high expectations based on promising results from high-throughput experiments – the availability of all features just mentioned is vital for the future success of any genome annotation system.

3.2. Microarray analysis

The properties mentioned in section 2.2 impose new challenges for the storage and evaluation of large scale microarray data and demand for well designed systems that support robust, efficient and reliable bioinformatics methods. As the analysis of microarray data is a constantly evolving field and new algorithms are permanently being published, such a system should provide flexible mechanisms to exchange or add such methods.

3.2.1. Storage and analysis of expression data

At present, there already exists a variety of commercial and non-commercial software applications that aim at the analysis of microarray data and the list given in table 3.2 is by no means complete.

Software	**URL**
ImaGene	*http://www.biodiscovery.com/imagene.asp*
GeneSight	*http://www.biodiscovery.com/genesight.asp*
J-Express Pro	*http://www.molmine.com/*
J-Express	*http://www.ii.uib.no/~bjarted/jexpress/*
Cluster	*http://rana.lbl.gov/EisenSoftware.htm*
GeneX-Lite	*http://www.ncgr.org/genex/*
QuantArray	*http://lifesciences.perkinelmer.com/*
Base	*http://base.thep.lu.se/*
maxd	*http://www.bioinf.man.ac.uk/microarray/maxd/*
Nomad	*http://ucsf-nomad.sourceforge.net/*

Table 3.2.: URLs for microarray analysis software. Some of the systems listed above focus on the image acquisition and simple analysis while others provide comprehensive collections of methods for the evaluation of the measured spot intensities.

In the commercial segment, there are for example ImaGene and GeneSight from Biodiscovery and J-Express Pro from Molmine. The predecessor of J-Express Pro named J-Express [DJ01] is available free of charge. Other examples of software that is at least licensed free of charge to academics is Eisen's well known Cluster software [ESBB98], GeneX-Lite (the successor version of GeneX), Base [STVC$^+$02], maxd, and Nomad.

While ImaGene (see figure 3.2) is mainly an image analysis tool with very restricted data analysis capabilities, GeneSight, J-Express (Pro) and Cluster focus on the analysis of measured expression data and methods like clustering and visualizations for the computed results.

All these tools operate on flat files for data input and output and a plain file with the measured data has to be imported each time. The systems mentioned above neither support a common shared data repository nor do they provide structured storage of microarray data and experimental setups. The omission of well defined and open interfaces of most commercial systems is also a major hurdle when trying to integrate user defined new methods.

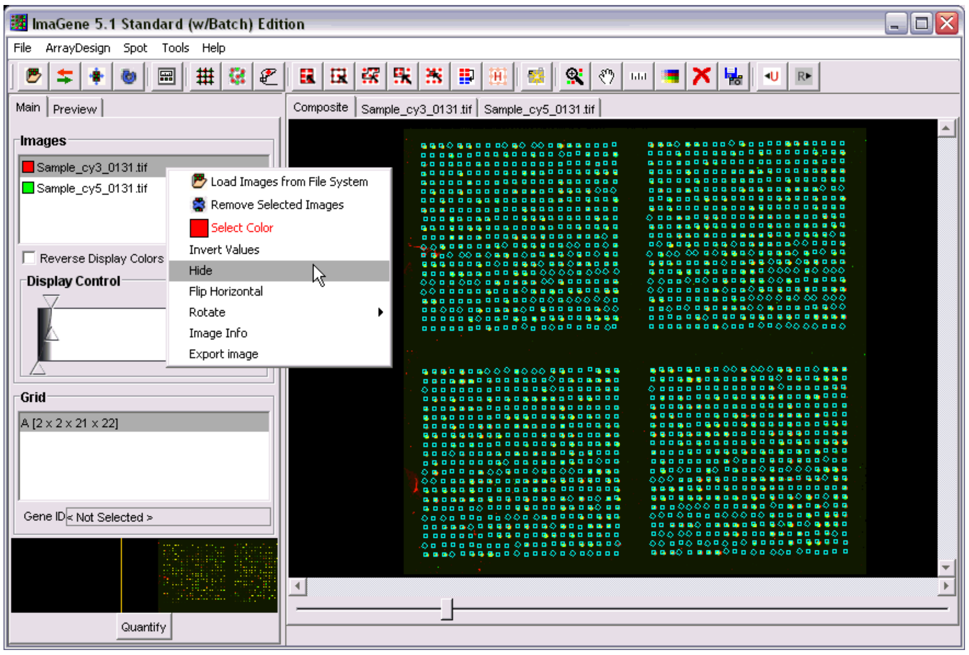


Figure 3.2.: Screenshot of the ImaGene image analysis software. After loading the scanned images, grids have to be arranged that roughly fit onto the spotted slide layout before the spot detection can be started.

In contrast to this, the open source systems GeneX, Base, maxd, and Nomad have been designed as platforms that feature most aspects of microarray data storage and analysis. They use a relational database management system (RDBMS) as their storage backend and provide web frontends or graphical user interfaces (GUIs) to the data. All of these tools provide data normalization and data analysis techniques. While Base and Nomad provide a web-based user interface, maxd and GeneX consist of several applications for data upload and analysis. The major drawback of these systems is that they do not provide a structured interface that allows a bidirectional data exchange between different applications. In addition to that, the extension of most systems is often impossible since no API is available. Thus, advanced features like the integration with genome annotation systems cannot be easily implemented except by simple hyperlinks.

3.2.2. MIAME and MAGE-ML

Since microarray experiments do not only produce large amounts of data but also require a number of experimental steps and procedures that should be protocolled consistently, standards for exchanging and storing this information have to be defined. The MIAME (Minimum Information About a Microarray Experiment) format [BHQ+01] has been defined by the MGED (Microarray Gene Expression Data) Society as a standard for microarray data annotation and exchange. The goal of this standard is to outline the minimum information required to interpret unambiguously and potentially reproduce and verify an array based gene expression monitoring experiment. Although details for particular experiments may be different, MIAME aims to define the core that is common to most experiments.

A major objective of MIAME is to guide the development of microarray databases and data management software. While MIAME is not a formal specification, but a set of guidelines, the MAGE [MGEb] format has been developed as a standard microarray data model and exchange format. MAGE is able to capture information specified by MIAME and recently became an Adopted Specification of the OMG standards group[2]. Many organizations, including Agilent, Affymetrix, and Iobion, have contributed ideas to MAGE. Although MIAME concentrates on the content of the information and should not be confused with a data format, it also tries to provide a conceptual structure for microarray experiment descriptions as a basis for the MAGE format. The MAGE group aims to provide a standard for the representation of microarray expression data that should facilitate the exchange of microarray information between different data systems. Currently, this is done through the OMG (Object Management Group) by the establishment of a data exchange model (MAGE-OM: Microarray Gene Expression-Object Model) and a data exchange format (see MAGE-ML, Microarray Gene Expression-Markup Language[3] for the full specification and for the Document Type Definition (DTD)[4]) for microarray expression experiments. MAGE-OM has been modelled using the Unified Modelling Language (UML) and MAGE-ML has been implemented using XML (eXtensible Markup Language). The additional MAGEstk (or MAGE Software Toolkit) is a collection of packages that act as converters between MAGE-OM and MAGE-ML under various programming platforms.

Microarray Gene Expression Markup Language (MAGE-ML) is a language designed to describe and communicate information about microarray based experiments. MAGE-ML can describe microarray designs, microarray manufacturing information, microarray experiment setup and execution information, gene expression data and data analysis results. Since the structure of MAGE-ML is not simple, user-friendly tools are currently being developed that support the creation of MAGE-ML documents.

For reasons of simplicity and readability, related classes are grouped together into packages (Experiment, Bioassay, ArrayDesign, DesignElement, Biomaterial, BioAssayData, QuantitationType, Array, Bioevent, Protocol, AuditAndSecurity, Description, and HigherLevelAnalysis) since the complete MAGE-OM is quite too large to be represented on a single diagram.

[2] *http://www.mged.org/mage*
[3] *http://cgi.omg.org/cgi-bin/doc?lifesci/01-10-01*
[4] *http://cgi.omg.org/cgi-bin/doc?lifesci/01-11-02*

3.3. Databases and visualizations for metabolic pathways

Pathway databases are widely used to store and model the biochemical relationships of more or less well-known metabolic pathways. Currently, more than a 10 databases exist that offer information about biochemical pathways, metabolic reactions, enzymes, and the genes encoding such functions at different levels of detail and complexity.

Database	**URL**
Biochemical Pathways	*http://www.expasy.org/cgi-bin/search-biochem-index/*
KEGG	*http://www.genome.ad.jp/kegg/kegg.html*
EcoCyc	*http://biocyc.org/ecocyc/*
MetaCyc	*http://biocyc.org/metacyc/*
WIT	*http://wit.mcs.anl.gov/WIT2/*
Biocatalysis/Biodegradation	*http://umbbd.ahc.umn.edu/index.html*
BioPath	*http://biopath.fmi.uni-passau.de/index.html*
PathDB	*http://www.ncgr.org/pathdb/*
PathFinder	*http://pathfinder.genetik.uni-bielefeld.de/*
ENZYME	*http://www.expasy.ch/sprot/enzyme.html*
BRENDA	*http://www.brenda.uni-koeln.de/*

Table 3.3.: Some URLs of metabolic databases. While KEGG provides manually drawn general pathways, systems like EcoCyc or BioPath incorporate dynamic visualizations.

Some of the databases described in table 3.3 focus on static (manually drawn) representations (e.g. KEGG) whereas other systems support dynamic visualizations based on graph drawing algorithms (e.g. BioPath [FPR$^+$02], *PathFinder* [GHM$^+$02]). In addition to general ressources (e.g. KEGG), various organism specific databases like EcoCyc [KRS$^+$00] concentrate on the metabolism of selected species. In addition to the pathway databases described above, the ENZYME [Bai00] and BRENDA [SSS95] databases contain detailed information about characterized enzymatic reactions. Both databases support a mapping onto the EC number classification and contain comprehensive descriptions of the catalyzed reactions and involved chemical compounds (e.g. substrate, product, cofactors). Additional links to sequence databases like SwissProt provide useful information about annotated genes encoding such enzymes.

3.3.1. Boehringer Mannheim wall charts

The Boehringer Mannheim wall charts have been published as the first comprehensive archive of metabolic information ([Mic99] and [Mic92]). The online version (see URL for Biochemical Pathways) also provides zoomable views and clickable image maps with links to the ENZYME database.

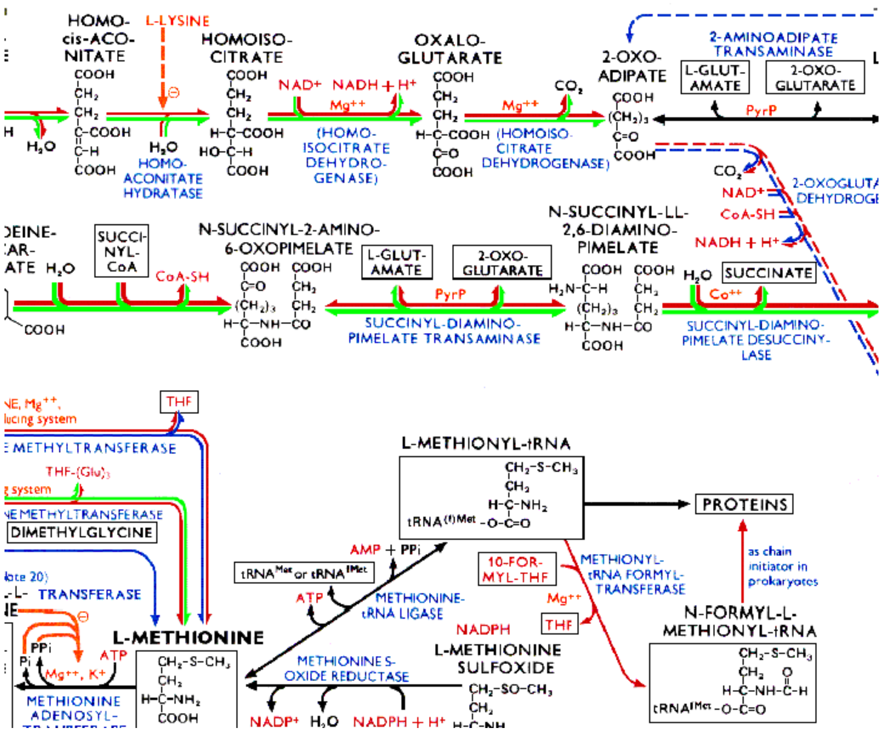

HOMO-cis-ACONITATE
L-LYSINE
HOMOISOCITRATE
OXALOGLUTARATE
2-OXOADIPATE
2-AMINOADIPATE TRANSAMINASE
L-GLUTAMATE
2-OXOGLUTARATE
NAD+
NADH + H+
Mg++
CO2
PyrP
HOMOACONITATE HYDRATASE
(HOMOISOCITRATE DEHYDROGENASE)
2-OXOGLUTARATE DEHYDROGENASE
N-SUCCINYL-2-AMINO-6-OXOPIMELATE
N-SUCCINYL-LL-2,6-DIAMINOPIMELATE
SUCCINYL-CoA
CoA-SH
SUCCINATE
Co++
SUCCINYL-DIAMINOPIMELATE TRANSAMINASE
SUCCINYL-DIAMINOPIMELATE DESUCCINYLASE
THF
METHYLTRANSFERASE
THF-(Glu)3
DIMETHYLGLYCINE
TRANSFERASE
L-METHIONINE
L-METHIONYL-tRNA
PROTEINS
as chain initiator in prokaryotes
10-FORMYL-THF
METHIONYL-tRNA FORMYLTRANSFERASE
N-FORMYL-L-METHIONYL-tRNA
METHIONINE-tRNA LIGASE
AMP + PPi
ATP
METHIONINE S-OXIDE REDUCTASE
L-METHIONINE SULFOXIDE
NADPH
NADP+
NADPH + H+
METHIONINE ADENOSYLTRANSFERASE

Figure 3.3.: Excerpt of the Boehringer Mannheim wall charts. These manually drawn maps were originally published in a book and as posters. Recently, they were also made available online as interactive pathways charts.

The Boehringer wall charts represent very detailed and colorful views onto the metabolic pathways, including the chemical formula of many compounds (see figure 3.3). Subways of a pathway that are only represented in specific organisms are highlighted as well as special disease related enzymes. The complexity of the displayed information and the level of detail is a major drawback that complicates using these charts, especially for unexperienced users.

3.3.2. KEGG

The KEGG database [KG00] (Kyoto Encyclopedia of Genes and Genomes) contains more than 100 different metabolic pathways derived from literature (sources: *Metabolic Maps* [Nis97], *Boehringer wall charts* [Mic92], *Enzyme Handbook* [SSS95]). A single masterpathway thus represents a consensus of the known pathways from different organisms. Figure 3.4 shows an image of the KEGG Lysine biosynthesis pathway that contains less details in contrast to the Boehringer wall charts.

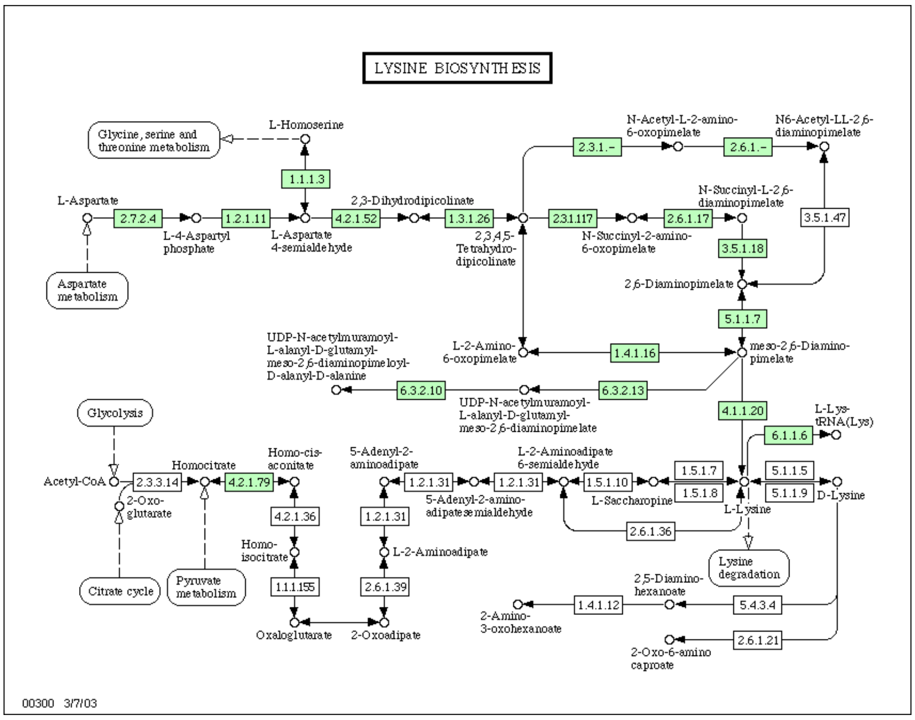


Figure 3.4.: Diagram of the KEGG Lysine biosynthesis pathway for *C. glutamicum*. All enzymes of a pathway that were annotated in the selected genome are indicated by green boxes.

Organism specific metabolic routes are displayed by highlighting the encoded enzymes in a masterpathway which simplifies the recognition of implemented subways. KEGG also links enzymes and compounds to databases with additional information and correlates each enzyme with other genomes it appears in. The main disadvantage of the KEGG metabolic pathways is the static nature of the manually drawn images that makes it difficult to further analyze the metabolism of any organism.

3.3.3. EcoCyc/MetaCyc

EcoCyc [KRS^{+}00] is a bioinformatics database that describes the genome and the biochemical machinery of *E. coli*. The long-term goal of the project is to describe the molecular catalog of the *E. coli* cell as well as the functions of each of its molecular parts to facilitate a system-level understanding of *E. coli*. EcoCyc is an electronic reference source for *E. coli* biologists and for biologists who work with related microorganisms. Scientists can use the Pathway/Genome Navigator user interface within EcoCyc to visualize the layout of genes within the *E. coli* chromosome, of an individual biochemical reaction, or of a complete biochemical pathway (with compound structures displayed). The navigation capabilities of the software allow a user to move from a display of an enzyme to a display of a reaction that the enzyme catalyzes, or to the gene that encodes the enzyme (see figure 3.5). The interface also supports a variety of queries, such as generating a display of the map positions of all genes that code for enzymes within a given biochemical pathway. As well as being used as a reference source to look up individual facts, EcoCyc supports computational studies of the metabolism, such as design of novel biochemical pathways for biotechnology, studies of the evolution of metabolic pathways, and simulation of metabolic pathways. EcoCyc is also used for computer-based education in biochemistry.

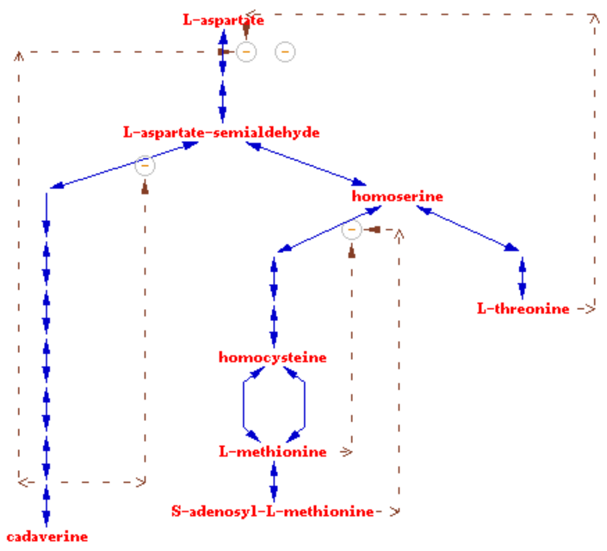


Figure 3.5.: Partial biosynthesis pathway for L-Methionine as displayed by the EcoCyc system. The user can zoom in and out thus choosing the desired level of detail and complexity.

In contrast to the EcoCyc database, the MetaCyc metabolic pathway database contains pathways from over 150 different organisms. MetaCyc describes metabolic pathways, reactions, enzymes, and substrate compounds. The MetaCyc data was gathered from a variety of literature and online sources and contains citations to the source of each pathway. MetaCyc is a collaborative project between SRI International, the Carnegie Institution, and Stanford University. MetaCyc employs the same database schema as the EcoCyc database. The pathways within MetaCyc are annotated at different levels of detail. Some pathways include objects for each enzyme in the pathway, and the pathway and enzyme include extensive commentary and literature citations. Other pathways consist of a sequence of reactions only, with more details to be added in a future version. Unlike EcoCyc, MetaCyc does not provide genomic data. Both systems use the Pathway Tools for data retrieval and visualization. Originally, MetaCyc was initialized with all metabolic pathways of EcoCyc but many additional pathways were then added to the database. The majority of the pathways within MetaCyc are from micro-organisms. Each MetaCyc pathway has a slot (attribute) called “species distribution” that lists the one or more species in which the experimental literature reports that this pathway has been observed. The fact that a given species is not listed in the species distribution of a pathway does not necessarily imply that the pathway is not present in that species, but only that no report of its presence has yet been found in the literature. MetaCyc contains all enzyme-catalyzed reactions that have been assigned EC numbers by the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology (NC-IUBMB). In addition to that, MetaCyc also contains over 400 enzyme-catalyzed reactions that have not yet been assigned an EC number.

3.3.4. The MPW/EMP or WIT database

The Enzymes and Metabolic Pathways database (EMP) [Kar98] contains information about enzymes and their occurrence in metabolic pathways derived from literature. The Metabolic Pathways Database (MPW) [SGMS98] has been derived from the EMP system and is the basis for the reconstruction of metabolic pathways implemented in the WIT system [OLP$^+$02]. The WIT (What Is There) system has been designed to support comparative analysis of sequenced genomes and to generate metabolic reconstructions based on chromosomal sequences and metabolic modules from the EMP/MPW family of databases. This system contains data derived from about 40 completed or nearly completed genomes. Sequence homologies, various ORF-clustering algorithms, relative gene positions on the chromosome, and placement of gene products in metabolic pathways (metabolic reconstruction) can be used for the assignment of gene functions and for the development of overviews of genomes within WIT. The integration of a large number of phylogenetically diverse genomes in WIT facilitates the understanding of the physiology of different organisms.

3.3.5. Biocatalysis/Biodegradation database (UM-BBD)

The University of Minnesota Biocatalysis/Biodegradation Database (UM-BBD) provides curated information on microbial catabolism and related biotransformations, primarily for environmental pollutants. It contains data on microbial biocatalytic reactions and biodegradation pathways for primarily xenobiotic, chemical compounds. The goal of the UM-BBD is to provide information on microbial enzyme-catalyzed reactions that are important for biotechnology. Currently, it consists of over 130 metabolic pathways, 800 reactions, 750 compounds, and 500 enzymes. In the past two years, it has increased its breadth to include more examples of microbial metabolism of metals and metalloids. Furthermore, the types of information the database includes were expanded to contain microbial biotransformations of and binding interactions with many chemical elements. It has also increased the number of ways in which this data can be accessed (mined). Structure-based searching was added for exact matches, similarity, or substructures. Analysis of UM-BBD reactions has lead to a prototype of a guided pathway prediction system. Guided prediction means that the user is shown all possible biotransformations (see figure 3.6) at each step and guides the process to its conclusion. Mining the UM-BBD's data provides a unique view into how the microbial world recycles organic functional groups.[EHKW03]

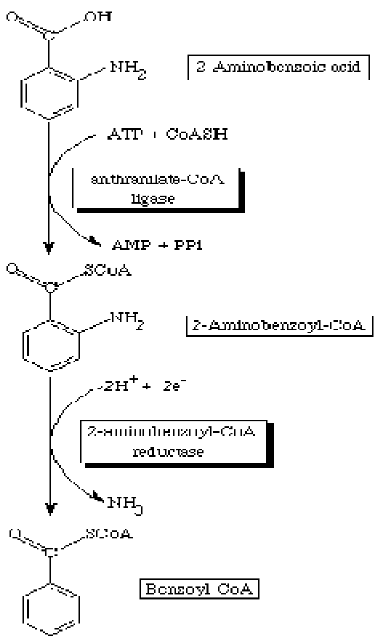


Figure 3.6.: Visualization of a pathway from the Biocatalysis/Biodegradation database. In contrast to other systems, the UM-BBD concentrates on the detailed analysis of chemical reactions and on the prediction of novel pathways.

3.3.6. BioPath

The BioPath project was motivated by the Boehringer Mannheim wall charts in order to generate dynamic views to specific problems. The system was also designed to keep pace with the enormous expansion of the scientific knowledge in biochemistry. BioPath is a platform for a convenient electronic access to the biochemical knowledge. It provides information from different views and in distinct levels of detail (see figure 3.7). The dynamically created visualizations are based on graph drawing algorithms and support specialized layout algorithms for open and closed circles. As an additional feature, the system tries to layout a pathway preserving the mental map, e.g. small changes of the data imply only small changes of the displayed path.

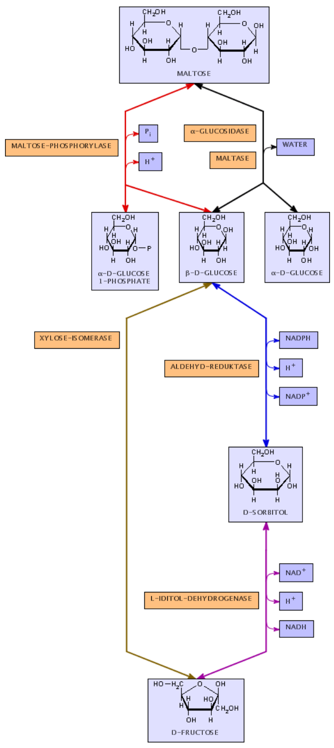


Figure 3.7.: Visualization of the maltose-fructose pathway as provided by the BioPath system. This project was especially focused on optimizing the layout and visualization of dynamically drawn pathway maps.

BioPath provides powerful search mechanisms for substances and reaction networks and allows comparisons of reaction networks. It gives access to different types of information on enzymes, reactions, and metabolism and automatically computes visual representations of complex reaction nets. BioPath is easy to use and can be updated and extended. Current activities are the improvement and adaptation of the major components of BioPath. Since the BioPath system has been bought by the LION Bioscience AG, it is no longer available to the public free of charge.

3.3.7. PathDB

PathDB (unpublished) is both a data repository and a system for building, visualizing, and comparing cellular networks. PathDB version 2 is based on an abstract approach such that modelling of data is not only restricted to metabolism but also supports signal cascade data, gene regulatory data, protein-protein interaction data, and protein-small molecule binding data. New user defined pathways can be added to the system and visualized with a PathwayViewer. PathDB features the possibility to merge different networks together and to view them as one network. In addition to the metabolic data, the system contains the Gene Ontology database and a query interface. PathDB can be integrated with the ISYS environment[5] (see also section 3.5.6) and the maxdView gene expression visualization component[6] can be used as a plug-in to synchronize clusters from gene expression data with the PathwayViewer. The software is available free for non-commercial use as a client or complete server installation.

3.3.8. PathFinder

PathFinder[GHM$^+$02] is a tool for the dynamic visualization of metabolic pathways based on annotation data. Pathways are represented as directed acyclic graphs, graph layout algorithms accomplish the dynamic drawing and visualization of the metabolic maps. A more detailed analysis of the input data on the level of biochemical pathways helps to identify genes and detect improper parts of annotations. As an RDBMS based Internet application *PathFinder* reads a list of EC-numbers or a given annotation in EMBL- or Genbank-format and dynamically generates pathway graphs.

3.4. Functional classification

As outlined in section 2.1.4, a simple comparison of different organisms is often quite difficult to obtain without consistent naming conventions being used for the (an)notation of genes and their products. Additionally, the lack of unique gene identifiers does not only complicate cross-linking of different experimental results but also makes it almost impossible to

[5] *http://www.ncgr.org/isys/*
[6] *http://www.bioinf.man.ac.uk/microarray/resources.html*

integrate inhomogeneous data from different sources (e.g. to correlate genome annotations and expression profiles). A number of functional classification schemes that have been developed in the past few years are presented here showing different approaches that try to find a remedy for the above mentioned problems.

Various schemes exist that can be used for the functional classification of genes and their protein products. Most of these categories consist of simple lists that define categories and describe their functionality. With the exception of COG [TNG$^+$01] and Gene Ontology [The00] classifications it is also difficult to assign such categories automatically. The following sections describe the classification via Gene Ontologies and other categories in more detail. In particular, the concept of defining Gene Ontologies for the assignment of functions seems to represent the most sophisticated approach developed to date.

3.4.1. Monica Riley categories

The first extensive functional classification scheme (see table 3.4) for gene products was devised in 1993 by Monica Riley [Ril93] to catalogue the 1171 *Escherichia coli* genes known at that time. This was some 4 years before the complete genome for *E. coli*, currently estimated to have approximately 4,300 genes, was sequenced and annotated with the slightly modified functional categories of Fred Blattner [BPB$^+$97]. Several updated versions of the classification scheme have been published and can be found for example in GenProtEC [Ril98] and EcoCyc [KRS$^+$00].

3.4.2. TIGR roles

The Institute for Genomic Research (TIGR[7]) maintains a fairly extensive list of human and many other organism nomenclatures. In addition to unique accession numbers for each database entry (e.g. genes, ESTs), the institute has also established a set of standard categories (*TIGR roles*) adapted from the Monica Riley categories that can be used for functional assignments.

3.4.3. The hierarchy of EMBL features

The GenBank[8], EMBL[9], and DDBJ[10] nucleic acid sequence databanks use tables of sites and features to describe the roles and locations of higher order sequence domains and elements within the genome of an organism. In February, 1986, GenBank and EMBL began a

[7] *http://www.tigr.org/*
[8] *http://www.ncbi.nlm.nih.gov/*
[9] *http://www.ebi.ac.uk/embl/*
[10] *http://www.ddbj.nig.ac.jp/*

Amino acid biosynthesis
Purines, pyrimidines, nucleosides, and nucleotides
Fatty acid and phospholipid metabolism
Biosynthesis of cofactors, prosthetic groups, and carriers
Central intermediary metabolism
Energy metabolism
Transport and binding proteins
DNA metabolism
Transcription
Translation
Regulatory functions
Cell envelope
Cellular processes
Other categories
Hypothetical

Table 3.4.: The original main functional categories as described by Monica Riley. These roles were initially created and adapted for the annotation of *E. coli*.

collaborative effort (joined by DDBJ in 1987) to devise a common feature table format and common standards for annotation practice[11]. The overall goal of the feature table design is to provide an extensive vocabulary for describing features in a flexible framework. The hierarchical structure contains features to describe e.g. biological functions, interactions, different effects on sequences, repeats, and structural information and allows the three databases to exchange data on a daily basis.

3.4.4. EC-numbers

As already outlined in section 2.4.1, the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology (IUBMB[12]) maintains a catalog of known enzymes and their functions. Enzyme nomenclature is based on the reactions that are catalyzed, and not the genes that make up the enzymes, or the protein structure of those enzymes [KKB03]. As a consequence this naming scheme is not accurate for the description of genes in particular because even some enzymes may consist of several subunits. Enzyme databases are available online and can be searched using for example the Expert Protein Analysis System (EXPASY[13]) or the Kyoto Encyclopedia of Genes and Genomes (KEGG[14]).

[11] *http://www.ebi.ac.uk/embl/Documentation/FT_definitions/feature_table.html*
[12] *http://www.chem.qmul.ac.uk/iubmb/*
[13] *http:ca.expasy.org/enzyme/*
[14] *http://www.genome.ad.jp/kegg/*

3.4.5. Gene Ontology

The Gene Ontology classification scheme [The00] has been developed by the Gene Ontology Consortium as "a gold standard for the unification of systematic biology" (Michael Ashburner, ISMB 2002). It provides a dynamic controlled vocabulary that can be applied to all organisms. The three organizing principles separate all assignments into molecular function, biological process, and cellular component. Related characterizations can be expressed in terms of *isa* and *part-of* relationships. All Gene Ontology data is represented as a directed acyclic graph (DAG) and each term is exactly defined and described by a GO number, textual explanations, and references from the literature (e.g. homoserine dehydrogenase is represented by the GO accession number GO:0004412 and is exactly defined as "catalysis of the reaction: L-homoserine + NADP$^+$ = L-aspartate 4-semialdehyde + NADPH + H$^+$").

All GO terms and their relationships are represented by the graph and stored in a relational database as displayed in figure 3.8. Monthly releases of the GO database are available either as an XML file or as a MySQL database dump[15].

In addition to the GO terms itself, the database contains mappings to SwissProt, EC numbers, EGAD, GenProtEC (Monica Riley), TIGR roles, InterPro, and MIPS Funcat.

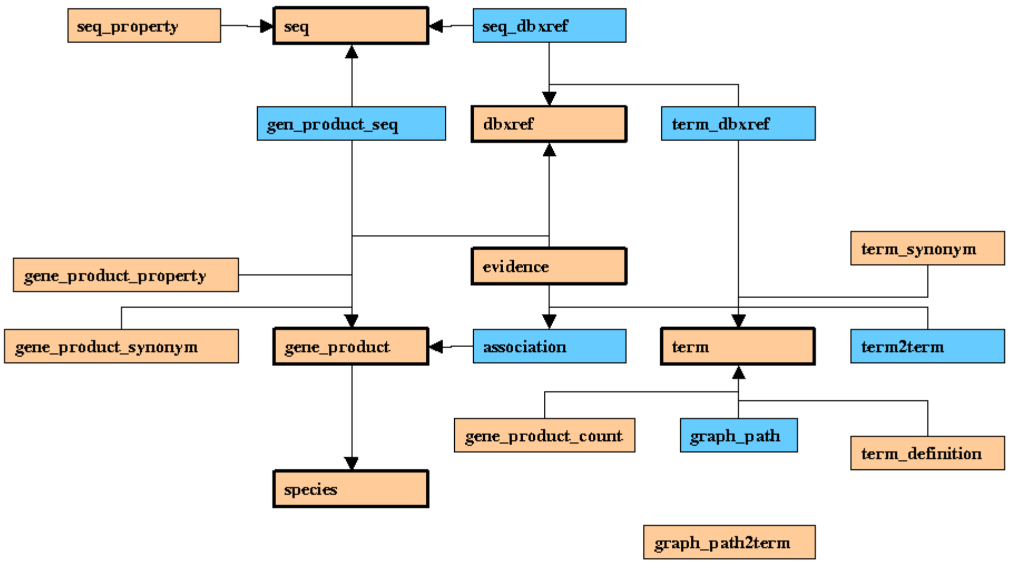


Figure 3.8.: Schematic dependencies diagram of the GO database. Beyond the GO terms themselves, gene products, sequences, species information, and other related data are stored in the database.

[15] *http://www.godatabase.org/dev/database/*

GO-Browsers

Several GO browsers with different features have been developed to access the data and search for GO terms. Table 3.5 summarizes the most prominent tools available for browsing the Gene Ontologies.

Browser	**URL**
AmiGO	*http://www.godatabase.org/cgi-bin/go.cgi*
MGI GO	*http://www.informatics.jax.org/searches/GO_form.shtml*
QuickGO	*http://www.ebi.ac.uk/ego/*
EP GO	*http://ep.ebi.ac.uk/EP/GO*
GoFish	*http://llama.med.harvard.edu/~berriz/GoFishWelcome.html*
GenNav	*http://etbsun2.nlm.nih.gov:8000/perl/gennav.pl*

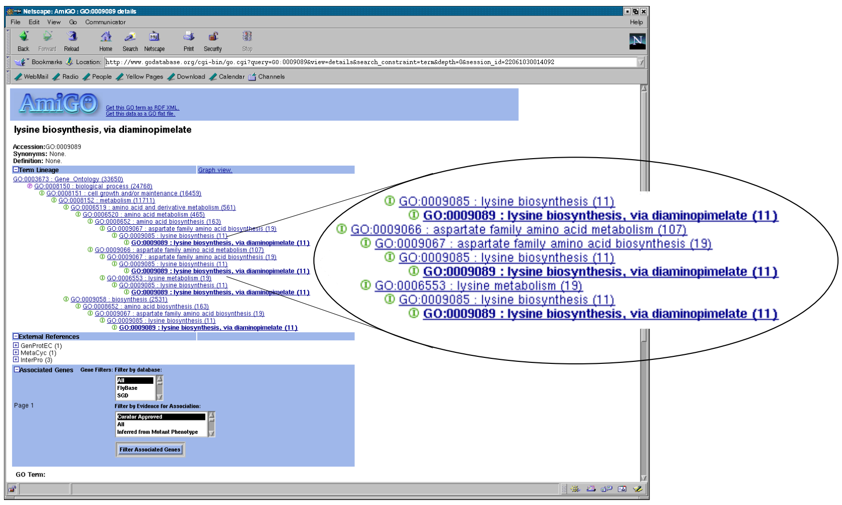
Table 3.5.: URLs of selected GO browsers. A complete list of available browsers can be found on the homepage of the Gene Ontology project.

The GO database

Most of the browsers described above are web-based frontends (see figure 3.9) with a fixed functionality. Although these web frontends are comfortable enough for many purposes, the integration capabilities of such browsers are quite limited.

3.4.6. COGs

The database of Clusters of Orthologous Groups of proteins (COGs) [TNG$^+$01] was delineated by comparing protein sequences encoded in 43 complete genomes, representing 30 major phylogenetic lineages. Each COG consists of individual proteins or groups of orthologs from at least 3 lineages and thus corresponds to an ancient conserved domain. The COG categories can be assigned automatically to unclassified genes by blasting against a FASTA database of prokaryotic or eukaryotic sequences with already assigned groups. The COGs database can serve as a platform for functional annotation of newly sequenced genomes and for studies on genome evolution. To facilitate functional studies, the COGs have been classified into 17 broad functional categories, including a class for which only a general functional prediction, usually that of biochemical activity, was feasible and a class of uncharacterized COGs. Additionally, some of the COGs with known functions are organized to represent specific cellular systems and biochemical pathways. The database is accompanied by the COGNITOR program, which assigns new proteins, typically from newly sequenced genomes, to pre-existing COGs.


Netscape: AmiGO : GO:0009089 details
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://www.godatabase.org/cgi-bin/go.cgi?query=GO:0009089&view=details&search_constraint=terms&depth=0&session_id=22061030014092
WebMail Radio People Yellow Pages Download Calendar Channels
AmiGO
Get this GO term as RDF XML.
Get this data as a GO flat file.
Graph view.
Filter Associated Genes

lysine biosynthesis, via diaminopimelate

Accession:GO:0009089
Synonyms: None.
Definition: None.

Term Lineage

```
GO:0003673 : Gene_Ontology (33650)
  GO:0008150 : biological_process (24768)
    GO:0008151 : cell growth and/or maintenance (16459)
      GO:0008152 : metabolism (11711)
        GO:0006519 : amino acid and derivative metabolism (561)
          GO:0006520 : amino acid metabolism (465)
            GO:0008652 : amino acid biosynthesis (163)
              GO:0009067 : aspartate family amino acid biosynthesis (19)
                GO:0009085 : lysine biosynthesis (11)
                  GO:0009089 : lysine biosynthesis, via diaminopimelate (11)
            GO:0009066 : aspartate family amino acid metabolism (107)
              GO:0009067 : aspartate family amino acid biosynthesis (19)
                GO:0009085 : lysine biosynthesis (11)
                  GO:0009089 : lysine biosynthesis, via diaminopimelate (11)
              GO:0006553 : lysine metabolism (19)
                GO:0009085 : lysine biosynthesis (11)
                  GO:0009089 : lysine biosynthesis, via diaminopimelate (11)
        GO:0009058 : biosynthesis (2531)
          GO:0008652 : amino acid biosynthesis (163)
            GO:0009067 : aspartate family amino acid biosynthesis (19)
              GO:0009085 : lysine biosynthesis (11)
                GO:0009089 : lysine biosynthesis, via diaminopimelate (11)
```

External References
GenProtEC (1)
MetaCyc (1)
InterPro (3)

Associated Genes **Gene Filters: Filter by database:** All, FlyBase, SGD
Page 1
Filter by Evidence for Association: Curator Approved, All, Inferred from Mutant Phenotype

GO Term:

Figure 3.9.: This screenshot of the AmiGO browser illustrates how the GO categories can be explored. By clicking on one of the entries, further information can be obtained about the selected category.

3.4.7. Other classification schemes

Several other classification schemes like InterPro numbers or SwissProt keywords exist but since the Gene Ontologies provide a mapping to these categories, they are not discussed in more detail here. In addition to the classification schemes mentioned above, a number of (sometimes even manually curated) databases exist like LocusLink[16], GenBank[17], UniGene[18], IMAGE[19], and others which have implemented their own nomenclature with specific sets of identifiers and descriptors.

3.5. Integrating approaches

Several systems have been designed so far that focus on a more general approach for the integration of heterogeneous data into a common platform for systems biology. Although there are many products that advertise solutions for a complete integration of all kinds of data from functional genomics – also due to the immense commercial interest in this field of research – most of them have some major drawbacks and do not keep their promises.

[16] *http://www.ncbi.nlm.nih.gov/locuslink/*
[17] *http://www.ncbi.nlm.nih.gov/*
[18] *http://www.ncbi.nlm.nih.gov/UniGene/*
[19] *http://image.llnl.gov/*

3.5.1. BioMOBY

BioMOBY [WL02] is an ongoing Open Source research project which aims to generate an architecture for the discovery and distribution of decentralized biological data sources through web services. A central registry (MOBY Central) holds the input and output object types of all registered services, the URLs for these services, and their service types. Structured data (so called MOBY objects) are passed between client and server in a lightweight XML format using SOAP[20]. These objects may also contain an optional Cross-Reference Information Block (CRIB) for providing cross-references to other data objects. Additionally, BioMOBY features the concept of data retrieval workflows that can be constructed based on the given input/output types of different services. In comparison to other approaches, BioMOBY is focused on a minimalistic model for data discovery and transport instead of standardisation or representation.

3.5.2. MyGRID

Similar to BioMOBY, the MyGRID project [SRG03] aims to exploit Grid technology (for distributing large scale computations) and provides high level services for bioinformatics data and application integration. In addition to nearly identical ideas for data discovery and service execution methodologies, MyGRID is more focused on the inclusion of bench-scientist's tools such as workflows including automated notification and updates, personalised data repositories, and provenance management similar to lab books.

3.5.3. Discovery Net

The Discovery Net system [RKO$^+$03] is another middleware that allows service developers to integrate tools based on existing and emerging Grid standards such as web services. It was primarily designed to create reusable workflows, data flow processes, or pipelines that can be composed based on the integrated tools and deployed as new services. Therefore, a process is described using the XML-based Discovery Process Markup Language (DPML). The modular architecture of the Discovery Net system currently provides six service components:

- A *Component Service* manages the integration of different components and services into the system.
- The *Execution Service* distributes the execution of jobs.
- A *Data Access and Storage Service* was designed to aid common data access tasks.

[20] *http://www.w3.org/TR/soap12-part1/*

- The *Computational Service* integrates computational services directly into the core Discovery Net system.
- An *Info Grid Service* provides a standard query interface for heterogeneous databases.
- The *User Defined Service* describes new services that were added using standard interfaces provided by the *Computational Service*.

Furthermore, the system features a *Discovery Net API* for programmatic access to all services and *Discovery Net Clients* that provide users with graphical interfaces for constructing their knowledge discovery workflows.

3.5.4. SRS

The *Sequence Retrieval System* (*SRS*, [EA93]) developed by LION Bioscience AG is an integration system for data retrieval and sequence analysis applications. It is a web-based gateway to most of the important databases in the field of molecular biology (GenBank, SwissProt, PIR, etc.). SRS provides a unified interface for querying more than 150 databases that are grouped into specialized sections. It is a keyword-based system and thus limited to free text descriptions that are indexed for faster searching. Recent versions of the SRS system also support virtual databases that can be set up for easily querying the major releases and the incremental update versions of a database in a single step. Furthermore, XML databases like InterPro, the GO database, MEDLINE, and metabolic pathway databases are integrated into the system and user friendly views for these data can be generated as HTML pages. Among other features, SRS provides a quick search and user defined bookmarks.

3.5.5. SEMEDA

The Semantic Metadatabase *SEMEDA* [KSK02] was designed as a three-tiered application for “intelligent” semantic integration and querying of federated databases. The system features the following three main components: the *MARGBench* module provides SQL access to integrated databases by database federation, while the ontology based semantic metadatabase (*SEMEDA*) stores information that can be accessed by an ontology based query interface (*SEMEDA-query*). Therefore, ontologically structured information from different data sources can be integrated based on a set of common database attributes. Available ontologies and knowledge sources can be imported automatically, but it is also possible to manually curate the database. Furthermore, the system can derive relationships by exploring the incorporated data.

3.5.6. ISYS

ISYS [SFT+01], the integrated system from NCGR[21] has been developed as a dynamic and flexible platform for the integration of bioinformatics software tools and databases. ISYS offers a component-based architecture that enables scientists to "plug and play" with tools of interest. These tools may be developed separately and evolve independently and they may include a group's own databases and analytical programs as well as those available publicly or for a fee.

In addition, ISYS allows web-based resources to be integrated with programs running on the scientist's desktop. The ISYS DynamicDiscovery technology creates an exploratory environment in which scientists can navigate freely among registered components. DynamicDiscovery helps to guide the user by suggesting appropriate registered components to process selected data objects. Furthermore, ISYS supports visual synchronization among components which helps each one to complement the others.

ISYS is written in Java for platform independence and is supported on Windows and Solaris platforms. It is also available without a Java Virtual Machine for Linux and other types of UNIX. ISYS is highly customizable for the needs of individual scientists and organizations. The current version of the system contains eight components summarized in table 3.6.

Component	Description
Sequence Viewer	Graphical viewer for sequence annotation.
Similarity Search Launcher	Graphical interface to configure and compute BLAST batch analysis.
Similarity Search Browser	Customizable graphical browser for the results of a similarity search.
Table Viewer	Generalized component for displaying data in tabular form.
Entrez server proxy.	A proxy to the NCBI's Entrez data retrieval system.
ORF-to-gene mapper	ISYS Service Provider that maps between ORF and gene names for yeast.
maxdView	Full-featured and highly-customizable gene expression viewer.
BDGP GO Browser	Graphical browser for the database of the Gene Ontology Consortium

Table 3.6.: Available components for the ISYS system. All modules can be used interactively in order to explore genome or transcriptome data on a basic level.

Although the ISYS system features a generic approach for the integration of individual components, it does not include full featured systems (e.g. a complete genome annotation system or a platform for efficient storage and analysis of microarray data). ISYS is more a collection of small "plug-ins" than a comprehensive architecture that appears as a single highly customizable platform. It also lacks a project management unit, an API for the extension by other programmers, and a consistent graphical user interface.

[21] *http://www.ncgr.org/isys*

3.5.7. DAVID

DAVID [DJSH+03] is a Database for Annotation, Visualization, and Integrated Discovery that provides some data-mining tools which systematically combine functionally descriptive data with intuitive graphical displays. The system features an annotation tool, GoCharts, KeggCharts, and domain charts for visualizing weekly updated lists of genes for several genomes, e.g. human, mouse, rat, or fly. Expression data from Affymetrix experiments can be mapped onto the pathways or functional categories and hyperlinks to related data sources (e.g. Unigene, LocusLink, RefSeq, Gene symbol) provide additional information. DAVID collects information from different sources and imports these data into its own database. The system is neither a complete application for genome annotation nor does it provide an API for programmers.

3.5.8. GeneData product series

The GeneData[22] product series is outlined here as one example for commercial bioinformatics applications. GeneData advertizes a knowledge management system that allows integrating information from various technologies and provides comprehensive insight into organisms, disease mechanisms, and drug actions. The GeneData products listed in table 3.7 cover the most important aspects in functional genomics data analysis.

Product	**Description**
Phylosopher	Genome analysis and gene function prediction.
Expressionist	Analysis of gene expression data.
Impressionist	Analysis of protein expression data.
Metabolist	Analysis of metabolic data.
Screener	High-throughput screening data analysis and compound profiling.

Table 3.7.: Available products of GeneData. The company offers separate packages for genome annotation, transcriptome, proteome, and metabolome data analysis.

Since all programs are commercial products, the source code is not available free of charge to academics for evaluation and/or further extension. All GeneData systems also lack a well-designed infrastructure or common interface for the integration of other third party components. Furthermore, it is unclear whether the products can only share some of their data sources or if they can also be integrated in a single graphical user frontend that allows direct interaction between different components.

[22] *http://www.genedata.com/*

3.6. Conclusions

As described in the previous sections, several special purpose systems exist for the analysis of genome and transcriptome data and for the visualization of metabolic pathways. In general, most of these systems lack a consistent internal data representation and well-defined extensible APIs for accessing and manipulating the data. Furthermore, the ommission of a common and standardized interface complicates the integration of such systems into a common framework for comprehensive data exploration. Those systems that already integrate different types of data are often restricted to a specific scope of applications. Most of them cannot be used as a full featured data analysis pipeline since higher level input formats (e.g. XML files) are required.

CHAPTER 4

Specification analysis

The existing systems described so far all have their limitations and drawbacks. While the focus of all research in the field of molecular genetics is moving from single gene analysis towards large scale whole genome explorations on the transcriptome, proteome, and metabolome level, new demands for bioinformatics software arise. In this chapter some of the high expectations that are imposed on such new systems will be identified and translated into more concrete specifications from the bioinformatician's perspective.

4.1. From functional genomics towards systems biology

All novel technological developments in the field of functional genomics are currently directed towards analyzing complex interactions by a bottom-up approach. Employed high-throughput techniques such as transcriptomics or proteomics produce a flood of experimental data and many other details about separate biological components that have to be stored systematically as a basis for the challenging task of gaining insight into the complex function of an organism as a whole. Thus, genome research is trying to reduce life into more simple components for detailed analysis (*high-throughput reductionism*) [Kat03].

However, the discovery and exact chemical definition of all components in a living system does not mean that we will understand how it works. Therefore, systems biology generates knowledge from the components of a complex biological system by incorporating the following types of analyses as defined by [IGH01]:

- System Structure Identification (pathways & networks)
- System Behavior Analyses (dynamics of the system)
- System Control (change variables to control other variables, apply control theory on the system)
- System Design (design biological systems based on components).

Knowledge about single components concerning their structure, regulation, control, adaptation, robustness, redundancy, or evolution has to be combined in order to understand the global principles, interactions, and the dynamics of a living system. Systems biology deals with enzymes that form pathways, their control, and interaction with other macromolecules, and with control mechanisms that keep these structures functioning from generation to generation. Finally, this also includes the creation of sophisticated (mathematical) models and the (a priori) simulation of *in silico* experiments that can then be verified by *in vitro* or *in vivo* experiments as displayed in figure 4.1.

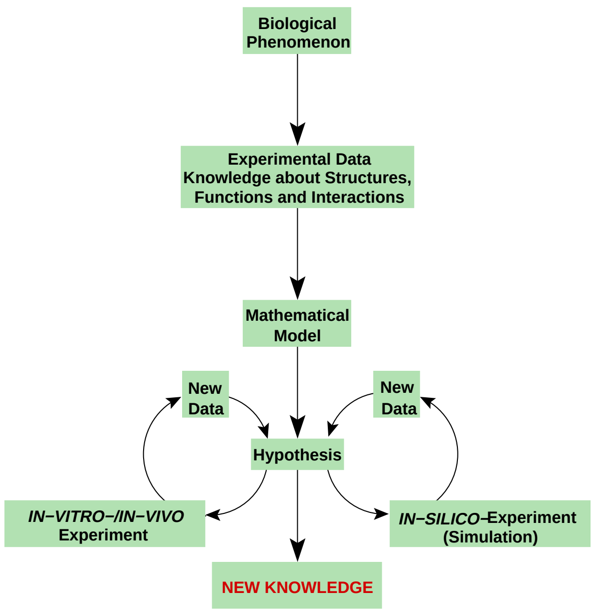

Biological Phenomenon
Experimental Data Knowledge about Structures, Functions and Interactions
Mathematical Model
New Data
New Data
Hypothesis
IN–VITRO–/IN–VIVO Experiment
IN–SILICO–Experiment (Simulation)
NEW KNOWLEDGE

Figure 4.1.: The dynamic process of new knowledge generation in systems biology. Based on mathematical models that can be derived from experimental data, hypotheses are generated and verified, discarded, or enhanced by *in silico*, *in vitro*, and *in vivo* experiments (adapted from [Bun02]).

From a more practical point of view this means that a platform for systems biology should support arbitrary complex queries that are not limited to a certain scope but allow finding distinctive features in complex data structures.

4.2. Data types and sources

As already outlined in the introduction (see chapter 2), today researchers are confronted with a variety of methods for the analysis of genes and genomes. The application of high-throughput techniques such as microarray experiments and mass spectrometry produces a wealth of information that has to be evaluated and interpreted. As shown in figure 4.2, different analyses can be based on other preliminary results but on the other hand, their outcomings have to be compared and related to the original data.

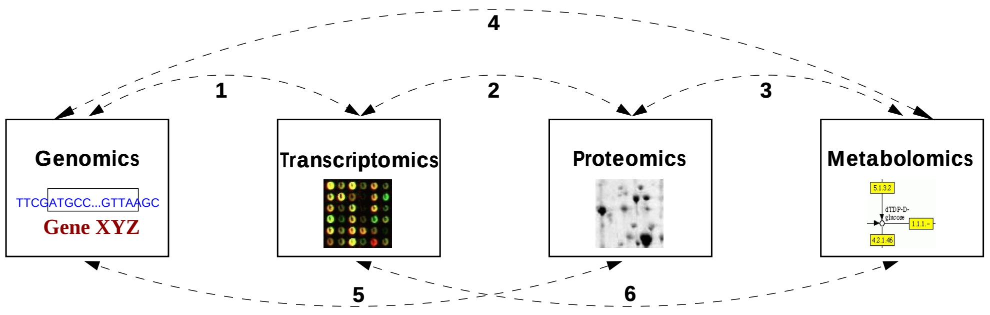


Figure 4.2.: Genomics, transcriptomics, proteomics, and metabolomics provide high throughput methods for the analysis of uncharacterized genes. The arrows indicate the mutual influence of results from different experiments: for example, the arrows 1, 4, and 5 indicate that the obtained experimental results reflect the set of genes that were predicted and annotated for the genome under investigation. Inconsistent results (e.g. additional spots on a 2D gel, missing enzymes) may indicate errors in the original genome annotation and thus require corrections or updates. On the other hand, comparing different experimental results as indicated by the arrows 2, 3, and 6 can support or invalidate previously stated hypotheses.

Furthermore, storing the experimental setups and laboratory protocols is also essential to guarantee the reproducibility and reliability of the obtained results from often complex workflows. The data acquired thereby ranges from unstructured flat files or ASCII tables to XML documents and high resolution images. For all of these data, well structured persistent storage is needed that also keeps track of cross-references to other related data sources. It would also be quite advantageous to have a central component that contains a unique instance of each analyzed piece of sequence or gene that can be referenced and linked to different experiments where it is involved. Such a system should also be able to store already known facts, observations obtained from experiments, annotations, and all other available information about it. Since many experiments contain sensitive (e.g. unpublished proprietary) data, access has to be restricted in order to prevent unauthorized access or even loss of results.

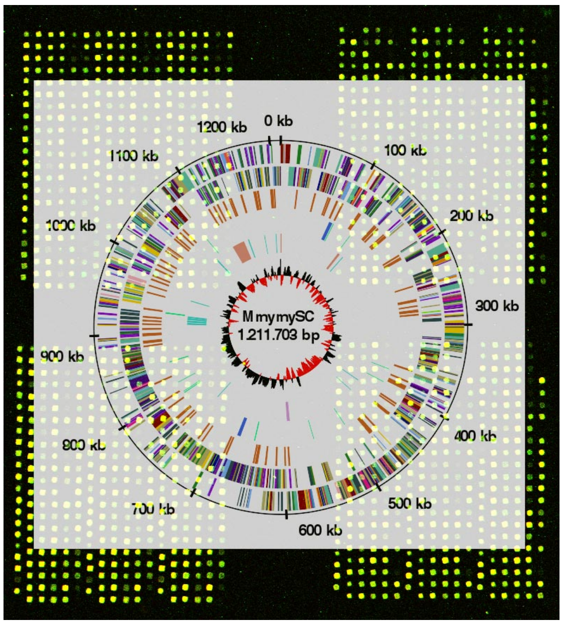


Figure 4.3.: Data integration is hard: Related information has to be combined but at the same time different points of view onto the same data are required to reveal certain aspects of interest. In this example, the interest of the user might be focused on the combined analysis of expression ratios and functionally related genes.

Besides the task of integrating such heterogeneous data types and sources, it is also important to be aware of different points of view that users have in their mind when they analyze the data. Depending on the type and on the stage of an experiment, the focus may change and require other visualizations. For example, a microarray experiment might first involve some visual inspection of spots and their replicates on a slide to ensure a desired level of quality, but for a further evaluation it would be more interesting to see significantly up or down regulated genes in a circular plot of the whole genome with a color code for the functional categories of the affected genes (see figure 4.3). Both views are focused on different aspects or types of experimental results and therefore they require their own analysis methods and sometimes customizable specialized visualizations.

4.3. Users and developers

Other aspects that have to be considered for any larger software project are the needs of individual users of the system. Obviously, for both users and developers, it is important that any software system is easy to install and to maintain. But there are quite a few other important aspects that have to be considered carefully.

From a **user's** perspective, the usability of any system plays the key role. Optimized and highly customizable applications should not only provide "nice" graphics and inspire the user with various graphical user frontends. Above all, it is still the homogeneous and consistent usage of widely accepted graphical elements such as menus and buttons that should allow an intuitive exploration of the application's functionality. These demands can be fulfilled by simply using a common look and feel for all graphical user interfaces (GUI) that are related to each other. Applications that provide access to large data sets should also support searching for special features as well as the identification of common properties among related items. In addition to this, further insight into complex data structures can be achieved by presenting standardized (meta-) views onto heterogeneous data (e.g. by mapping genome and expression data onto metabolic pathways).

From a **developer's** point of view, the key feature is often the extensibility of an existing system. A surprising lesson learned from the analysis of the existing systems is the lack of consistent internal data representation. However, an internal data representation using a well defined data model is the prerequisite needed to provide an application programmer's interface (API) for any larger software system. Ideally, such an API allows the implementation of human readable code that can be derived easily from more abstract descriptions of algorithms, e.g. written in pseudo code as illustrated in section 1.1. On the other hand, a modular system should be open for further modifications and improvements thus allowing other researchers to integrate their own ideas and extensions without rewriting large parts of the software. The successful extension of any open source product can be supported by providing a well defined and documented API but it is also inevitable to ensure the stability of (well tested) software by a central release management. Further, it is important to notice that the availability of operating system and programming language independent interfaces is another desirable feature which is often underestimated.

For the integration of heterogeneous data from functional genomics into a platform for systems biology, such a system should not only support single directed pipelines but feedback-loops that can help to create enriched genome annotations. The central design concept that can be applied for complex systems is the use of exchangeable specialized components. Nevertheless, such a module should be able to be executed as a stand-alone application as well as a plug-in for a completely integrated solution (e.g. execute stand-alone genome annotation system or combine it with a system for microarray analysis).

Last but not least, a sophisticated system should also be compliant to standard data formats and support common input and output data formats to allow data exchange and ensure the compatibility with other tools.

4.4. Data management

In many areas of research modern software applications have to deal with huge amounts of information that are frequently collected from high-throughput experiments. This data has to be stored in well structured repositories (e.g. database management systems) that provide efficient automated access for all further downstream analysis of the obtained results. Further, new knowledge generated by higher level data evaluation has to be stored consistently and needs to be cross-linked to the original information. Since all research projects often produce highly confidential results, it is also important that essential parts of the data can be protected from unauthorized access. Obviously, increasing numbers of projects require extensive administration of project related data sources, users, and their individual privileges for accessing the information. A central component for managing projects and users is necessary to address these issues and to provide a systematic approach that helps to keep an overview across all projects.

In times of rapidly increasing demands for high performance computing methods and increasing storage requirements, a majority of software applications has to store and access external data sources. For example, a number of software systems is currently developed at the Center for Genome Research, Bielefeld University, helping to organize the flood of genomic and post-genomic data. Obviously, all information that is acquired from wet lab experiments or from manual analysis of the obtained results requires persistent storage and reliable backup capabilities in order to ensure data-integrity.

While several relational or object-oriented database management systems like MySQL,[1] PostgreSQL,[2] DB2,[3] or Oracle[4] already provide well-suited and stable solutions for storing and maintaining large datasets, there are still some additional issues that have to be addressed for real world applications. Figure 4.4 illustrates a typical access procedure that is implemented in many software systems.

[1]http://www.mysql.com/
[2]http://www.postgresql.org/
[3]http://www.ibm.com/software/data/db2/
[4]http://www.oracle.com/

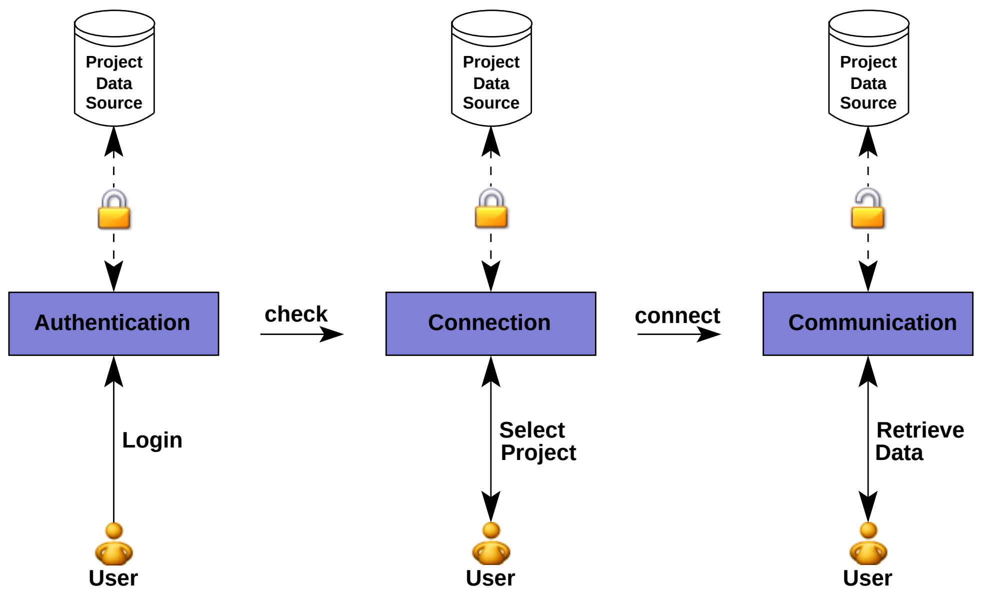


Figure 4.4.: A typical access procedure requires an initial authentication of the user. After logging in, a dataset can be selected and the connection to the corresponding data source is established. Finally, the requested information is presented to the user.

Basically, the initial step for accessing data requires a connection to a database management system. Connections can be established by command-line interfaces or graphical user frontends via an API. Although most systems provide different comfortable ways for accessing their data, details of the access protocol and maybe even the type of the data source (e.g. flat file or RDBMS) should be hidden from the user by a frontend application, e.g. a web interface. In general, it is also important to provide transparent and consistent access to all data within the same scope (e.g. all information that has been acquired in a transcriptome project). This also includes the use of standard access routines that should be available independently of the chosen storage method. This data and all related information can be collected and organized in projects. Once a user has established the connection to a data source, the level of access is often defined by special permissions or privileges. While some database systems allow very fine grained access control, the administration of such permissions is usually a laborious task for the maintainers of the data repository. Often, additional work is required in cases where it is desirable to restrict access to specific users for projects containing sensitive data. In such cases, different roles can be identified that manifest the level of access by assigning appropriate privileges. On the other hand, an individual user can thus act in various roles for different projects (e.g. with read only access as a guest user or read/write permissions as a developer) as illustrated in figure 4.5.

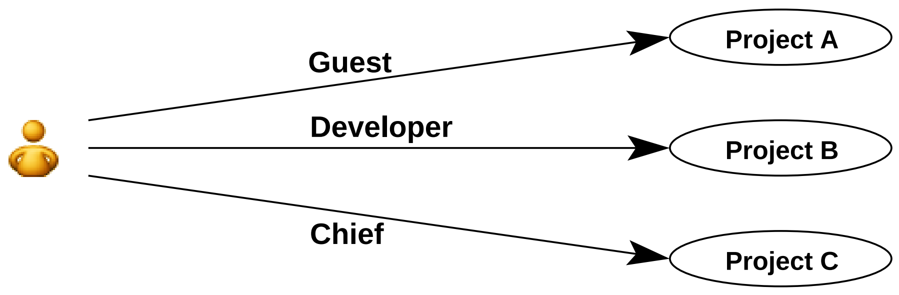


Figure 4.5.: For different projects a user can act in various roles. Guest access may only include very few privileges while acting as a software developer would require almost complete control over all data.

Obviously, it is desirable to keep the administration overhead as small as possible; the maintainers of (large) database management systems should be supported with an easy-to-use interface that helps to keep an overview of all users and the projects they are involved in. Such a system could also provide some kind of user management interface that allows maintainers of a project to grant dedicated access to (parts of) the information to specific users without involving a database administrator.

In addition to the data itself, almost every modern application stores a number of individual settings per user. In this case, a project management system should just as well be utilized for storing these settings separately for each project, independently of the frontend that is employed by the user (e.g. web frontend or GUI). Extending the scope of organizing data in separate projects, applications often refer to related data from different sources. Therefore, it is essential to provide a simple means for accessing and linking these information. Again, these references should be hidden from the user but they should allow asking questions across different data sources. Thus the application has to know where to find the requested information and how to access it.

CHAPTER 5

Choice of core technologies

Since most requirements described in the previous chapter are not unique for the development of applications in the field of functional genomics or bioinformatics, there are already a number of existing solutions for such problems. In general, for the developer of any software system it is always a good advice to use widespread, stable, and ready-to-use production systems. In the following sections some existing general approaches and concepts will be presented that can be employed to fulfil the requirements described in the previous chapter.

5.1. Existing systems revisited

The increasing number of applications of high-throughput methods for the simultaneous analysis of hundreds or thousands of genes in a single experiment leads to the demand for solutions that allow the flexible integration of heterogeneous data types and data sources into an extensible platform for systems biology. Such a system should not only be able to cope with high dimensional data but also provide different (meta) views on the data that therefore has to be cross-linked. Although there are many software packages available that can be used for the analysis of data from one of the research areas described in chapter 2 (e.g. MAGPIE by [GS96], ERGO by [OLW$^+$03], Artemis by [RPC$^+$00] and PEDANT by [FAH$^+$01] for genome annotation or J-Express by [DJ01] and BASE by [STVC$^+$02] for microarray analysis), there is no open source system known to the author that features the complete integration of different data sources **and** their corresponding frontend applications. While recently developed systems such as BioMOBY [WL02], Discovery Net [RKO$^+$03], or MyGrid [SRG03] focus on providing decentralized web services, other approaches like SEMEDA [KSK02] try to attack the problem of integrating heterogeneous data sources by an ontology based semantic metadatabase. On the other hand, the component-based approach of the ISYS [SFT$^+$01] software tries to solve the problem of heterogeneity by implementing specialized client side user interfaces that can communicate with each other. Nevertheless, all systems developed so far lack capabilities for initially accessing objects located on a remote server, then using their already implemented functionality, and finally also modifying them directly. Therefore, we are envisioning a platform for systems biology that not only supports the integration and visualization of decentralized heterogeneous data but also allows the direct manipulation of objects using a sophisticated access control policy.

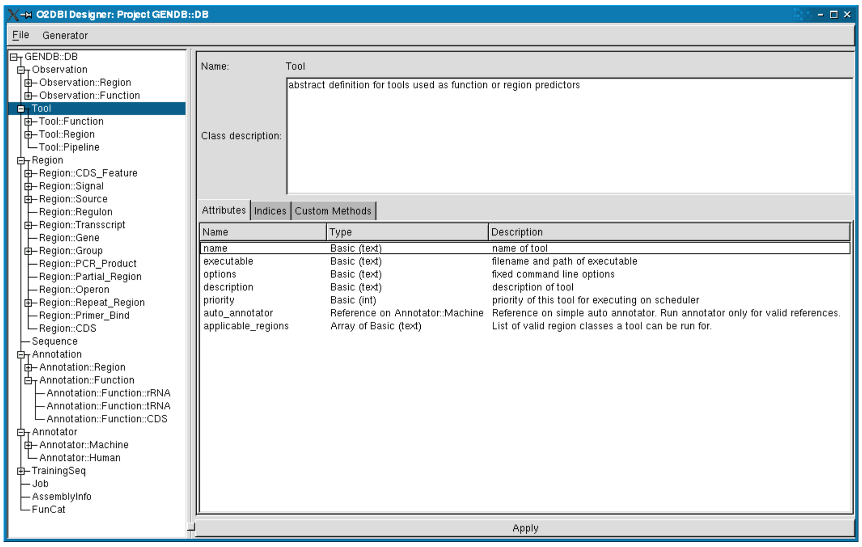
5.2. Relational object-oriented modeling

One of the key features of many applied (bioinformatics) systems such as laboratory inventory management systems (LIMS) is the ability to provide persistent storage mechanisms. Laboratory protocols, data acquired from complex experiments, the results of automatic and manual analysis, and increasing loads of other data sources (e.g. experimental setups and parameters) have to be stored efficiently. Only well structured data repositories can guarantee easy and efficient access for a long time. As a consequence, these considerations may finally include the necessity to provide (public) access to the data.

Efficient persistent storage and comfortable access to large datasets can be achieved by employing relational or object-oriented database management systems like MySQL, DB2, PostgreSQL, Oracle, and others. The design of well structured data models as a basis for the implementation of documented and easy-to-use application programmer's interfaces (APIs) can be supported by using UML (Unified Modeling Language[1]) for the definition of entities and their relationships.

Therefore, the O2DBI-II system [Lin02] (see Figure 5.2) was developed that allows the mapping of Perl objects to relational tables. All classes and their attributes can be defined using the comfortable O2DBI-II Designer (see figure 5.1). The resulting data schema is then stored in an XML [BPSMM00] file. Based on the data structures defined with the Designer, a library of Perl classes with Perl and C++ client-server bindings can be generated automatically. It is also possible to convert object descriptions from the UML (XMI) format into the O2DBI-II XML format.

[1]http://www.omg.org/uml/



- GENDB::DB
  - Observation
    - Observation::Region
    - Observation::Function
  - Tool
    - Tool::Function
    - Tool::Region
    - Tool::Pipeline
  - Region
    - Region::CDS_Feature
    - Region::Signal
    - Region::Source
    - Region::Regulon
    - Region::Transscript
    - Region::Gene
    - Region::Group
    - Region::PCR_Product
    - Region::Partial_Region
    - Region::Operon
    - Region::Repeat_Region
    - Region::Primer_Bind
    - Region::CDS
  - Sequence
  - Annotation
    - Annotation::Region
    - Annotation::Function
      - Annotation::Function::rRNA
      - Annotation::Function::tRNA
      - Annotation::Function::CDS
  - Annotator
    - Annotator::Machine
    - Annotator::Human
  - TrainingSeq
  - Job
  - AssemblyInfo
  - FunCat

Name: Tool

Class description: abstract definition for tools used as function or region predictors

Name	Type	Description
name	Basic (text)	name of tool
executable	Basic (text)	filename and path of executable
options	Basic (text)	fixed command line options
description	Basic (text)	description of tool
priority	Basic (int)	priority of this tool for executing on scheduler
auto_annotator	Reference on Annotator::Machine	Reference on simple auto annotator. Run annotator only for valid references.
applicable_regions	Array of Basic (text)	List of valid region classes a tool can be run for.

Figure 5.1.: The O2DBI-II Designer can be used to develop the data model. Classes and their attributes can be defined via the graphical user interface. It is also possible to add database indices and extensible comments for each class.

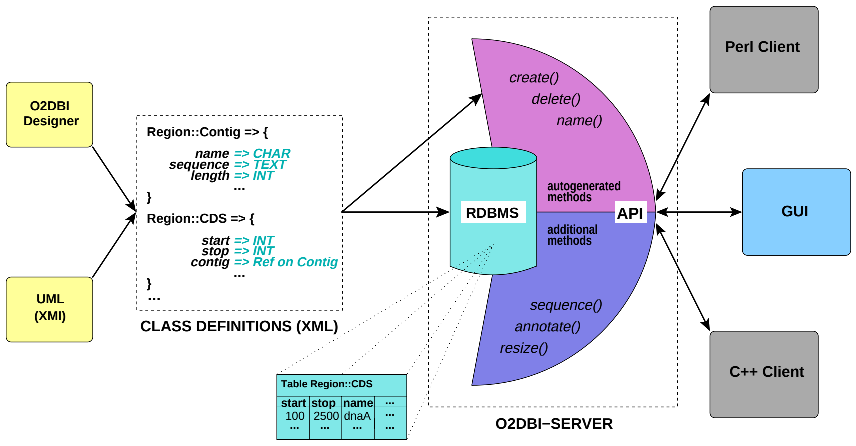


Figure 5.2.: Class declarations are created with the O2DBI-II designer or can be converted from UML descriptions. Based on the class hierarchy described in XML, O2DBI-II maps Perl objects to relational tables, generating both SQL tables and Perl server modules. O2DBI-II also generates Perl and C++ client code that can be used to implement remote access mechanisms.

All objects are stored in a relational database (e.g. MySQL or PostgreSQL) and Perl as well as C++ source code is generated that implements standard methods (create, delete, init, get/set etc.) to access the objects. These automatically generated object methods are stored in a Perl O2DBI-II server module. Extension of the object functionality is possible in separate Perl modules. All accesses to the data of a specific database via O2DBI-II methods are managed by a special O2DBI-II master module. As stated above, the auto-generated classes and the manually added methods form the API.

5.3. Interaction and communication

With the open-source O2DBI-II toolkit, data can be stored efficiently and at the same time the system provides an API that allows easy access to all information and individual extensions of the auto-generated classes. Consequently, using an object-oriented approach for the design of all data structures (e.g. by creating class hierarchies and employing inheritance) allows a rapid development and enhances the modularity and usability of all components. But nevertheless, additional mechanisms are needed that allow the interaction and communication between different components. For example, the user might request experimental transcriptomics data for a selected gene while she/he is assigning functional classifications in a genome annotation system. In this case, immediate interaction, response, and maybe also some kind of visualization is needed to provide the user with the requested information. Since user interfaces that allow the integration of different data sources are highly dynamic, customizable views and dynamic visualizations are needed that exactly represent the desired information. It is also important that the different integrated graphical user frontends provide a common look and feel where the user can easily find what she/he is looking for. For standard applications, the various existing GUI toolkits (Tk, Gtk, Swing, Qt) support different mechanisms (either callbacks or signals and slots) that allow the interaction of graphical elements (widgets).

For the exchange of different types of data, the widespread application of XML has shown its usefulness as a well suited format since all parsing of the data structure can be done automatically. For remote data exchange, commonly used protocols such as XML-RPC or SOAP can be used that are platform independent and thus available for almost every operating system.

5.4. CORBA

CORBA (Common Object Request Broker Architecture[2]) is a complete, complex communication structure built to become an industry standard, capable of networked, cross-platform,

[2]http://www.omg.org

reliable communication between any applications that subscribe to the standard. Because of its functionality, completeness, availability, and robust C++ bindings, CORBA has ruled out many other approaches in this field but at the same time it has several limitations that confirmed the decision to implement other solutions:

- Especially for simple interactions, CORBA involves too much overhead for the requirements of direct communication between different graphical user interfaces.
- The dynamic character of a plug-in architecture is not supported sufficiently by the static nature of CORBA.
- Compilation of a CORBA-enabled code base can be very time-consuming and resource demanding.
- It would be difficult to convince (sometimes unexperienced) developers of small applications that they have to read and understand thousand-page manuals before they can integrate their own components and enable interprocess communication. Therefore, the simplicity provided by the (partially automatically generated) consistent and easy-to-use O2DBI-II-APIs is superior and supports the cooperative development of open-source software.

The discussion of all aspects mentioned above resulted in the conclusion that this communications technology would incorporate too much work and administrative overhead compared to the gained benefits. Thus, the (still possible) implementation of a CORBA layer for the interprocess communication was rejected for the current development of the platform.

5.5. Comparison of existing approaches

Although most of the systems presented so far were developed for completely different scopes and tasks, an evaluation of useful components for a further system design requires a comparison of the existing approaches. Because of their diversity, a comparison is performed for the following criteria:

- *Number of features*:
  This aspect tries to rate the completeness of an application concerning its usability for standard tasks in functional genomics such as annotating a genome or analyzing microarray data.
- *Programmer friendliness*:
  The programmer friendliness of an application can be evaluated by looking at the available interfaces for programmer's (APIs) and their documentation. In contrast to those

aspects that are important for the frontend users, the availability of good APIs allows an easy adaption of a system and the implementation of special individual solutions.

- *Extensibility*:
  This criteria is used for evaluating the extensibility and modularity of a system. The rating depends on the availability of concepts for adding new components to an existing system.

- *Access policy*:
  Since most researchers are concerned about sharing their private unpublished data (certainly for good reasons), a system should support well defined access control mechanisms that implement a sophisticated policy in order to prevent unauthorized access. Thus the availability and quality of such features is also evaluated for all systems.

Each approach was then analyzed based on its description or publication (see also section 3.5). The results evaluated for each criteria are listed in table 5.1 where each of the existing systems was scored (++ denotes a very positive aspect, + a positive aspect, o a neutral aspect, - a negative, and - - a very negative aspect).

System	**features**	**progr. friendly**	**extensible**	**access policy**
BioMoby	--	+	++	o
MyGrid	--	+	++	o
Discovery Net	o	o	o	-
SEMEDA	o	--	--	o
ISYS	+	o	+	--
O2DBI-II	--	++	++	o
CORBA	--	+	+	o
David	+	-	-	-

Table 5.1.: Comparison of existing approaches and evaluation of their usability as building blocks for a platform for systems biology.

For example, BioMoby, MyGrid, and O2DBI-II were positively evaluated for their modularity and extensibility. On the other hand, systems like BioMoby, MyGrid, O2DBI-II, and CORBA were rated negatively for the obvious lack of features concerning their usability for standard tasks in functional genomics. As illustrated in table 5.1, none of the existing systems obtained a top ranking for all of the evaluated criteria, but nevertheless there are many useful approaches and good ideas that could be adapted from one or another system. Especially some concepts of the ISYS system could be adapted for the development of a platform for systems biology. From a programmer's perspective, the O2DBI-II system seems to be the best choice for implementing the core functionality since it is easier to learn and maintain than CORBA. Summarizing the evaluation of this table, it can be stated that all of these systems lack some full featured functionality for standard tasks in functional genomics since they focus on more abstract data integration. Furthermore, these systems provide only a limited access control if any so that unpublished data cannot be treated in a protected and confidential way.

CHAPTER 6

System design

Based on the specifications described in chapter 4 and with respect to the already existing solutions, a general concept has been designed for the implementation of a **B**ioinformatics **R**esource for the **I**ntegration of heterogeneous **D**ata from **G**enomic **E**xplorations (**BRIDGE**). The system has been developed as a common and extensible framework that is flexible enough and well suited to serve as a platform for systems biology.

6.1. Specialized components for separate scopes

Concerning the variability of the different data sources, it is clear that the design of exactly tailored data models for separate scopes is the most important prerequisite as a basis for all further development.

If no such scopes are clearly defined and separated from each other, the different data types and sources get mixed up. References between corresponding data sets or experiments get confused and end up in chaotic collections of unusable descriptions. In some cases, important information may finally disappear uncontrollably and vanish in a "deep black hole".

Although separating data from different types of experiments is not that difficult during and right after their creation (e.g. transcriptomics and proteomics experiments are hardly ever performed by the same facility), it is more important than ever to uniquely identify them and describe a convenient level of detail when different results are combined afterwards in order to derive new hypotheses. Since all areas of research in the field of functional genomics require at least some profound knowledge about the experimental design and sophisticated methods for the analysis of obtained data, the BRIDGE system is based on a concept that features the implementation of specialized components for separate scopes as displayed in figure 6.1.

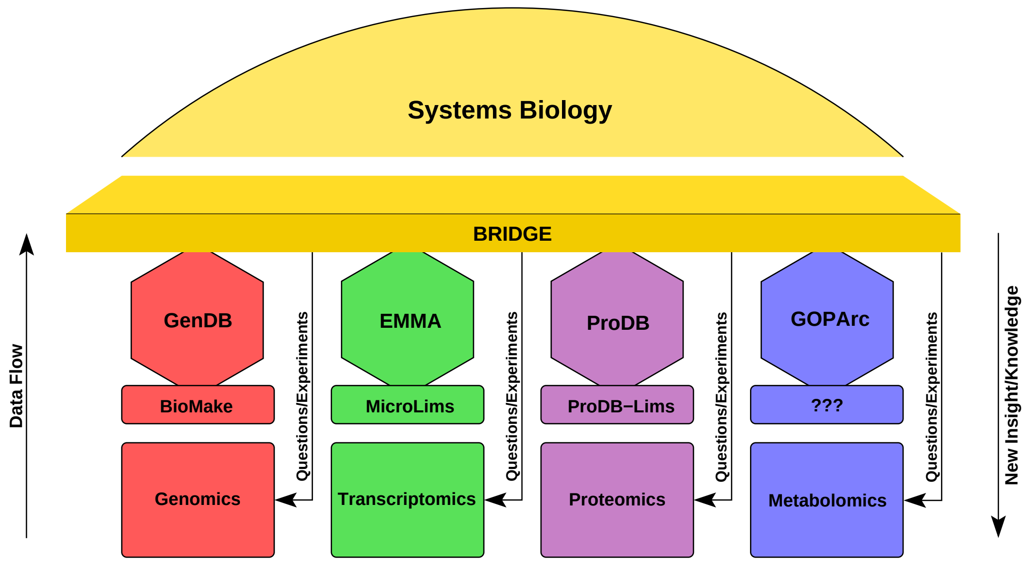


Figure 6.1.: A separate scope can be defined for the fields of genomics, transcriptomics, proteomics, and for the analysis of metabolic pathways. Special LIMS components can be used for storing the details about experiments. All modules are integrated into a common platform for systems biology via the BRIDGE layer. New knowledge and insights into more complex relationships can be derived by querying the different components and by combining the results obtained.

Since this work is focused on the integration of genome and transcriptome data in the context of metabolic pathways, only the required modules for these scopes will be described in detail in the next chapter:

- Genomics module: a central repository that stores all kinds of sequence data and serves as a major annotation facility. Enriched annotations can be created by linking different experimental results in this component and it should be possible to show a summary of all available information about a single gene as a kind of a gene report. Therefore this module should be able to collect recently available information from all related sources (e.g. from transcriptomics or proteomics experiments).

- Transcriptomics module: this module stores the complete experimental setup and parameters (as a LIMS) but also provide standard algorithms for normalization, filters, and data analysis (e.g. t-test, SOM, clustering). Additional user friendly visualizations should facilitate an easy exploration of the data obtained.

- Pathway module: such a module stores an internal representation of metabolic pathways and it should provide adequate visualizations that allow mapping of (multi-dimensional) data onto the pathway maps. For further detailed data analysis, this component should support the creation and visualization of individual implementations for specific metabolic routes. Data acquired from future metabolite analyses could also be stored in a separate LIMS component.

We are currently also developing other modules that might be useful and could be integrated into the BRIDGE system via the same mechanisms:

- Proteomics module: this component will store experiment conditions, setups, and parameters of proteomics experiments similar to the transcriptomics module. Further extensions are required for the analysis of 2D gel images and their corresponding peaklists, and for the large scale high-throughput identification of expressed proteins.

- Genome comparison module: such a module could perform genome comparison on different levels (e.g. sequences, genes, protein families, regulatory regions, pathways) and provide user friendly views on the computed data.

- Gene regulation module: a special component for the analysis of co-regulated genes and regulatory networks could be used to identify gene clusters and operons.

Another important additional requirement that has to be met by all components is their ability to be run as stand-alone applications. This allows an independent development of separate components and supports an easy replacement of individual modules.

6.2. Three-tier components

All of the specialized components described in the previous section have been designed following the classical standard three-tier architecture approach that is widely used when an effective distributed client/server design is needed. While hiding the complexity of distributed processing from the user, this architecture provides (when compared to the two-tier) increased performance, flexibility, maintainability, reusability, and scalability. For detailed information on three-tier architectures see [Sch95] and [Eck95].

This means that three-tier components have a "*client-server architecture in which the user interface, functional process logic ('business rules') and data storage and access are developed and maintained as independent modules*".[1] "*A three tier distributed client/server architecture includes a user system interface top tier where user services (such as session, text input, dialog, and display management) reside*".[2]

The middle tier provides process management services (such as process development, process enactment, process monitoring, and process resourcing) that are shared by multiple applications.

The bottom tier provides database management functionality and is dedicated to data and file services that can be optimized without using any proprietary database management system languages. The data management component ensures that the data is consistent throughout the distributed environment by using features such as data locking and replication. It should be noted that connectivity between tiers can be altered dynamically depending upon the user's request for data and services.

Apart from the usual advantages of modular software with well defined interfaces, the three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently as requirements or technology evolves. For example, upgrading the storage backend or changing the database management system (e.g. from MySQL to PostgreSQL) would only affect the O2DBI-II backend code. Typically, the user interface runs on a desktop PC or workstation and uses a standard graphical user interface; functional process logic may consist of one or more separate modules running on a workstation or application server, and an RDBMS on a database server or mainframe contains the data storage logic.

The classical three-tier architecture is illustrated in figure 6.2 including the design process as it is currently realized with the O2DBI-II system. After designing a complete module using UML, the O2DBI-II Designer can be used for optimizing the storage backend data model. The O2DBI-II Code Generator finally generates the object-relational mapping (standard methods for routine access) and a data handler specific for the employed storage backend. The main functional process logic is implemented in the class extensions. Different user interfaces (e.g. Gtk GUI, web frontend) on top of the business classes are provided for modifying and visualizing the stored data.

6.3. Integration

While the specialized components described in the previous section could be used as standalone applications in their specific scopes, the integration into a bioinformatics resource of heterogeneous data also needs a systematic approach that allows connecting two or more

[1]http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?three-tier
[2]http://www.sei.cmu.edu/str/descriptions/threetier.html

components for data exchange. Especially in this context, data exchange is not just meant to be accessing external data and displaying it. Instead, a comprehensive data integration infrastructure should include direct access and communication with external data objects that are referenced by other internal data structures.

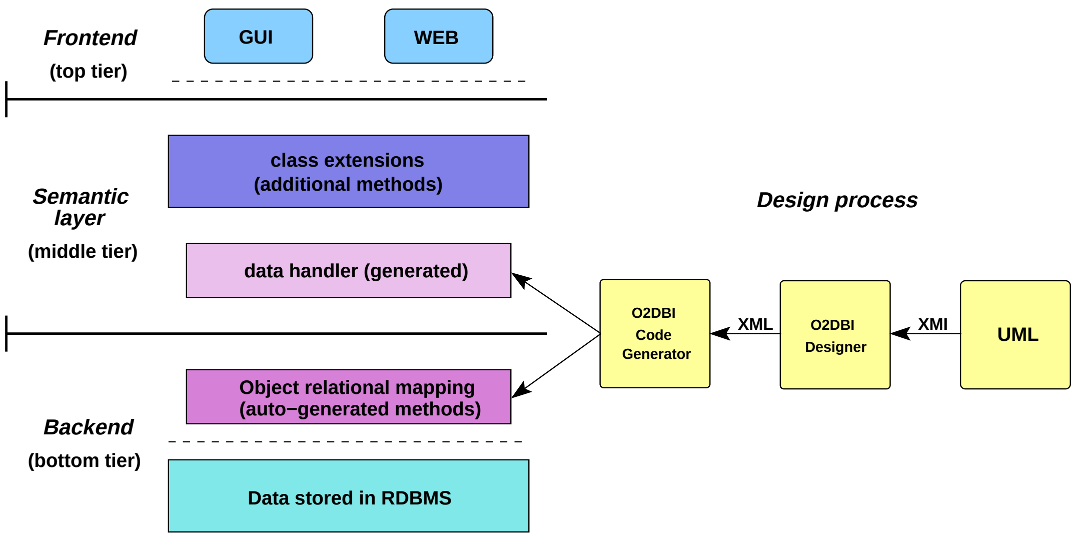


Figure 6.2.: Each component of the BRIDGE system has been designed as a three-tier module: a backend is responsible for storing all data, while the main functionality is implemented in the business classes of the middle tier. Different user interfaces are then based on the functional process logic and focus on data visualization. The design process is based on comprehensive UML modelling, routine access to the storage backend and parts of the semantic layer are automatically generated by the O2DBI-II system. Dashed lines represent potentially different hardware units (e.g. workstations or servers) that were used for the physical implementation of a module.

To explain this behavior we consider the following example:

Let:

- *objA* be an O2DBI-II object of class A,
- *objB* be another O2DBI-II object of class B that has another data source and thus another O2DBI-II master than *objA*, and
- class A has an attribute *refB* that contains external references to objects of class B

then

- should a statement like *objA*→*refB*(*objB*) write an external reference to *objB* in form of some unique identifier into the data representation of *objA* and
- the statement *objA*→*refB*() should return the object represented by the referencing identifier in *objA*. The returned object is then equivalent to *objB* and it belongs in particular to another O2DBI-II master than *objA*.

It is clear that the major advantage of this approach is the availability of full featured objects that provide access to their complete functionality. Once a referenced object has been initialized automatically from a reference, all methods are accessible that even allow the direct manipulation of the object properties. Certainly, this also includes the automated and specific visualization of an object by standard routines that are provided by the object itself (e.g. each referenceable object could implement a method *show* or *display*).

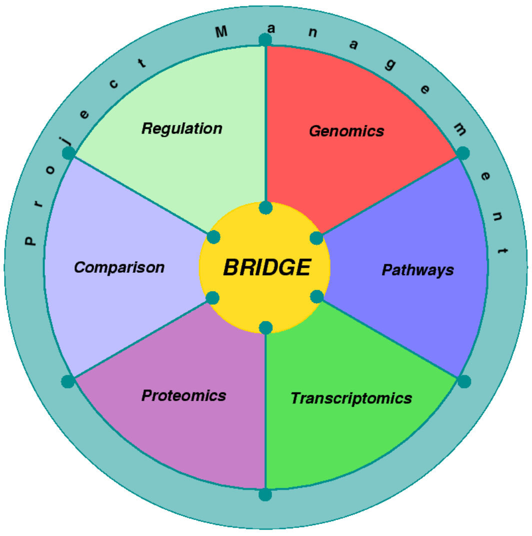


Figure 6.3.: Design of the BRIDGE system: a central module (BRIDGE) organizes different specialized components while the project management system controls all data access. For loading a component and for establishing a connection to a data source, the BRIDGE layer communicates with the project management in order to locate the data source and for checking the access privileges.

Nevertheless, for realizing the functionality described above, a general controlling mechanism is required for initializing the referenced objects from the proper project or data source. As displayed in figure 6.3, a project management system could be used as a comprehensive master or control unit to organize separate projects and maintain relationships between them. Furthermore, this project management system could be helpful for the administration of users that have different levels of access to project-related data (e.g. by assigning the required SQL privileges).

By encapsulating all data access aspects in a separate layer, the complete backend for storing the information can be implemented regardless of the graphical user frontends or other client APIs. All direct interactions are then left to the BRIDGE layer that has to provide standardized mechanisms for data exchange and communication.

Figure 6.4 illustrates a modified three-tier architecture with the optional BRIDGE layer on top of the O2DBI-II business classes. This module is responsible for resolving references to external objects. Access for individual users to different projects is controlled by a project management system that stores all required information for connecting to a project and for retrieving requested objects.

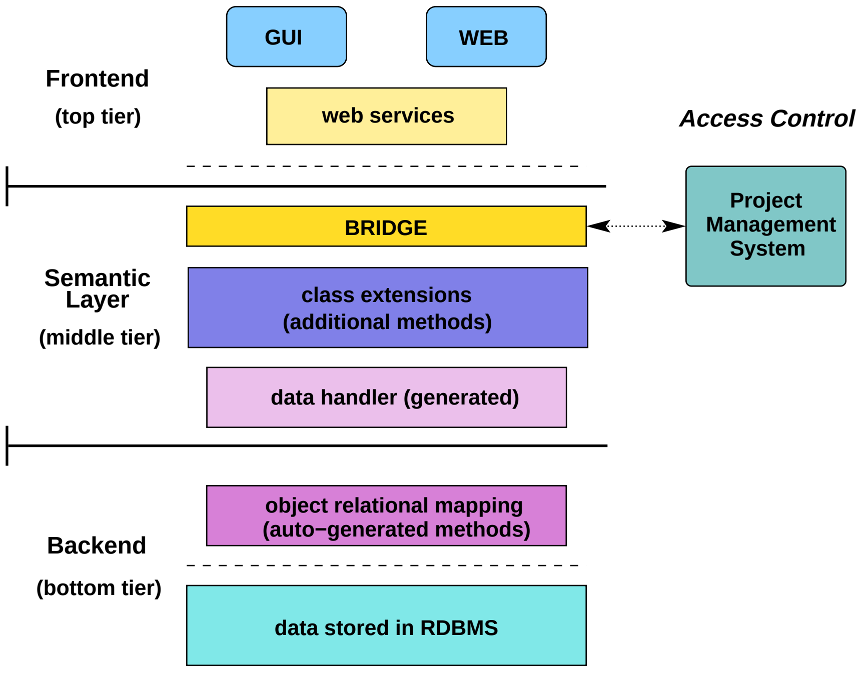


Figure 6.4.: In addition to the standard three-tier architecture, we have introduced a separate BRIDGE layer for connecting different data sources. All access to data sources stored in the backends is controlled via a project management system.

Another future extension that is already shown here is an additional layer that provides access to more and more widely used web services. Such a module on top of the BRIDGE layer can be used to integrate the data of each specialized component into almost any other application that features an interface for such web services.

CHAPTER 7

Specialized components

Currently, three specialized components have been developed in collaborative projects that can be “plugged” into the BRIDGE system: GenDB [MGM+03], EMMA [DGB+03], and GOPArc (unpublished). Instead of running each component as a separate tool, they can be integrated into a common platform via the BRIDGE where the modules can communicate directly with each other (see chapter 8). In the following sections these specialized modules are described in more detail.

7.1. GenDB-2.0

GenDB [MGM+03] is an open source genome annotation system for prokaryotic genomes. Hierarchical regions (e.g. contigs, CDSs, ESTs), observations on these regions (e.g. BLAST results), and manual or human annotations are stored in a relational database and form the three main building blocks of the system. Beyond various navigation metaphors (contig view, circular and linear plot, virtual 2D gel, etc.), GenDB also offers different export facilities (e.g. GenBank, EMBL). The GenDB system can be used as a repository for many different kinds of sequences and since all regions stored in the database have a unique identifier, other systems can easily link their information to a region by referring to this identifier.

GenDB-2.0 has been developed based on the GenDB-1 system as described by F. Meyer in his PhD thesis [Mey01]. The substantial extensions of the data model required a major re-design of the complete software, including the replacement of the O2DBI-I database backend by the improved O2DBI-II system. At the same time, widely used general purpose functions such as methods for parsing a FASTA file or writing an EMBL file, etc. were sourced out into separate "common" modules (see appendix A.6) in order to increase their reusability in other projects.

7.1.1. Data model design

Similar to its predecessor, GenDB-2 is based on a data model with three core types of objects. **Regions** describe arbitrary (sub-) sequences. A region can be related to a parent region, e.g. a CDS is part of a contig. **Observations** correspond to information computed by various tools (e.g. BLAST or InterPro) for those regions. **Annotations** store the interpretation of a (human) annotator. They describe regions based on the evidence stored in the observations. Figure 7.1 shows the relationships between the different core objects. As can be seen, there is a clear distinction between the results from various bioinformatics tools (observations) and their interpretation (annotations) which was implemented in the data model. While this data model seems very generic, it represents a hierarchy of classes, including the complete EMBL feature set for prokaryotes with several extensions (see figure 7.2).

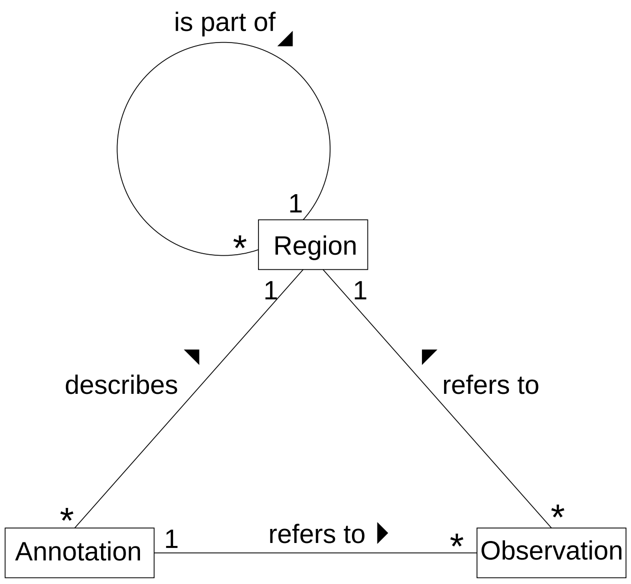


Figure 7.1.: The core data model of GenDB in UML. Only the three central classes are shown, the classes actually represent a hierarchy of specialized objects, e.g. a BLAST observation object and an InterPro observation object.

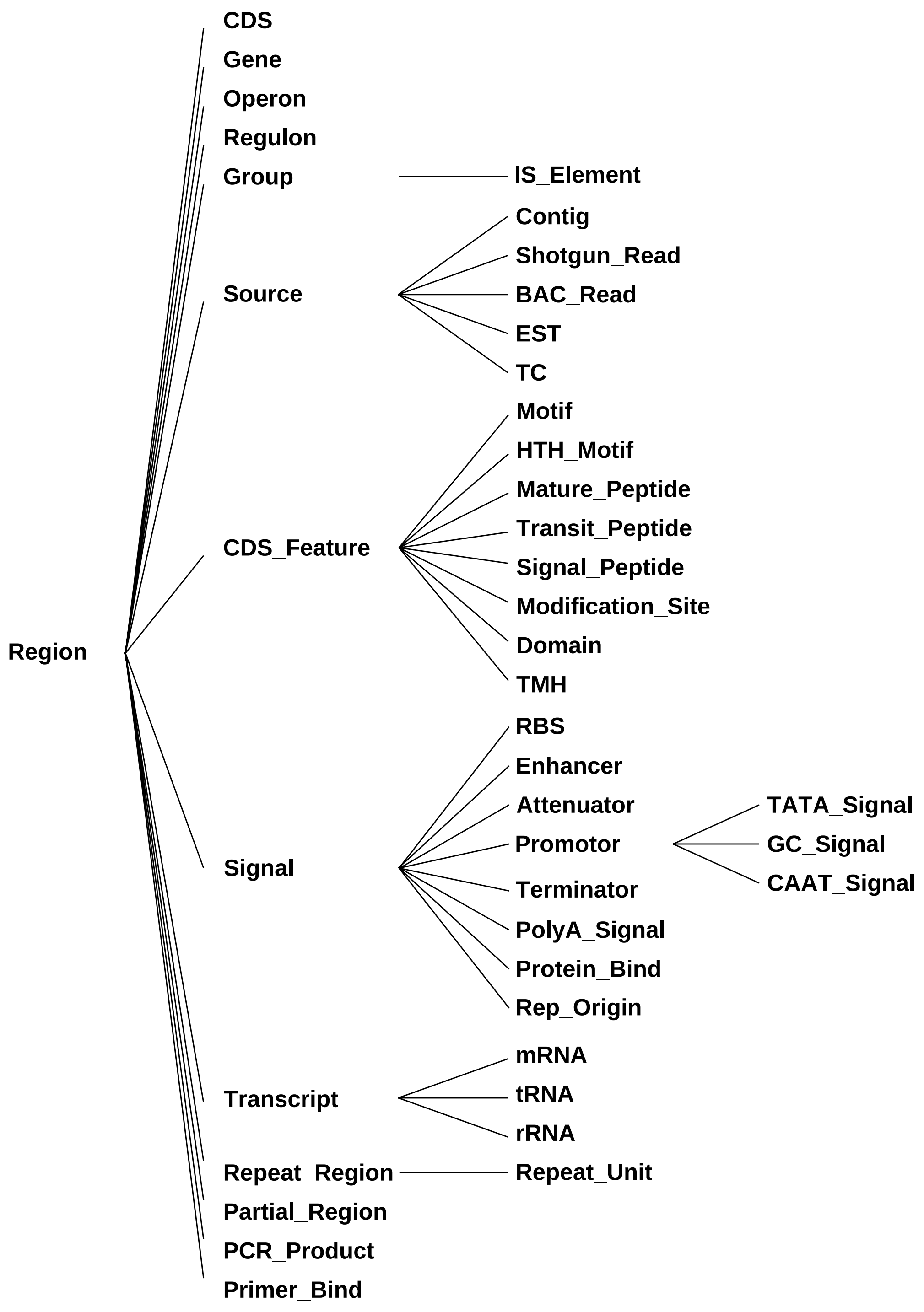


Figure 7.2.: The hierarchy of regions implemented in GenDB-2.0.

There are additional classes (e.g. tools and annotators) that complement the three core classes. Since data access is implemented via the objects described above, the classes in GenDB themselves form the API. This object-oriented approach makes code maintenance easy and also the data and methods in the system accessible to other programs. At the same time, these classes provide a means to extend the GenDB system. Although GenDB is currently limited to analyzing prokaryotic genomes, it would only require small extensions in the data model and the integration of other gene prediction tools to support the analysis of eukaryotic genomes.

7.1.2. General overview

Figure 7.3 illustrates the architecture of the GenDB system with the main building blocks.

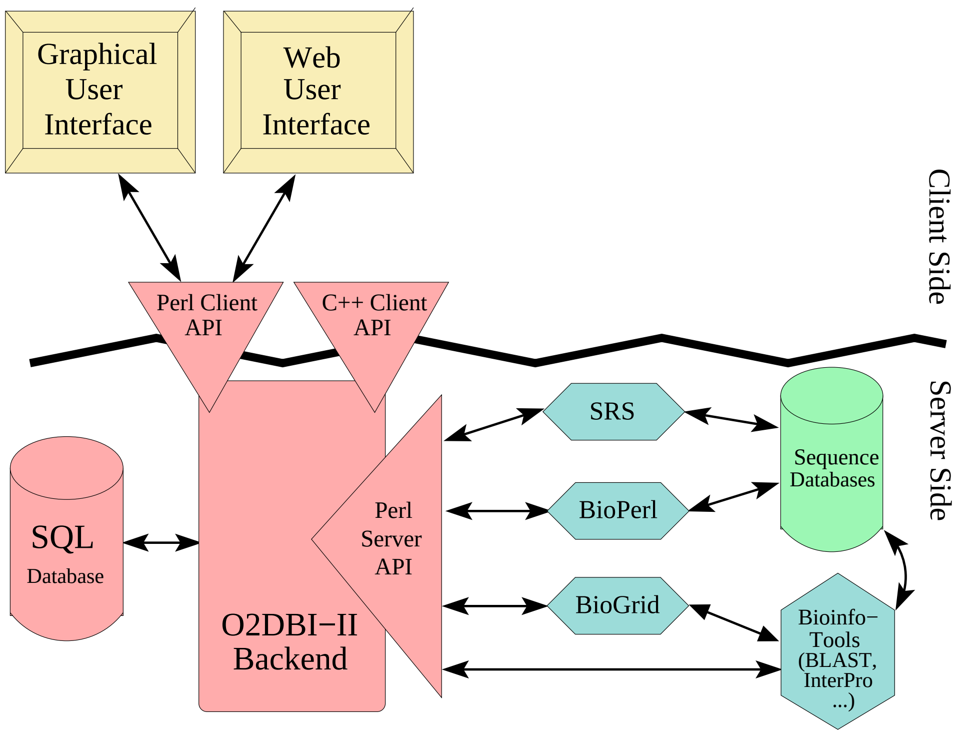


Figure 7.3.: On the server side all data that is stored in a relational database management system can be accessed using the O2DBI-II modules. The Perl server API is basically used for the integration of sequence databases and bioinformatics tools. On the client side, different user frontends are implemented that access the system via a Perl or C++ client API.

The complexity of the system encourages using an object-oriented approach not only in designing (see figure 7.1) but also in implementing the system. Therefore, the enhanced O2DBI-II system was used to map Perl objects automatically to relational tables. The GenDB objects are mapped onto tables via O2DBI-II and stored in a relational database (here MySQL) as described in section 5.2. All access to the data via a Perl client or server API, or via a C++ client interface is managed again by the O2DBI-II module.

On the client side, user interfaces have been implemented that use the functionality of these APIs. On the server side, sequence databases can be accessed via the SRS system or via the BioPerl interfaces. Computation-intensive tools like BLAST or InterPro can be managed and scheduled via a BioGrid as described below (e.g. Sun GridEngine[1]).

Currently, the developers' version of GenDB-2.0 which is maintained via CVS[2], comprises more than 200 modules and more than 50,000 lines of Perl source code, not including more than 20 common modules that have been implemented for a number of general purpose tasks (e.g. for translating or reversing a DNA sequence, see appendix A.6).

7.1.3. Integration of tools

One major improvement of the GenDB system in comparison to the first version, is the modular concept for the integration of bioinformatics tools (e.g. BLAST). GenDB allows the incorporation of arbitrary programs for different kinds of bioinformatics analysis. According to the system design, each of these programs is integrated as a *Tool* (e.g. *Tool::Function::Blast*), which creates observations for a specific kind of region. A job that can be submitted to the scheduling system thus contains the information about a valid tool and region combination as illustrated in figure 7.4.

For most tools, GenDB-2.0 also features simple automatic annotators that can be activated. They are started upon completion of a tool run and create automatic annotations employing a simple "best hit" strategy based on the observations created by the tool run.

For an automated large scale computation of various bioinformatics tools, a scalable framework was developed and implemented which allows a batch submission of thousands of *Jobs* in a very simple manner. Therefore, the following steps have to be performed (see appendix A.4 for further details):

1. The desired *Jobs* have to be created, e.g. for region or function prediction by using the *JobSubmitter Wizard*. This can be done quite easily with the *submit_job.pl* script or via the graphical user interface. For all valid region and tool combinations as defined by the user, the requested *Jobs* will be created and stored in the GenDB project database. Initially, these new *Jobs* will then have the status *PENDING*.

[1] *http://www.sun.com/gridware*
[2] Concurrent Versions System

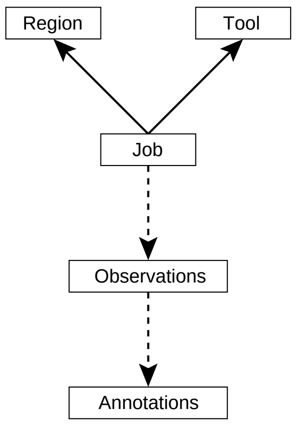


Figure 7.4.: The tool concept in GenDB-2.0. Jobs contain the information about valid tool/region combinations. Executing a tool on a given region thus creates observations for that region and in some cases also automatic annotations.

2. Before the *submit_job.pl* script finishes, it calls the *submit* method of the *JobSubmitter Wizard*. Thus, all previously created *Jobs* will be registered as a *Job Array* in the *Scheduler::Codine* using the *Scheduler::Codine–>freeze* method. Finally, the array of all *Jobs* is submitted by calling *Scheduler::Codine–>thaw*. All *Jobs* should now have the status *SUBMITTED* and a queue of *Jobs* should appear in the status report of the Sun GridEngine's `qstat` output.

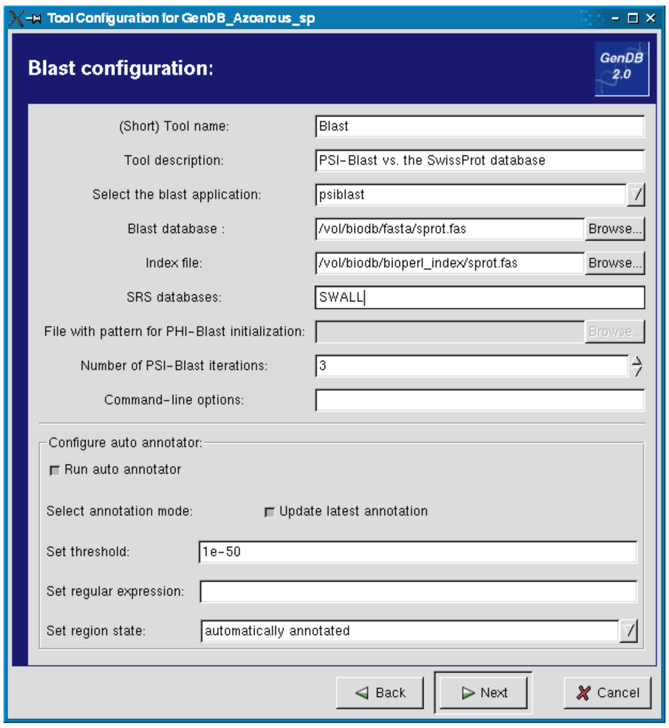
3. In the previous step, each *Job* was submitted to the scheduler by adding the command line for each single *Job* computation to the list of *Jobs*. Actually, the script *runtool.pl* is called for each *Job* with the corresponding arguments such as `runtool.pl -p <projectname> -j <jobid> [-a]`.

4. When such a command line is executed by one of the compute hosts, the script *runtool.pl* tries to initialize the *Job* object for the given id and project name. Since a *Job* contains the information about a specific region and a single tool that should be computed for that region, this script can now execute the *run* method that has to be defined for each tool. Such a *run* method normally starts a bioinformatics tool (e.g. BLAST, Pfam, InterPro) for the given region and stores some observations for the results obtained. During this computation the status of the current *Job* is *RUNNING*. If the option `-a` was specified an automatic annotation will be started upon successful computation of the tool. These are only very simple automatic annotations since they are based on the results of a single tool and region combination. Whenever the computation itself or the automatic annotation fails, the status of a *Job* is set to *FAILED*, otherwise the status is *FINISHED* and the computation is complete.

The inclusion of new tools in GenDB is very easy, with the most time-consuming step typically being the implementation of a parser for the result files. For the prediction of regions, such as coding sequences (CDS) or tRNAs, GLIMMER, CRITICA, and tRNAscan-SE have been integrated into the system.

Homology searches on DNA or amino acid level in arbitrary sequence databases can be done using the BLAST program suite. In addition to using HMMer for motif searches, we also search the BLOCKS and InterPro databases to classify sequence data based on a combination of different kinds of motif search tools. A number of additional tools have been integrated for the characterization of certain features of coding sequences, such as TMHMM for the prediction of α-helical transmembrane regions, SignalP for signal peptide prediction, or CoBias [MKPM04] for analyzing trends in codon usage.

Since all tools have to be defined separately for each project, a tool configuration wizard was implemented to support this task (see figure 7.5).



Blast configuration:

(Short) Tool name:	Blast
Tool description:	PSI-Blast vs. the SwissProt database
Select the blast application:	psiblast
Blast database :	/vol/biodb/fasta/sprot.fas
Index file:	/vol/biodb/bioperl_index/sprot.fas
SRS databases:	SWALL
File with pattern for PHI-Blast initialization:	
Number of PSI-Blast iterations:	3
Command-line options:	

Configure auto annotator:

☐ Run auto annotator

Select annotation mode:	☐ Update latest annotation
Set threshold:	1e-50
Set regular expression:	
Set region state:	automatically annotated

Figure 7.5.: The tool configuration wizard in GenDB-2.0. In this example, a BLAST tool can be configured for blasting against a special database (e.g. PSI-BLAST vs. the SwissProt database). Additionally, an automatic annotator can be activated which automatically annotates a function based on a simple cut-off strategy that can be combined with a check for a regular expression.

Whereas some tools only return a numeric score and/or an E-value as a result, other tools like BLAST or HMMer additionally provide more detailed information, such as an alignment.

Although the complete tool results are available to the annotator, only a minimum data subset is stored in form of observations. Based on this subset, the complete tool result record can be recomputed on demand. Storing only a minimal subset of data reduces the storage demands by two orders of magnitude when compared to the traditional “store everything” approach. Our performance measurements have shown this also to be more time efficient than data retrieval from a disk subsystem for any realistic genome project (see also [Mey01]).

7.1.4. Data navigation metaphors

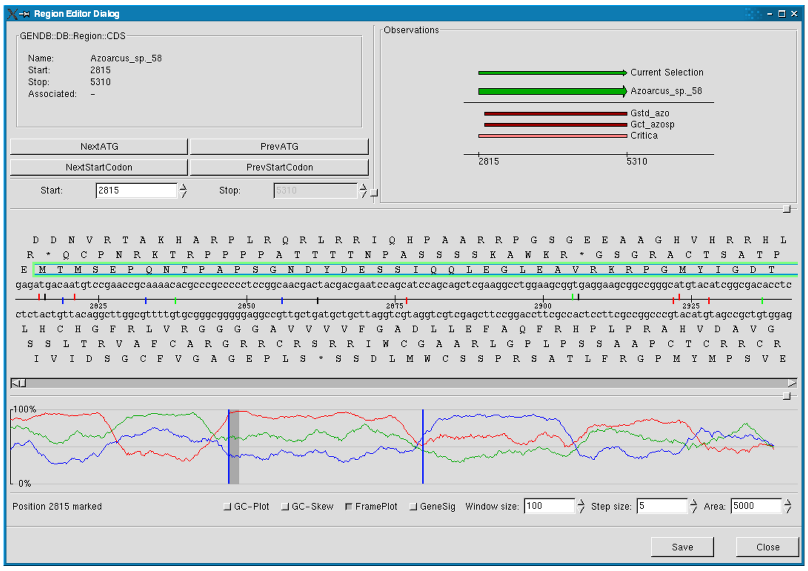
The design of the GenDB systems allows the projection of data from any component or plug-in onto all views (see also figures 7.7, 7.9, or 7.12). This allows the user to navigate through the genome with a wide variety of synchronized views. Sequence information is displayed at the level of contigs but also for each sub-region. For each region, a report can be generated that summarizes all properties (e.g. start, stop, length), available observations, and the latest annotation. Additionally, specialized views like the circular or linear plot or the virtual 2D gel complement the navigation metaphors.

7.1.5. Plug-in architecture

As all data in the system is accessible, almost any task can be performed by a plug-in, defined as a tool that operates on the GenDB data structures. While the core GenDB system provides a mechanism for manual annotation, two automatic annotation plug-ins perform automatic assignment of regions (e.g. genes) and/or functional annotations for those regions. Another plug-in can be used to colorize all kinds of regions in a user defined scheme that has to be given a list with start and stop positions and a color. A similar plug-in can be used to create additional circles in the circular genome plot.

7.1.6. Wizards

Repetitive tasks like updating the position of every downstream gene after a frame-shift correction are performed by “wizards”. These are software agents, modeling repetitive tasks and/or tasks that require complex and synchronized changes to different data objects.



GENDB::DB::Region::CDS

Name: Azoarcus_sp._58
Start: 2815
Stop: 5310
Associated: –

Start: 2815 Stop: 5310

```
  D  D  N  V  R  T  A  K  H  A  R  P  L  R  Q  R  L  R  R  I  Q  H  P  A  A  R  R  P  G  S  G  E  E  A  A  G  H  V  H  R  R  H  L
 R  *  Q  C  P  N  R  K  T  R  P  P  P  P  A  T  T  T  T  N  P  A  S  S  S  S  K  A  W  K  R  *  G  S  G  R  A  C  T  S  A  T  P
E  M  T  M  S  E  P  Q  N  T  P  A  P  S  G  N  D  Y  D  E  S  S  I  Q  Q  L  E  G  L  E  A  V  R  K  R  P  G  M  Y  I  G  D  T
gagatgacaatgtccgaaccgcaaaacacgcccgccccctccggcaacgactacgacgaatccagcatccagcagctcgaaggcctggaagcggtgaggaagcggccgggcatgtacatcggcgacacctc
ctctactgttacaggcttggcgttttgtgcgggcgggggaggccgttgctgatgctgcttaggtcgtaggtcgtcgagcttccggaccttcgccactccttcgccggcccgtacatgtagccgctgtggag
 L  H  C  H  G  F  R  L  V  R  G  G  G  G  A  V  V  V  V  F  G  A  D  L  L  E  F  A  Q  F  R  H  P  L  P  R  A  H  V  D  A  V  G
 S  S  L  T  R  V  A  F  C  A  R  G  R  R  C  R  S  R  R  I  W  C  G  A  A  R  L  G  P  L  P  S  S  A  A  P  C  T  C  R  R  C  R
  I  V  I  D  S  G  C  F  V  G  A  G  E  P  L  S  *  S  S  D  L  M  W  C  S  S  P  R  S  A  T  L  F  R  G  P  M  Y  M  P  S  V  E
```

Position 2815 marked

☐ GC-Plot ☐ GC-Skew ☑ FramePlot ☐ GeneSig Window size: 100 Step size: 5 Area: 5000

Figure 7.6.: The CDS-start correction wizard. The evidence (observations) computed for these regions and alternative predicted regions are presented to the user.

All actions performed using wizards are modeled as annotations. Currently, wizards are implemented for frame-shift and sequence data correction, CDS-start correction and reloading (updating) of contig sequences, and for comfortably submitting multiple jobs (e.g. computation of all tools for all regions or only selected tool/region combinations, refer to appendix A.4 for furhter details).

7.1.7. Annotation

As already mentioned, the GenDB data model features a strict separation of tool results (observations) and their interpretation (annotation). This confers a great amount of flexibility and enables researchers to freely define their application-specific annotation strategies. The GenDB system supports both manual annotation as well as the application of automated annotation strategies. For manual annotation, the user interface provides a “one-click” infrastructure; for automatic annotation the API can be used.

The core GenDB system offers simple automatic annotation functions which allow the application of user-defined “best tool result” strategies. In addition to this, a GenDB-Annotate plug-in for more complex annotation strategies based on the integration of an expert system is currently under development. There, the user can define a set of rules to be used for automatic annotation of regions or assignment of function to those regions. Due to the consistent internal data representation, all GenDB objects can be accessed directly by an expert system. While currently implementing a new annotation strategy entails writing program code, we are in the process of establishing a graphical editor (with XML export capabilities) for editing of annotation rules and a processor for computing annotations based on these rules.

For annotation projects, the linear contig with its list of genes often is only a starting point. The knowledge about metabolic pathways and the enzymes contained in them is connected to the data in GenDB via the GOPArc (Gene Ontology and Pathway Architecture) module (see section 7.3).

7.1.8. Data import and export

An important step for any genome analysis project is the availability of good import and export facilities in the genome annotation system. Currently, the GenDB system allows data import/export from/to GenBank, EMBL, and FASTA format files; an additional export format is GFF. A user configurable linear or circular whole genome view (see figures 7.9 and 7.10) which can be exported as a PNG or Scalable Vector Graphics (*SVG*) file complements the export formats. For each gene annotated with GenDB, the gene report can be also be generated in the printable PostScript format.

7.1.9. Interfaces

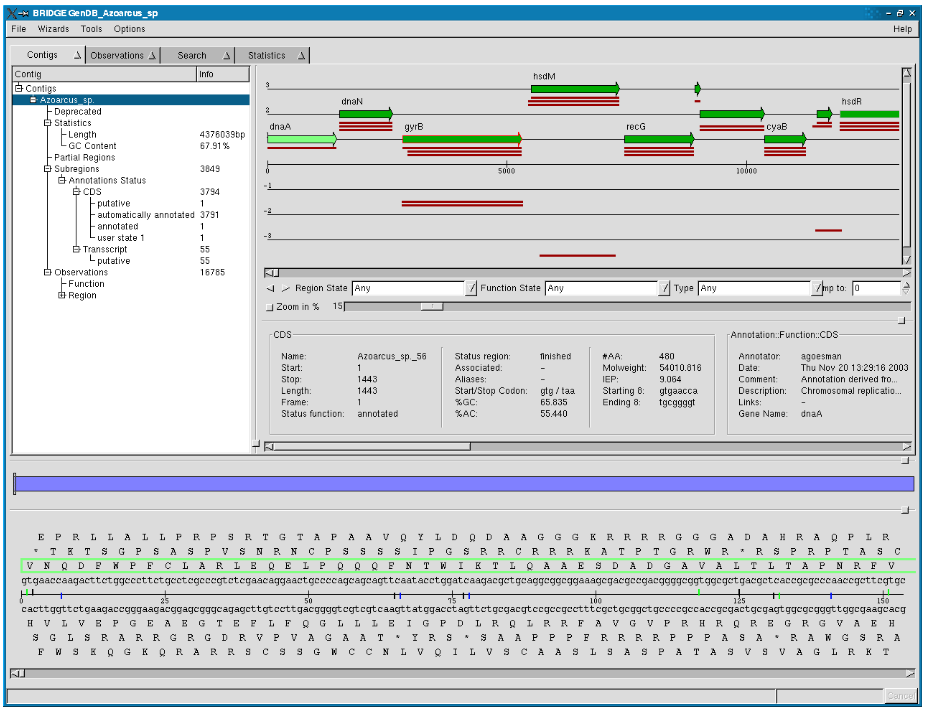
There are various comfortable ways of accessing the system: an API, graphical user interfaces, and a new C++ client-server interface. The more widely used frontend is a Gtk-Perl[3] based graphical user interface (GUI) that offers access to the data in the system by a variety of navigation metaphors (see figures on the following pages). Since not all users have access to a platform with Perl/Gtk, a web interface is also provided. The latter offers a somewhat restricted functionality with respect to the GUI. But due to its HTML standard compliance, the web interface provides access to GenDB for a wide range of platforms.

As stated above, the GenDB classes form the applications programmers interface (API). Documentation of each class and object property or method is available on the GenDB web site. The relative simplicity of the object model together with the documentation have led more than 30 research groups to use GenDB as a platform for their work. The web site has some sample scripts that show the functionality of the GenDB API. Using this interface, programmers are able to extract or manipulate the GenDB data objects. For example, this allows the user to write simple Perl scripts that compute the molecular weight for every protein in a given genome and generate a table (see also section A.3).

In addition to the Perl API, O2DBI-II supports a client-server programmer's interface. This will not only allow non-Perl platforms to connect to the GenDB system, but also clients to run on remote machines.

[3] *http://www.gtkperl.org/*

The GenDB-2.0 Gtk GUI The following screenshots show some selected interfaces of the graphical user interface implemented with Gtk.



```
Contig                          Info
⊟ Contigs
  ⊟ Azoarcus_sp.
    ├ Deprecated
    ⊟ Statistics
    │  ├ Length                     4376039bp
    │  └ GC Content                 67.91%
    ├ Partial Regions
    ⊟ Subregions                    3849
    │  ⊟ Annotations Status
    │     ⊟ CDS                     3794
    │     │  ├ putative             1
    │     │  ├ automatically annotated 3791
    │     │  ├ annotated            1
    │     │  └ user state 1         1
    │     ⊟ Transscript             55
    │        └ putative             55
    ⊟ Observations                  16785
       ├ Function
       ⊞ Region
```

```
Region State Any    Function State Any    Type Any    mp to: 0
Zoom in %   15
```

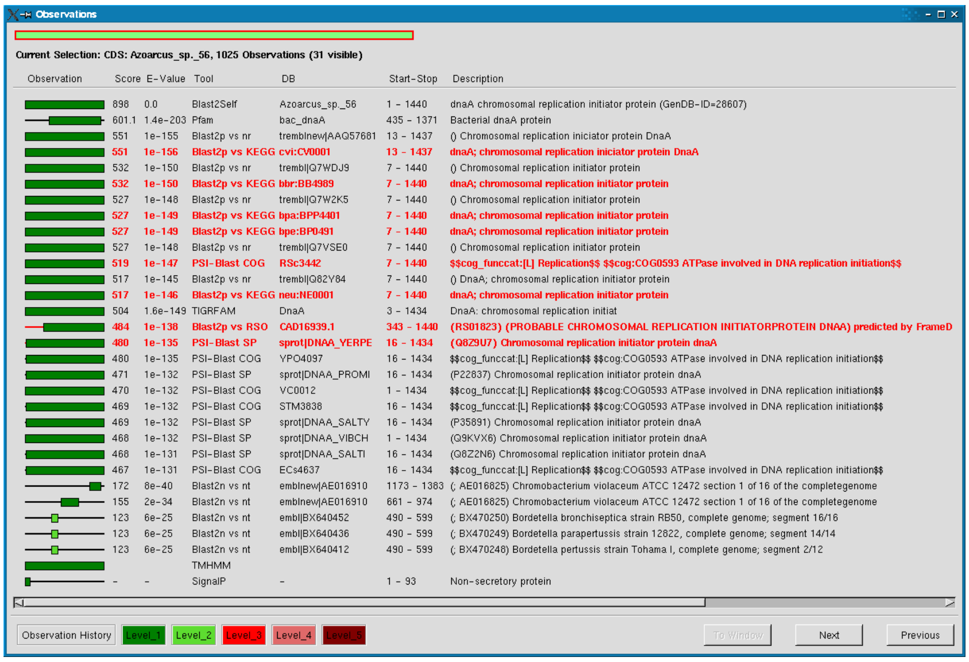
```
CDS
Name:             Azoarcus_sp._56   Status region:      finished    #AA:         480
Start:            1                 Associated:         -           Molweight:   54010.816
Stop:             1443              Aliases:            -           IEP:         9.064
Length:           1443              Start/Stop Codon:   gtg / taa   Starting 8:  gtgaacca
Frame:            1                 %GC:                65.835      Ending 8:    tgcggggt
Status function:  annotated         %AC:                55.440

Annotation::Function::CDS
Annotator:    agoesman
Date:         Thu Nov 20 13:29:16 2003
Comment:      Annotation derived fro...
Description:  Chromosomal replicatio...
Links:        -
Gene Name:    dnaA
```

```
 E  P  R  L  L  A  L  L  P  R  P  S  R  T  G  T  A  P  A  A  V  Q  Y  L  D  Q  D  A  A  G  G  G  K  R  R  R  R  G  G  G  A  D  A  H  R  A  Q  P  L  R
 *  T  K  T  S  G  P  S  A  S  P  V  S  N  R  N  C  P  S  S  S  S  I  P  G  S  R  R  C  R  R  R  K  A  T  P  T  G  R  W  R  *  R  S  P  R  P  T  A  S  C
V  N  Q  D  F  W  P  F  C  L  A  R  L  E  Q  E  L  P  Q  Q  Q  F  N  T  W  I  K  T  L  Q  A  A  E  S  D  A  D  G  A  V  A  L  T  L  T  A  P  N  R  F  V
gtgaaccaagacttctggcccttctgcctcgcccgtctcgaacaggaactgccccagcagcagttcaatacctggatcaagacgctgcaggcggcggaaagcgacgccgacggggcggtggcgctgacgctcaccgcgcccaaccgcttcgtgc
0                        25                       50                       75                       100                      125                      150
cacttggttctgaagaccgggaagacggagcgggcagagcttgtccttgacggggtcgtcgtcaagttatggacctagttctgcgacgtccgccgcctttcgctgcggctgccccgccaccgcgactgcgagtggcgcgggttggcgaagcacg
 H  V  L  V  E  P  G  E  A  E  G  T  E  F  L  F  Q  G  L  L  L  E  I  G  P  D  L  R  Q  L  R  R  F  A  V  G  V  P  R  H  R  Q  R  E  G  R  G  V  A  E  H
  S  G  L  S  R  A  R  R  G  R  G  D  R  V  P  V  A  G  A  A  T  *  Y  R  S  *  S  A  A  P  P  P  F  R  R  R  R  P  P  P  A  S  A  *  R  A  W  G  S  R  A
   F  W  S  K  Q  G  K  Q  R  A  R  R  S  C  S  S  G  W  C  C  N  L  V  Q  I  L  V  S  C  A  A  S  L  S  A  S  P  A  T  A  S  V  S  V  A  G  L  R  K  T
```

Figure 7.7.: The main window of GenDB-2.0 can be used for navigating through a contig that can be selected from the list at the left. Available regions and observations for such regions are displayed and all properties of a selected region can be shown. The absolute positions of selected regions (e.g. from a pathway) can be highlighted on the complete contig and a separate area at the bottom of the main window shows the current range of the sequence.

As illustrated in figure 7.7, a tree-view at the left side of the main window displays the contigs that have been imported. In addition to the length, *GC content*, and other standard information, a statistical overview shows among other data the number of CDS regions for each different status (e.g. putative or annotated CDS). The contig view on top of the right side displays all kinds of sub-regions and observations that predict those regions. Each type of region can be assigned a color and the user can select for each type whether it should be shown and have an arrow or not. Moving the mouse over a region opens a window that zooms into the region and shows the best observation of each tool. A small navigation bar below the region area can be used to quickly jump to the next region with a specific status or to a given position.

Another important element is the variable information frame below the navigation bar: here, the user can view a list of regions, a plotting interface (e.g. *GC content*, *GC skew*, or *frame plot*), or a sheet with all available information about a selected region. The contig overview in the middle displays all contigs. The current contig is enlarged and a sliding window can be used to move along the sequence. The sequence window at the bottom shows the DNA and amino-acid sequence with all sub-regions.



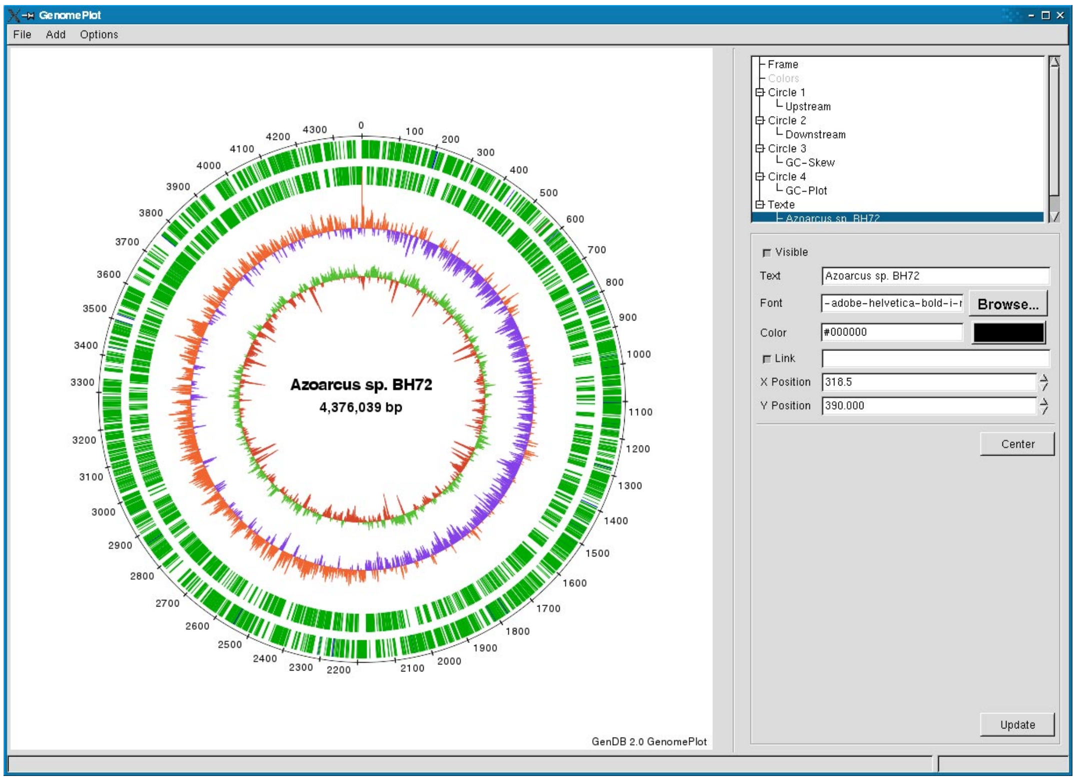
Current Selection: CDS: Azoarcus_sp._56, 1025 Observations (31 visible)

Observation	Score	E-Value	Tool	DB	Start-Stop	Description
	898	0.0	Blast2Self	Azoarcus_sp._56	1 - 1440	dnaA chromosomal replication initiator protein (GenDB-ID=28607)
	601.1	1.4e-203	Pfam	bac_dnaA	435 - 1371	Bacterial dnaA protein
	551	1e-155	Blast2p vs nr	tremblnew|AAQ57681	13 - 1437	() Chromosomal replication iniciator protein DnaA
	551	**1e-156**	**Blast2p vs KEGG**	**cvi:CV0001**	**13 - 1437**	**dnaA; chromosomal replication iniciator protein DnaA**
	532	1e-150	Blast2p vs nr	trembl|Q7WDJ9	7 - 1440	() Chromosomal replication initiator protein
	532	**1e-150**	**Blast2p vs KEGG**	**bbr:BB4989**	**7 - 1440**	**dnaA; chromosomal replication initiator protein**
	527	1e-148	Blast2p vs nr	trembl|Q7W2K5	7 - 1440	() Chromosomal replication initiator protein
	527	**1e-149**	**Blast2p vs KEGG**	**bpa:BPP4401**	**7 - 1440**	**dnaA; chromosomal replication initiator protein**
	527	**1e-149**	**Blast2p vs KEGG**	**bpe:BP0491**	**7 - 1440**	**dnaA; chromosomal replication initiator protein**
	527	1e-148	Blast2p vs nr	trembl|Q7VSE0	7 - 1440	() Chromosomal replication initiator protein
	519	**1e-147**	**PSI-Blast COG**	**RSc3442**	**7 - 1440**	**$$cog_funccat:[L] Replication$$ $$cog:COG0593 ATPase involved in DNA replication initiation$$**
	517	1e-145	Blast2p vs nr	trembl|Q82Y84	7 - 1440	() DnaA; chromosomal replication initiator protein
	517	**1e-146**	**Blast2p vs KEGG**	**neu:NE0001**	**7 - 1440**	**dnaA; chromosomal replication initiator protein**
	504	1.6e-149	TIGRFAM	DnaA	3 - 1434	DnaA: chromosomal replication initiat
	484	**1e-138**	**Blast2p vs RSO**	**CAD16939.1**	**343 - 1440**	**(RS01823) (PROBABLE CHROMOSOMAL REPLICATION INITIATORPROTEIN DNAA) predicted by FrameD**
	480	**1e-135**	**PSI-Blast SP**	**sprot|DNAA_YERPE**	**16 - 1434**	**(Q8Z9U7) Chromosomal replication initiator protein dnaA**
	480	1e-135	PSI-Blast COG	YPO4097	16 - 1434	$$cog_funccat:[L] Replication$$ $$cog:COG0593 ATPase involved in DNA replication initiation$$
	471	1e-132	PSI-Blast SP	sprot|DNAA_PROMI	16 - 1434	(P22837) Chromosomal replication initiator protein dnaA
	470	1e-132	PSI-Blast COG	VC0012	1 - 1434	$$cog_funccat:[L] Replication$$ $$cog:COG0593 ATPase involved in DNA replication initiation$$
	469	1e-132	PSI-Blast COG	STM3838	16 - 1434	$$cog_funccat:[L] Replication$$ $$cog:COG0593 ATPase involved in DNA replication initiation$$
	469	1e-132	PSI-Blast SP	sprot|DNAA_SALTY	16 - 1434	(P35891) Chromosomal replication initiator protein dnaA
	468	1e-132	PSI-Blast SP	sprot|DNAA_VIBCH	1 - 1434	(Q9KVX6) Chromosomal replication initiator protein dnaA
	468	1e-131	PSI-Blast SP	sprot|DNAA_SALTI	16 - 1434	(Q8Z2N6) Chromosomal replication initiator protein dnaA
	467	1e-131	PSI-Blast COG	ECs4637	16 - 1434	$$cog_funccat:[L] Replication$$ $$cog:COG0593 ATPase involved in DNA replication initiation$$
	172	8e-40	Blast2n vs nt	emblnew|AE016910	1173 - 1383	(; AE016825) Chromobacterium violaceum ATCC 12472 section 1 of 16 of the completegenome
	155	2e-34	Blast2n vs nt	emblnew|AE016910	661 - 974	(; AE016825) Chromobacterium violaceum ATCC 12472 section 1 of 16 of the completegenome
	123	6e-25	Blast2n vs nt	embl|BX640452	490 - 599	(; BX470250) Bordetella bronchiseptica strain RB50, complete genome; segment 16/16
	123	6e-25	Blast2n vs nt	embl|BX640436	490 - 599	(; BX470249) Bordetella parapertussis strain 12822, complete genome; segment 14/14
	123	6e-25	Blast2n vs nt	embl|BX640412	490 - 599	(; BX470248) Bordetella pertussis strain Tohama I, complete genome; segment 2/12
			TMHMM			
	-	-	SignalP	-	1 - 93	Non-secretory protein

Figure 7.8.: The completely customizable ObservationView in GenDB-2.0 displays the results of bioinformatics tools that were computed for a selected region. This sortable list only shows some common attributes like the start, stop, description, etc. The complete result appears in a popup window that opens when the user moves the mouse over an observation.

The graphical user interface for the display of tool results is depicted in figure 7.8. Upon selection of a certain region, all available tool results for this region are visualized in a completely customizable list. The GenDB system uses different levels that are helpful for classifying and comparing the observations of different tools. Each level has a configurable color. While the sortable list contains only the most important information, a popup window shows the complete tool result. The complete original results of most tools (e.g. all BLAST alignments) are not stored in the database in order to save space. They can be recomputed on demand and presented to the user immediately. More information about the underlying database record is available by a cross-link to the corresponding sequence databases with the SRS system. The observations can also be used for a “one-click” annotation so that essential information like EC numbers or gene names are directly extracted from an observation and written into the corresponding fields of the annotation dialog.

The circular and linear plot (see figures 7.9 and 7.10) can be used to create highly customizable whole genome graphics. All regions are displayed according to the selected color scheme (type of region, status region, status function) but it is also possible to import a user defined scheme. In the circular plot, additional circles (e.g. for all tRNAs or for genes of a specific function) can be displayed by importing a list of start and stop positions for the regions on a new circle. The current version features a *GC content* and a *GC skew* plot and allows setting text labels at arbitrary positions. The plot can be saved to different image formats, including the SVG format.



Visible

Text: Azoarcus sp. BH72

Font: -adobe-helvetica-bold-i-r

Color: #000000

Link:

X Position: 318.5

Y Position: 390.000

Figure 7.9.: The integrated CircularPlot of GenDB-2.0 displays all coding sequences of a contig and other optional circles that can be imported. The *GC content* and the *GC skew* can be plotted optionally.

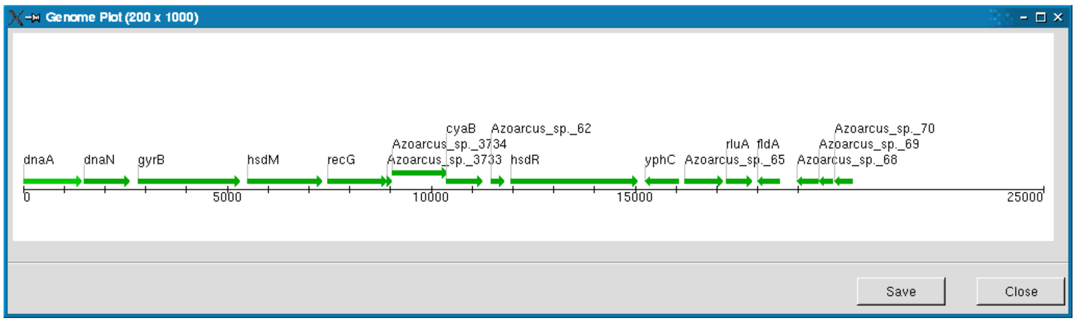


Figure 7.10.: The customizable linear plot of GenDB-2.0 can be used to visualize parts of the genome or even a complete contig. The plot can be scaled to user defined sizes in order to print out a whole genome on a poster or on postcard.

The linear plot displayed in figure 7.10 can be used to create printable images of selected genes or posters of whole genomes. All regions can be colored according to their status or in a user defined fashion. Optionally, the gene names or the original names of the regions can be displayed as well.

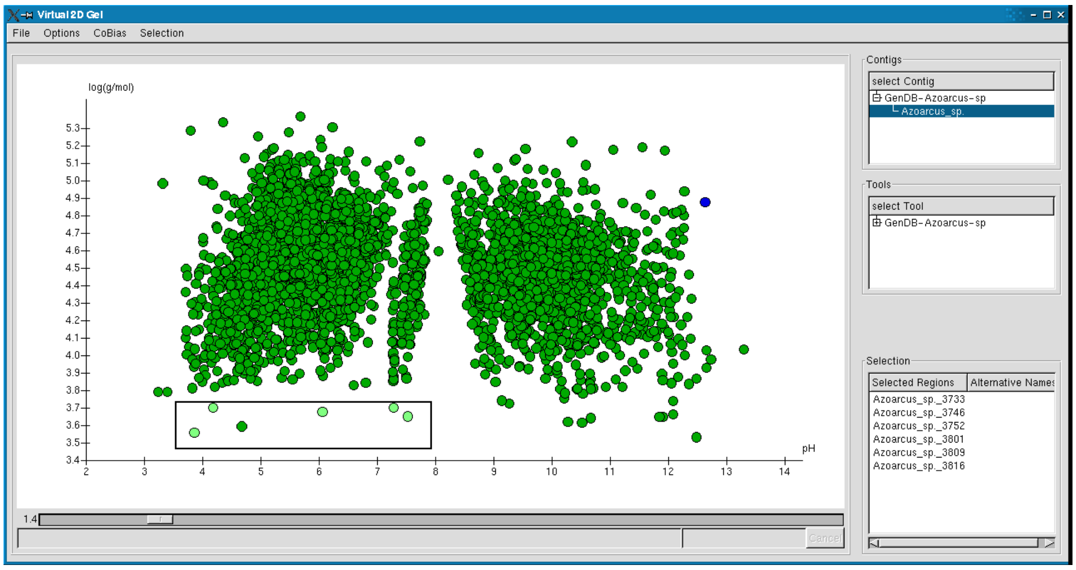
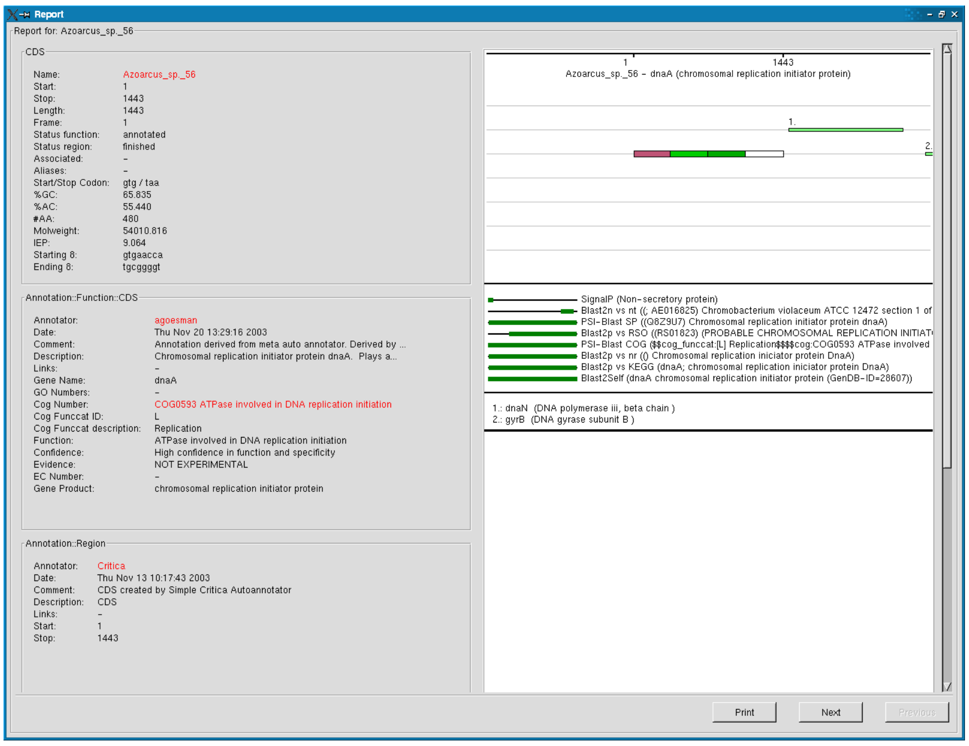


Figure 7.11.: The Virtual 2D Gel of GenDB-2.0 can be used to visualize all proteins of a contig according to their isoelectric point and molecular weight. Spots can be selected and the user can navigate to a region that corresponds to a spot.

Figure 7.11 shows a virtual 2D gel plotted with the theoretical values of the iso-electric point and the molecular weight computed for each CDS. Upon selection of a spot, the corresponding region is highlighted in the main window. The size of each spot can be adjusted depending on special predictions for the expected expression ratio (e.g. by using CoBias). It is also possible to select and display only those spots that have a signal peptide and/or a transmembrane helix and/or a helix-turn-helix motif.



Report for: Azoarcus_sp._56

CDS

Name:	Azoarcus_sp._56
Start:	1
Stop:	1443
Length:	1443
Frame:	1
Status function:	annotated
Status region:	finished
Associated:	–
Aliases:	–
Start/Stop Codon:	gtg / taa
%GC:	65.835
%AC:	55.440
#AA:	480
Molweight:	54010.816
IEP:	9.064
Starting 8:	gtgaacca
Ending 8:	tgcggggt

Annotation::Function::CDS

Annotator:	agoesman
Date:	Thu Nov 20 13:29:16 2003
Comment:	Annotation derived from meta auto annotator. Derived by ...
Description:	Chromosomal replication initiator protein dnaA. Plays a...
Links:	–
Gene Name:	dnaA
GO Numbers:	–
Cog Number:	COG0593 ATPase involved in DNA replication initiation
Cog Funccat ID:	L
Cog Funccat description:	Replication
Function:	ATPase involved in DNA replication initiation
Confidence:	High confidence in function and specificity
Evidence:	NOT EXPERIMENTAL
EC Number:	–
Gene Product:	chromosomal replication initiator protein

Annotation::Region

Annotator:	Critica
Date:	Thu Nov 13 10:17:43 2003
Comment:	CDS created by Simple Critica Autoannotator
Description:	CDS
Links:	–
Start:	1
Stop:	1443

1 1443
Azoarcus_sp._56 – dnaA (chromosomal replication initiator protein)

1.
2.

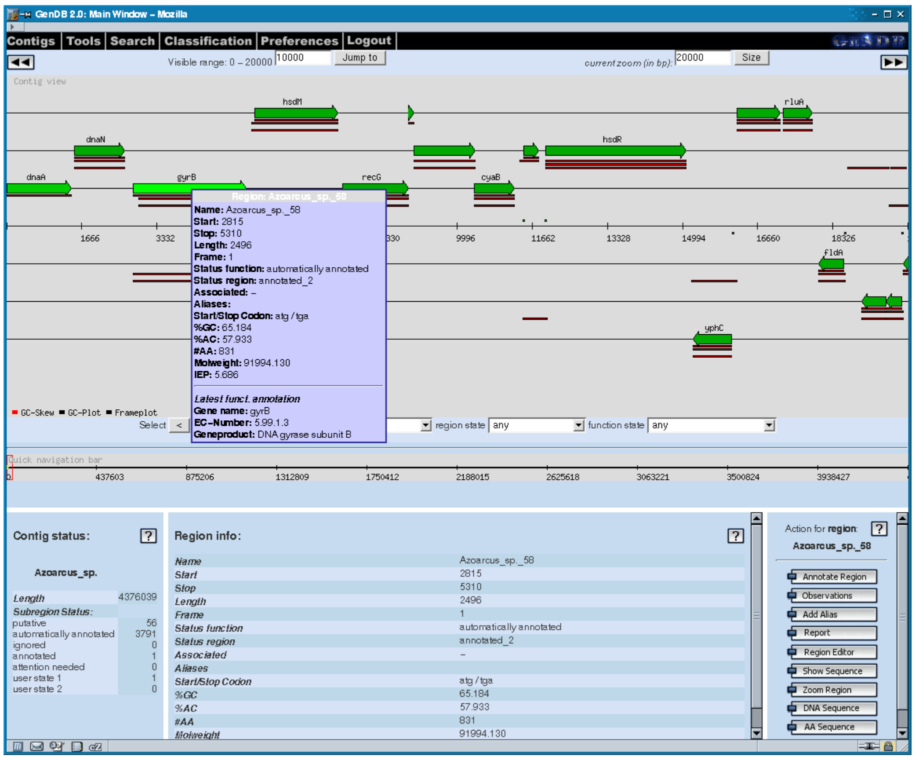
SignalP (Non–secretory protein)
Blast2n vs nt ((; AE016825) Chromobacterium violaceum ATCC 12472 section 1 of
PSI–Blast SP ((Q8Z9U7) Chromosomal replication initiator protein dnaA)
Blast2p vs RSO ((RS01823) (PROBABLE CHROMOSOMAL REPLICATION INITIAT
PSI–Blast COG ($$cog_funccat:[L] Replication$$$$cog:COG0593 ATPase involved
Blast2p vs nr (() Chromosomal replication iniciator protein DnaA)
Blast2p vs KEGG (dnaA; chromosomal replication iniciator protein DnaA)
Blast2Self (dnaA chromosomal replication initiator protein (GenDB–ID=28607))

1.: dnaN (DNA polymerase iii, beta chain)
2.: gyrB (DNA gyrase subunit B)

Figure 7.12.: The gene report in GenDB-2.0 shows all available information for a selected region in a printable format. It also displays the selected CDS and its surrounding regions as well as the best of observation of each bioinformatics tool that was computed for this region.

The new gene report of GenDB-2.0 displays a printable sheet with all available information about a selected CDS. As shown in figure 7.12, all CDS values (name, start, stop, length, etc.), the latest annotation region, the latest annotation function, and a zoomed image of the region are presented. The report also shows the best observation of each tool that has been computed for the current region.

The GenDB-2.0 web frontend In addition to the Gtk interface, GenDB-2.0 has a completely redesigned web frontend that can be used to annotate a genome with many researchers from different locations simultaneously.



Region: Azoarcus_sp._58
Name: Azoarcus_sp._58
Start: 2815
Stop: 5310
Length: 2496
Frame: 1
Status function: automatically annotated
Status region: annotated_2
Associated: –
Aliases:
Start/Stop Codon: atg / tga
%GC: 65.184
%AC: 57.933
#AA: 831
Molweight: 91994.130
IEP: 5.686

Latest funct. annotation
Gene name: gyrB
EC-Number: 5.99.1.3
Geneproduct: DNA gyrase subunit B

Contig status:

Azoarcus_sp.

Length	4376039
Subregion Status:	
putative	56
automatically annotated	3791
ignored	0
annotated	1
attention needed	0
user state 1	1
user state 2	0

Region info:

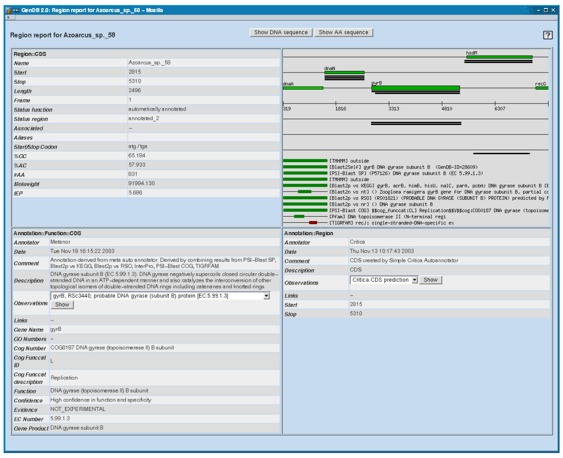
Name	Azoarcus_sp._58
Start	2815
Stop	5310
Length	2496
Frame	1
Status function	automatically annotated
Status region	annotated_2
Associated	–
Aliases	
Start/Stop Codon	atg / tga
%GC	65.184
%AC	57.933
#AA	831
Molweight	91994.130

Action for **region**:
Azoarcus_sp._58

Figure 7.13.: The main window of the web frontend in GenDB-2.0 can be used for navigating through a contig sequence. A popup window shows more details about a selected region, buttons in the lower right corner provide access to more context specific functions.

The main window of the GenDB-2.0 web interface looks quite similar to the Gtk GUI. For performance reasons, we did not include a sequence browser in the main window, but a separate window can be opened for viewing the DNA and amino-acid sequence. Three different navigation metaphors (COG, GO, and KEGG) can be opened via the "Classification" menu on top of this window. Since the right mouse button is already reserved for special popup menus of the web browser, we decided to emulate a context sensitive right mouse button popup menu in the lower right corner of the main window. Moving the mouse over one of the regions opens a popup window with the most important features of that region (e.g. name, length, gene name, gene product, EC number). The quick navigation bar in the middle of this window can be used to navigate to a specific position in the current contig.

The gene report displayed in figure 7.14 shows all available data about a selected CDS on a printable one page sheet. Standard properties of a CDS (e.g. start, stop, length) are listed as well as the latest annotation of the region and the latest annotation of the function. A graphical overview shows the best tool results and the neighbouring regions.



Region report for Azoarcus_sp._58

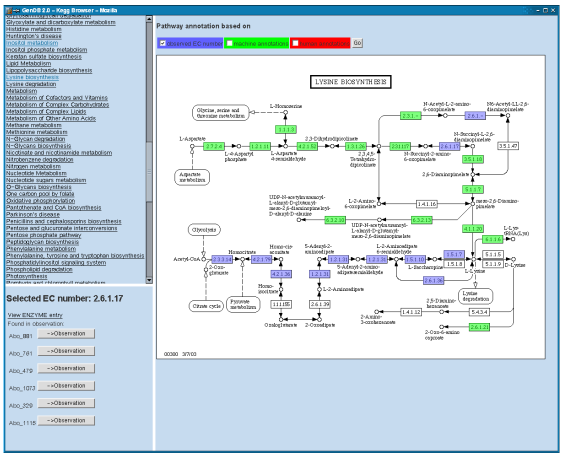
Region::CDS	
Name	Azoarcus_sp._58
Start	2815
Stop	5310
Length	2496
Frame	1
Status function	automatically annotated
Status region	annotated_2
Associated	–
Aliases	
Start/Stop Codon	atg / tga
%GC	65.184
%AC	57.933
#AA	831
Molweight	91994.130
IEP	5.686

```
[TMHMM] outside
[Blast2Self] gyrB DNA gyrase subunit B  (GenDB-ID=28609)
[PSI-Blast SP] (P57126) DNA gyrase subunit B (EC 5.99.1.3)
[TMHMM] outside
[Blast2p vs KEGG] gyrB, acrB, himB, hisU, nalC, parA, pcbA; DNA gyrase subunit B [E
[Blast2n vs nt] () Zoogloea ramigera gyrB gene for DNA gyrase subunit B, partial cd
[Blast2p vs RSO] (RS01821) (PROBABLE DNA GYRASE (SUBUNIT B) PROTEIN) predicted by F
[Blast2p vs nr] () DNA gyrase subunit B
[PSI-Blast COG] $$cog_funccat:[L] Replication$$¥$$cog:COG0187 DNA gyrase (topoisome
[Pfam] DNA topoisomerase II (N-terminal regi
[TIGRFAM] recJ: single-stranded-DNA-specific ex
```

Annotation::Function::CDS	
Annotator	Metanor
Date	Tue Nov 18 16:15:22 2003
Comment	Annotation derived from meta auto annotator. Derived by combining results from PSI–Blast SP, Blast2p vs KEGG, Blast2p vs RSO, InterPro, PSI–Blast COG, TIGRFAM.
Description	DNA gyrase subunit B (EC 5.99.1.3). DNA gyrase negatively supercoils closed circular double–stranded DNA in an ATP–dependent manner and also catalyzes the interconversion of other topological isomers of double–stranded DNA rings including catenanes and knotted rings.
Observations	gyrB, RSc3440; probable DNA gyrase (subunit B) protein [EC:5.99.1.3] Show
Links	–
Gene Name	gyrB
GO Numbers	–
Cog Number	COG0187 DNA gyrase (topoisomerase II) B subunit
Cog Funccat ID	L
Cog Funccat description	Replication
Function	DNA gyrase (topoisomerase II) B subunit
Confidence	High confidence in function and specificity
Evidence	NOT_EXPERIMENTAL
EC Number	5.99.1.3
Gene Product	DNA gyrase subunit B

Annotation::Region	
Annotator	Critica
Date	Thu Nov 13 10:17:43 2003
Comment	CDS created by Simple Critica Autoannotator
Description	CDS
Observations	Critica CDS prediction Show
Links	–
Start	2815
Stop	5310

Figure 7.14.: The report of the web frontend in GenDB-2.0 shows all available information about a CDS. The surrounding of the selected region and the best observation of each tool is displayed together with the latest annotations in a printable format.



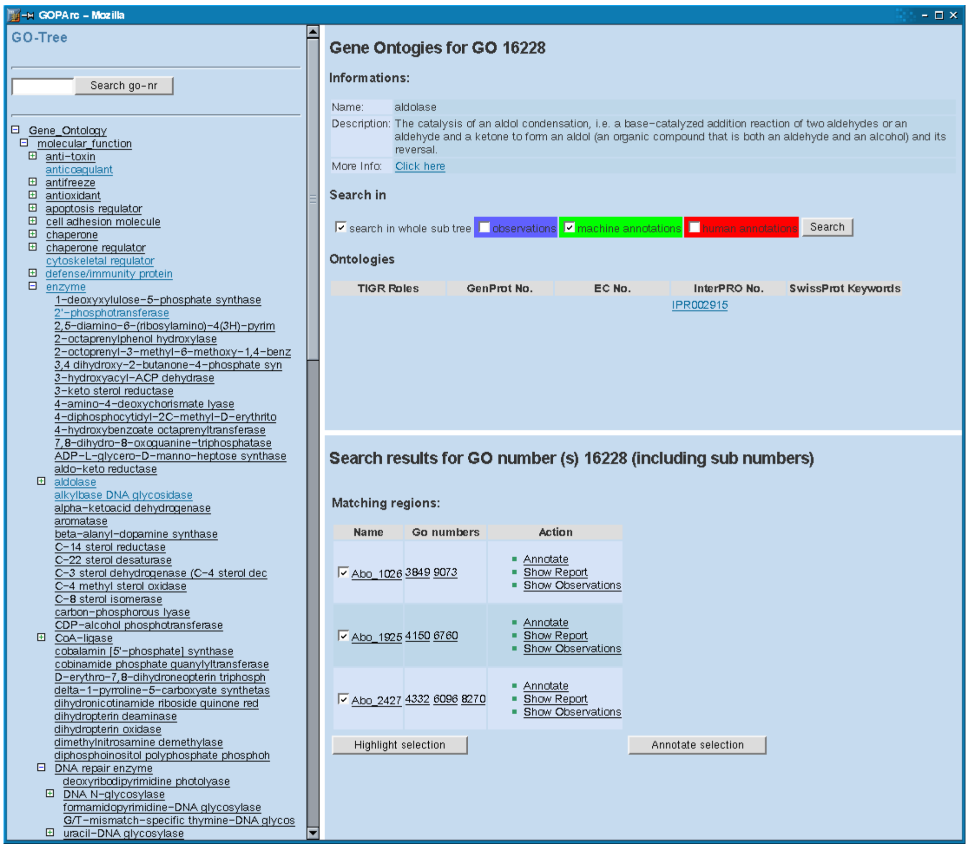
Glyoxylate and dicarboxylate metabolism
Histidine metabolism
Huntington's disease
Inositol metabolism
Inositol phosphate metabolism
Keratan sulfate biosynthesis
Lipid Metabolism
Lipopolysaccharide biosynthesis
Lysine biosynthesis
Lysine degradation
Metabolism
Metabolism of Cofactors and Vitamins
Metabolism of Complex Carbohydrates
Metabolism of Complex Lipids
Metabolism of Other Amino Acids
Methane metabolism
Methionine metabolism
N–Glycan degradation
N–Glycans biosynthesis
Nicotinate and nicotinamide metabolism
Nitrobenzene degradation
Nitrogen metabolism
Nucleotide Metabolism
Nucleotide sugars metabolism
O–Glycans biosynthesis
One carbon pool by folate
Oxidative phosphorylation
Pantothenate and CoA biosynthesis
Parkinson's disease
Penicillins and cephalosporins biosynthesis
Pentose and glucuronate interconversions
Pentose phosphate pathway
Peptidoglycan biosynthesis
Phenylalanine metabolism
Phenylalanine, tyrosine and tryptophan biosynthesis
Phosphatidylinositol signaling system
Phospholipid degradation
Photosynthesis

Selected EC number: 2.6.1.17

View ENZYME entry
Found in observation:

Abo_881	–>Observation
Abo_761	–>Observation
Abo_479	–>Observation
Abo_1073	–>Observation
Abo_329	–>Observation
Abo_1115	–>Observation

Figure 7.15.: The KEGG viewer for navigating the KEGG metabolic pathways has been integrated into the GenDB-2.0 web frontend. Different colors highlight automatic/manual annotated enzymes and those that can be found in the observations.



GOPArc – Mozilla

GO-Tree

Search go-nr

- Gene_Ontology
  - molecular_function
    - anti-toxin
    - anticoagulant
    - antifreeze
    - antioxidant
    - apoptosis regulator
    - cell adhesion molecule
    - chaperone
    - chaperone regulator
    - cytoskeletal regulator
    - defense/immunity protein
    - enzyme
      - 1-deoxyxylulose-5-phosphate synthase
      - 2'-phosphotransferase
      - 2,5-diamino-6-(ribosylamino)-4(3H)-pyrim
      - 2-octaprenylphenol hydroxylase
      - 2-octoprenyl-3-methyl-6-methoxy-1,4-benz
      - 3,4 dihydroxy-2-butanone-4-phosphate syn
      - 3-hydroxyacyl-ACP dehydrase
      - 3-keto sterol reductase
      - 4-amino-4-deoxychorismate lyase
      - 4-diphosphocytidyl-2C-methyl-D-erythrito
      - 4-hydroxybenzoate octaprenyltransferase
      - 7,8-dihydro-8-oxoguanine-triphosphatase
      - ADP-L-glycero-D-manno-heptose synthase
      - aldo-keto reductase
      - aldolase
      - alkylbase DNA glycosidase
      - alpha-ketoacid dehydrogenase
      - aromatase
      - beta-alanyl-dopamine synthase
      - C-14 sterol reductase
      - C-22 sterol desaturase
      - C-3 sterol dehydrogenase (C-4 sterol dec
      - C-4 methyl sterol oxidase
      - C-8 sterol isomerase
      - carbon-phosphorous lyase
      - CDP-alcohol phosphotransferase
      - CoA-ligase
      - cobalamin [5'-phosphate] synthase
      - cobinamide phosphate guanylyltransferase
      - D-erythro-7,8-dihydroneopterin triphosph
      - delta-1-pyrroline-5-carboxyate synthetas
      - dihydronicotinamide riboside quinone red
      - dihydropterin deaminase
      - dihydropterin oxidase
      - dimethylnitrosamine demethylase
      - diphosphoinositol polyphosphate phosphoh
      - DNA repair enzyme
        - deoxyribodipyrimidine photolyase
        - DNA N-glycosylase
        - formamidopyrimidine-DNA glycosylase
        - G/T-mismatch-specific thymine-DNA glycos
        - uracil-DNA glycosylase

Gene Ontogies for GO 16228

Informations:

Name:	aldolase
Description:	The catalysis of an aldol condensation, i.e. a base-catalyzed addition reaction of two aldehydes or an aldehyde and a ketone to form an aldol (an organic compound that is both an aldehyde and an alcohol) and its reversal.
More Info:	Click here

Search in

[✓] search in whole sub tree [ ] observations [✓] machine annotations [ ] human annotations Search

Ontologies

TIGR Roles	GenProt No.	EC No.	InterPRO No.	SwissProt Keywords
			IPR002915	

Search results for GO number (s) 16228 (including sub numbers)

Matching regions:

Name	Go numbers	Action
✓ Abo_1026	3849 9073	Annotate Show Report Show Observations
✓ Abo_1925	4150 6760	Annotate Show Report Show Observations
✓ Abo_2427	4332 6096 8270	Annotate Show Report Show Observations

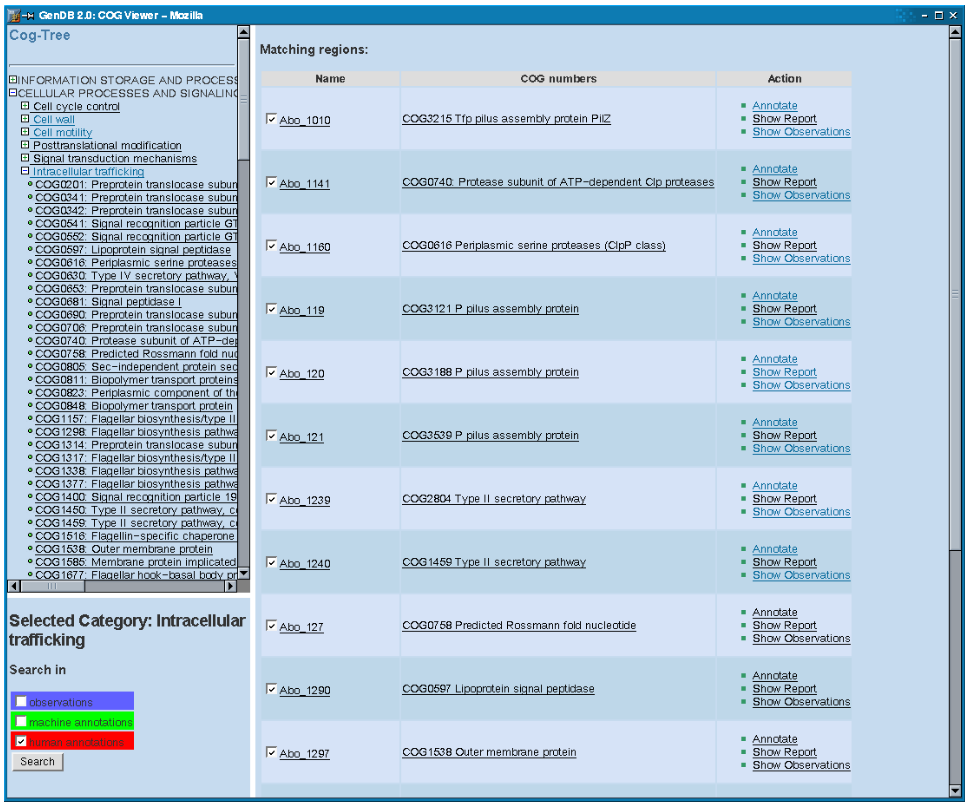
Highlight selection   Annotate selection

Figure 7.16.: The Gene Ontology browser of the web frontend in GenDB-2.0. The user can search for all regions that belong to a GO category and select them for visualization or annotation.

The GenDB-2.0 web frontend directly integrates three additional navigation metaphors of the GOPArc system (see section 7.3) for browsing or annotating a genome based on the KEGG metabolic pathways, Gene Ontologies (GO), or Clusters of Orthologous Genes (COG). The KEGG browser (see figure 7.15) can be used to visualize a metabolic pathway of the KEGG database with those enzymes that were found in an observation (blue), automatically annotated (green), or manually annotated (red). The user can then navigate to a selected set of genes for these enzymes and also annotate them. Each enzyme is linked to the ENZYME database and the COMPOUND entries can be accessed by clicking on the chemical substrates.

As illustrated in figure 7.16, the Gene Ontologies can be used to search for regions that were annotated (automatically: green, manually: red) with a GO number. It is also possible to search in the observations (blue) for likely candidate genes that were annotated with a selected GO category. Identified regions can then be displayed or annotated.

The same navigation has also been implemented in the COG viewer (see figure 7.17) for browsing the Clusters of Orthologous Genes (COG).


GenDB 2.0: COG Viewer – Mozilla

Cog-Tree

- ⊞INFORMATION STORAGE AND PROCESS
- ⊟CELLULAR PROCESSES AND SIGNALINC
  - ⊞ Cell cycle control
  - ⊞ Cell wall
  - ⊞ Cell motility
  - ⊞ Posttranslational modification
  - ⊞ Signal transduction mechanisms
  - ⊟ Intracellular trafficking
    - COG0201: Preprotein translocase subun
    - COG0341: Preprotein translocase subun
    - COG0342: Preprotein translocase subun
    - COG0541: Signal recognition particle GT
    - COG0552: Signal recognition particle GT
    - COG0597: Lipoprotein signal peptidase
    - COG0616: Periplasmic serine proteases
    - COG0630: Type IV secretory pathway, V
    - COG0653: Preprotein translocase subun
    - COG0681: Signal peptidase I
    - COG0690: Preprotein translocase subun
    - COG0706: Preprotein translocase subun
    - COG0740: Protease subunit of ATP–dep
    - COG0758: Predicted Rossmann fold nuc
    - COG0805: Sec–independent protein sec
    - COG0811: Biopolymer transport proteins
    - COG0823: Periplasmic component of th
    - COG0848: Biopolymer transport protein
    - COG1157: Flagellar biosynthesis/type II
    - COG1298: Flagellar biosynthesis pathwa
    - COG1314: Preprotein translocase subun
    - COG1317: Flagellar biosynthesis/type II
    - COG1338: Flagellar biosynthesis pathwa
    - COG1377: Flagellar biosynthesis pathwa
    - COG1400: Signal recognition particle 19
    - COG1450: Type II secretory pathway, c
    - COG1459: Type II secretory pathway, c
    - COG1516: Flagellin–specific chaperone
    - COG1538: Outer membrane protein
    - COG1585: Membrane protein implicated
    - COG1677: Flagellar hook–basal body pr

Selected Category: Intracellular trafficking

Search in

- ☐ observations
- ☐ machine annotations
- ☑ human annotations

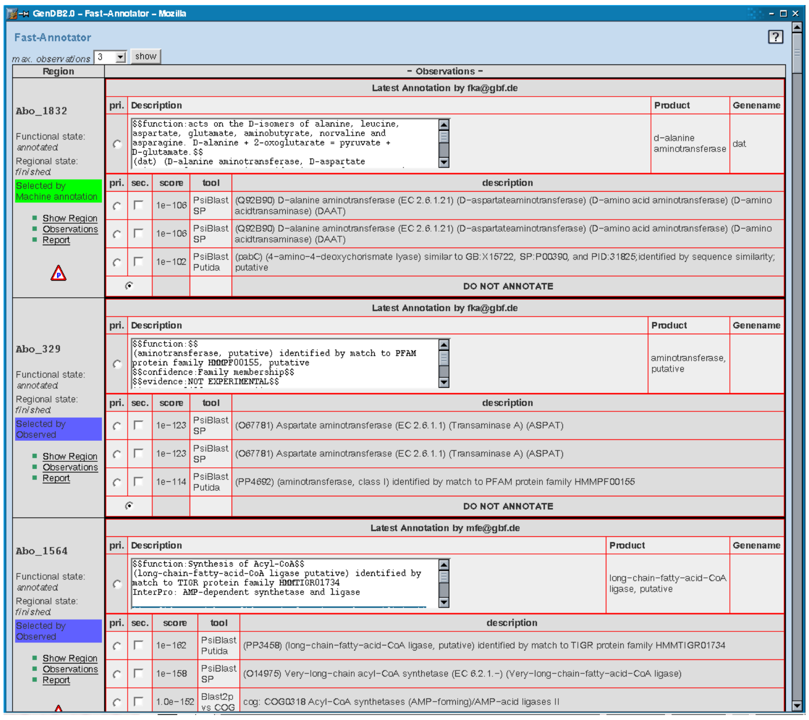
Search

Matching regions:

Name	COG numbers	Action
☑ Abo_1010	COG3215 Tfp pilus assembly protein PilZ	Annotate, Show Report, Show Observations
☑ Abo_1141	COG0740: Protease subunit of ATP–dependent Clp proteases	Annotate, Show Report, Show Observations
☑ Abo_1160	COG0616 Periplasmic serine proteases (ClpP class)	Annotate, Show Report, Show Observations
☑ Abo_119	COG3121 P pilus assembly protein	Annotate, Show Report, Show Observations
☑ Abo_120	COG3188 P pilus assembly protein	Annotate, Show Report, Show Observations
☑ Abo_121	COG3539 P pilus assembly protein	Annotate, Show Report, Show Observations
☑ Abo_1239	COG2804 Type II secretory pathway	Annotate, Show Report, Show Observations
☑ Abo_1240	COG1459 Type II secretory pathway	Annotate, Show Report, Show Observations
☑ Abo_127	COG0758 Predicted Rossmann fold nucleotide	Annotate, Show Report, Show Observations
☑ Abo_1290	COG0597 Lipoprotein signal peptidase	Annotate, Show Report, Show Observations
☑ Abo_1297	COG1538 Outer membrane protein	Annotate, Show Report, Show Observations

Figure 7.17.: The COG viewer of the GenDB-2.0 web frontend.

For each region that was found by searching the genome via one of the navigation metaphors, different actions are provided: the region can be annotated, displayed in the report, or the observations for a selected region can be listed. It is also possible to select several regions by activating the corresponding checkboxes. These are then highlighted in the quick navigation bar of the main window. All checked regions can be annotated simultaneously using the *Multiple Annotator* that is displayed in figure 7.18. This interface can be used to annotate several genes consistently in a "one-click" manner. The user can select the number of observations that are displayed for each region. In the next step, a single observation can be chosen that will be used to annotate the region (e.g. the system will try to extract a gene name, gene product, EC number, and a description from the selected observation). Other observations from the list can be added to the annotation as supporting evidence. After submitting the selected regions for annotation, the remaining regions (if any) will appear again for further analysis and final annotation. At the left side of the window the user can also find some information about the status of each region and in some cases a warning will be displayed that indicates potential paralogous genes.



Abo_1832

Functional state: *annotated*
Regional state: *finished*
Selected by Machine annotation

- Show Region
- Observations
- Report

Latest Annotation by fka@gbf.de

pri.	Description	Product	Genename
	$$function:acts on the D-isomers of alanine, leucine, aspartate, glutamate, aminobutyrate, norvaline and asparagine. D-alanine + 2-oxoglutarate = pyruvate + D-glutamate.$$ (dat) (D-alanine aminotransferase, D-aspartate	d-alanine aminotransferase	dat

pri.	sec.	score	tool	description
		1e-106	PsiBlast SP	(Q92B90) D-alanine aminotransferase (EC 2.6.1.21) (D-aspartateaminotransferase) (D-amino acid aminotransferase) (D-amino acidtransaminase) (DAAT)
		1e-106	PsiBlast SP	(Q92B90) D-alanine aminotransferase (EC 2.6.1.21) (D-aspartateaminotransferase) (D-amino acid aminotransferase) (D-amino acidtransaminase) (DAAT)
		1e-102	PsiBlast Putida	(pabC) (4-amino-4-deoxychorismate lyase) similar to GB:X15722, SP:P00390, and PID:31825;identified by sequence similarity; putative
●				DO NOT ANNOTATE

Abo_329

Functional state: *annotated*
Regional state: *finished*
Selected by Observed

- Show Region
- Observations
- Report

Latest Annotation by fka@gbf.de

pri.	Description	Product	Genename
	$$function:$$ (aminotransferase, putative) identified by match to PFAM protein family HMMPF00155, putative $$confidence:Family membership$$ $$evidence:NOT EXPERIMENTAL$$	aminotransferase, putative	

pri.	sec.	score	tool	description
		1e-123	PsiBlast SP	(O67781) Aspartate aminotransferase (EC 2.6.1.1) (Transaminase A) (ASPAT)
		1e-123	PsiBlast SP	(O67781) Aspartate aminotransferase (EC 2.6.1.1) (Transaminase A) (ASPAT)
		1e-114	PsiBlast Putida	(PP4692) (aminotransferase, class I) identified by match to PFAM protein family HMMPF00155
●				DO NOT ANNOTATE

Abo_1564

Functional state: *annotated*
Regional state: *finished*
Selected by Observed

- Show Region
- Observations
- Report

Latest Annotation by mfe@gbf.de

pri.	Description	Product	Genename
	$$function:Synthesis of Acyl-CoA$$ (long-chain-fatty-acid-CoA ligase putative) identified by match to TIGR protein family HMMTIGR01734 InterPro: AMP-dependent synthetase and ligase	long-chain-fatty-acid-CoA ligase, putative	

pri.	sec.	score	tool	description
		1e-162	PsiBlast Putida	(PP3458) (long-chain-fatty-acid-CoA ligase, putative) identified by match to TIGR protein family HMMTIGR01734
		1e-158	PsiBlast SP	(O14975) Very-long-chain acyl-CoA synthetase (EC 6.2.1.-) (Very-long-chain-fatty-acid-CoA ligase)
		1.0e-152	Blast2p vs COG	cog: COG0318 Acyl-CoA synthetases (AMP-forming)/AMP-acid ligases II

Figure 7.18.: A multiple annotation interface has been implemented for the web frontend in GenDB-2.0 that can be used to annotate a group of genes simultaneously.

7.1.10. Annotation pipeline

The GenDB-2.0 system features all steps for the analysis and annotation of bacterial genomes starting from the raw contig sequence. Figure 7.19 shows an example for a genome annotation pipeline that has been implemented with GenDB. Upon import of the raw sequence data, a parent region object describing the genome sequence is created. Following this step, user-defined tools for the prediction of different kinds of regions, such as coding sequences (CDS) or tRNA-encoding genes can be run. The output of these tools is stored as observations which refer to the parent region object. Based on these observations, an annotator, human or machine, performs a "region annotation". This means confirming or rejecting the results of gene prediction tools by creating region objects like CDSs or tRNAs. The annotations form a complete protocol of all "region annotation" events. Following the creation of different kinds of regions, additional tools such as BLAST, HMMer, or CoBias can be run creating information related to their potential function. Each of these tools can have its own automatic annotator that creates a very simple annotation based solely on the results of a single tool run. After computing a number of standard tools, a more sophisticated automatic annotation can be accomplished by combining the results of different tools. Finally, a manual "function annotation" step can be performed by an annotator in which a putative function is assigned to these regions by an interpretation of the observations (see below).

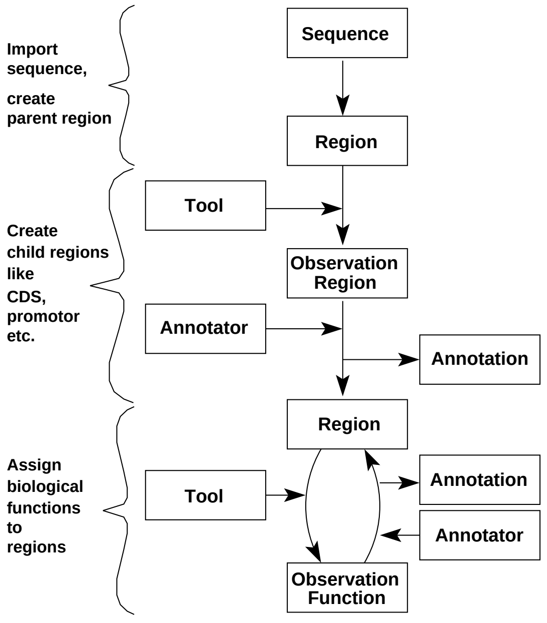
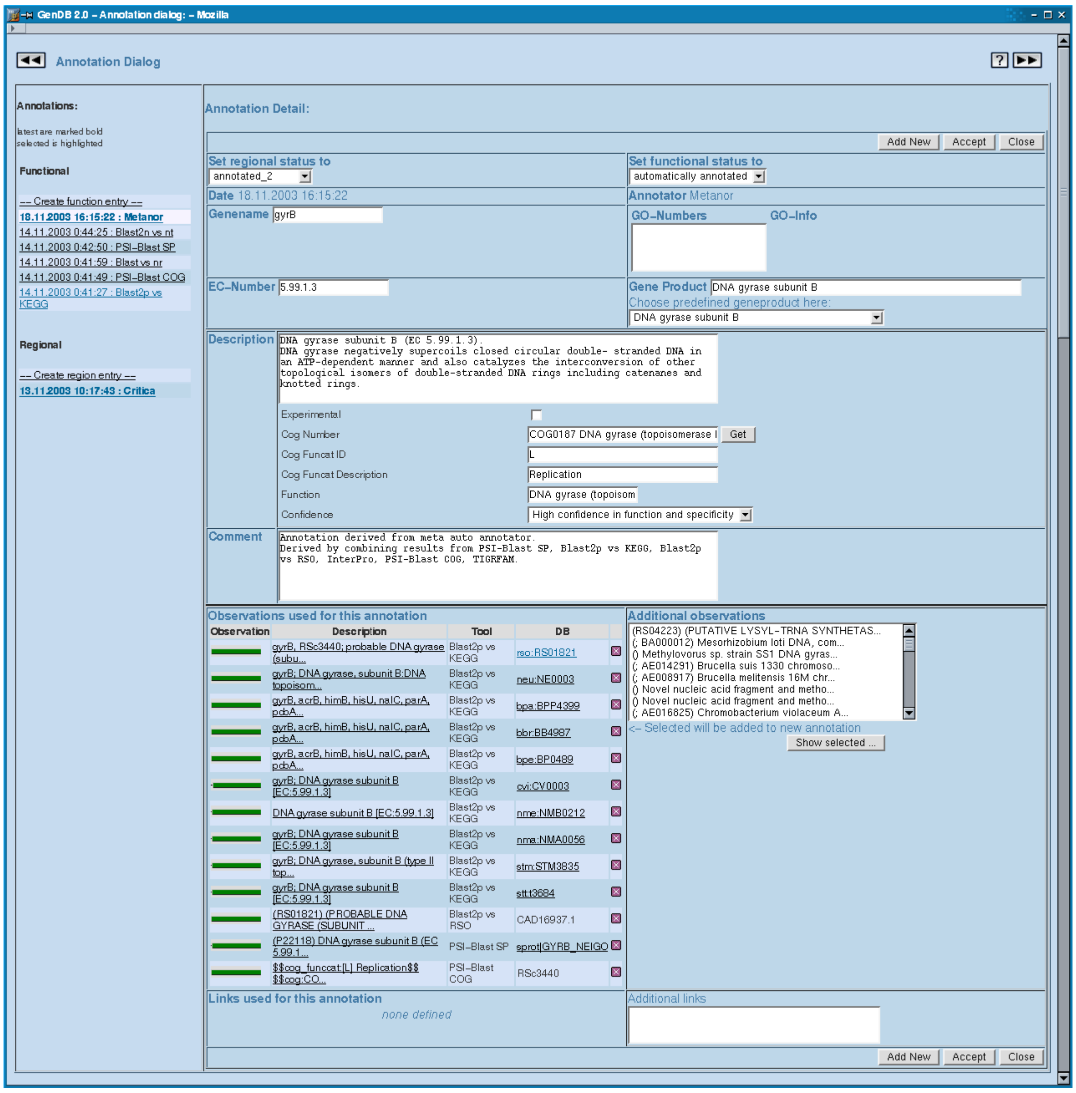


Figure 7.19.: This standard sample pipeline implemented with GenDB-2.0 starts with an import of a contig sequence. Afterwards, regions are predicted and created by a regional annotation (*Annotation::Region*). A biological function for these regions can then be assigned by computing different bioinformatics tools that often generate large numbers of observations. Based on these results an automatic or manual functional annotation (*Annotation::Function*) can be assigned.

The current Gtk version of the GenDB system features a graphical interface (*Annotation Pipeline Wizard*) for the configuration of different individual pipelines. The user can choose one or more steps (*Import, Edit Sequence, Region Prediction*, and *Function Prediction*) which are then combined to a separate pipeline. After some initial configuration, the pipeline is submitted as a special job and the corresponding steps are executed in the specified order without any further user interaction. Using these pipelines allows a very comfortable automated annotation and increases the productivity in large-scale genome annotation projects.

Nevertheless, it is still a laborious task to manually check the predicted regions and their function assignments. Both GenDB frontends therefore provide almost identical wizards for editing the start of a gene (see figure 7.6) and annotation interfaces that allow recording a comprehensive set of information about each region. Since the final manual annotation of a genome is not only the most time consuming step but also the most erroneous task, exactly defined roles for annotating a gene are essential in order to prevent inconsistent entries. The *Annotation Dialog* displayed in figure 7.20 and its available entry fields are therefore described in more detail below:

- **Header**
  The arrow buttons can be used to jump to the previous or next region.
- **Annotation List**
  The list of previously created annotations at the left side of the annotation-dialog shows the annotations for the function of a gene at the top and the annotations for the creation or modification of a region at the bottom. The latest annotation is printed in bold font, the selected annotation is highlighted.
- **Add new**
  The annotation is stored by clicking on this button.
- **Accept**
  A click on this button will automatically set the region status to finished and the function status to annotated. The button can be used for a "one-click" annotation, e.g. when the automatic annotation was almost perfect and needs no more corrections.
- **Close**
  To cancel an annotation press Close.
- **Region Status**
  The state of a region can be set by selecting a new status from the list.
- **Function Status**
  The state for the functional assignment of a region can be assigned by selecting a new status from the list.
- **Date**
  Date when the annotation has been created.
- **Annotator**
  Name of a person or tool that created the annotation.



Annotations:

latest are marked bold
selected is highlighted

Functional

-- Create function entry --
18.11.2003 16:15:22 : Metanor
14.11.2003 0:44:25 : Blast2n vs nt
14.11.2003 0:42:50 : PSI–Blast SP
14.11.2003 0:41:59 : Blast vs nr
14.11.2003 0:41:49 : PSI–Blast COG
14.11.2003 0:41:27 : Blast2p vs KEGG

Regional

-- Create region entry --
13.11.2003 10:17:43 : Critica

Annotation Detail:

Set regional status to: annotated_2

Set functional status to: automatically annotated

Date 18.11.2003 16:15:22

Annotator Metanor

Genename gyrB

GO–Numbers

GO–Info

EC–Number 5.99.1.3

Gene Product DNA gyrase subunit B

Choose predefined geneproduct here: DNA gyrase subunit B

Description

```
DNA gyrase subunit B (EC 5.99.1.3).
DNA gyrase negatively supercoils closed circular double- stranded DNA in
an ATP-dependent manner and also catalyzes the interconversion of other
topological isomers of double-stranded DNA rings including catenanes and
knotted rings.
```

Experimental	☐
Cog Number	COG0187 DNA gyrase (topoisomerase I
Cog Funcat ID	L
Cog Funcat Description	Replication
Function	DNA gyrase (topoisom
Confidence	High confidence in function and specificity

Comment

```
Annotation derived from meta auto annotator.
Derived by combining results from PSI-Blast SP, Blast2p vs KEGG, Blast2p
vs RSO, InterPro, PSI-Blast COG, TIGRFAM.
```

Observations used for this annotation

Observation	Description	Tool	DB	
🟩	gyrB, RSc3440; probable DNA gyrase (subu...	Blast2p vs KEGG	rso:RS01821	☒
🟩	gyrB; DNA gyrase, subunit B:DNA topoisom...	Blast2p vs KEGG	neu:NE0003	☒
🟩	gyrB, acrB, himB, hisU, nalC, parA, pcbA...	Blast2p vs KEGG	bpa:BPP4399	☒
🟩	gyrB, acrB, himB, hisU, nalC, parA, pcbA...	Blast2p vs KEGG	bbr:BB4987	☒
🟩	gyrB, acrB, himB, hisU, nalC, parA, pcbA...	Blast2p vs KEGG	bpe:BP0489	☒
🟩	gyrB; DNA gyrase subunit B [EC:5.99.1.3]	Blast2p vs KEGG	cvi:CV0003	☒
🟩	DNA gyrase subunit B [EC:5.99.1.3]	Blast2p vs KEGG	nme:NMB0212	☒
🟩	gyrB; DNA gyrase subunit B [EC:5.99.1.3]	Blast2p vs KEGG	nma:NMA0056	☒
🟩	gyrB; DNA gyrase, subunit B (type II top...	Blast2p vs KEGG	stm:STM3835	☒
🟩	gyrB; DNA gyrase subunit B [EC:5.99.1.3]	Blast2p vs KEGG	stt:t3684	☒
🟩	(RS01821) (PROBABLE DNA GYRASE (SUBUNIT ...	Blast2p vs RSO	CAD16937.1	☒
🟩	(P22118) DNA gyrase subunit B (EC 5.99.1...	PSI–Blast SP	sprot|GYRB_NEIGO	☒
🟩	$$cog_funccat:[L] Replication$$ $$cog:CO...	PSI–Blast COG	RSc3440	☒

Additional observations

(RS04223) (PUTATIVE LYSYL–TRNA SYNTHETAS...
(; BA000012) Mesorhizobium loti DNA, com...
() Methylovorus sp. strain SS1 DNA gyras...
(; AE014291) Brucella suis 1330 chromoso...
(; AE008917) Brucella melitensis 16M chr...
() Novel nucleic acid fragment and metho...
() Novel nucleic acid fragment and metho...
(; AE016825) Chromobacterium violaceum A...

<– Selected will be added to new annotation

Links used for this annotation

none defined

Additional links

Figure 7.20.: This screenshot of the annotation dialog of the GenDB-2.0 web frontend shows the latest automatic annotation. On the left side, the user can see the history of already existing annotations. For annotating a gene, a number of fields have to be filled out and a status can be set. All genes can be classified using the automatically assigned COG categories and Gene Ontology numbers. As a special feature of GenDB-2.0, observations can be stored with an annotation as supporting evidence.

- **Gene Name**
  A gene name should only be assigned to regions with high quality evidence from the observations. Usually, gene names are have four characters with a capital last letter, e.g. *dnaA*.
- **GO-Numbers**
  The list of GO-Numbers displays Gene Ontology numbers that were assigned to a region. Detailed information about each number can be obtained by clicking on the number itself in the GO-Info list at the right.
- **EC-Number**
  An EC-Number (Enzyme Commission) can be assigned in this field for enzymes, e.g. 1.2.3.4.
- **Gene Product**
  The gene product of a coding sequence (CDS) can be entered here as detailed as possible. If an annotator is unsure about a gene product, it is also possible to select one of the default gene products from the list of predefined gene products below.
- **Description**
  The description field of an annotation contains a number of fields that should be set:
  - **Plain Text**
    This is the place where annotators should add a detailed description of the CDS. The function of the gene should be explained and how this has been derived, e.g. identified by sequence similarity. Alternative gene names or gene products that may be important can also be added here. All information that is collected here will be exported into the EMBL file for submission to a public database. Thus any unnecessary details should be written as a comment (see below).
  - **Experimental**
    This check-box can be used to indicate experimental evidence for the annotation, e.g. wet lab experiments. During export, this value becomes the evidence field in the EMBL file.
  - **COG Number**
    The COG Number field can be used to assign a functional classification from the Clusters of Orthologous Genes. In most cases, the COG category is added automatically and should not be changed. Another category can be chosen by pressing on the “Get” button and selecting a category from the tree view that appears in a new window.
  - **COG Funccat ID**
    This field contains the one letter code assigned to the main category of the selected COG Number.

  - **COG Funccat Description**
    This field contains the description for the main COG category.
  - **Function**
    Other functional category terms (e.g. from TIGR roles, Monica Riley categories, or GO) can be entered here if there is no exactly matching term in the COG list. In most cases this will be the same function as the COG assignment. The function assigned in this field will be exported into the EMBL file as a function entry. Thus an annotator should always try to assign a function.
  - **Confidence**
    The confidence field can be used to describe the confidence that you have in your annotation. A level can be selected from the list.

- **Comment**
  This is the right place to add everything else that annotators have in mind about a CDS. Simple notes that might be useful for an annotator or others can be stored in this field. Everything that should **not** appear in the final annotation submitted as an EMBL file can be written here; the content of the comment field will not be exported.

- **Observations**
  The list of observations contains references to observations (e.g. BLAST results) that have been used to derive an annotation. These are used as supporting evidence and by clicking on one of them the observation will be redisplayed in the Observation-List. Observations can be deleted from this list by clicking on the small button next to each observation in the list when an observation does not support the annotation. Other additional observations can be added by selecting them from the list at the right.

- **Links**
  Supplementary hyper-links can be added to an annotation by entering them in the list at the right, e.g. a URL to medline.

7.2. EMMA

Since none of the existing systems fulfilled all requirements stated in section 3.2, the EMMA system [DGB$^+$03] was developed as an open source platform for the storage and efficient analysis of microarray data.

The EMMA software can be used to store all kinds of transcriptome data. Experimental setups, slide layouts, information about spot contents (e.g. EST libraries), and the measured spot intensities are stored in a relational database and can be accessed using an O2DBI API or a graphical interface (web or Gtk frontend). In addition to image analysis and editing facilities, the system also incorporates a number of algorithms for the normalization (e.g. print-tip and lowess regression) and analysis of expression data (simple statistics, t-test, k-means clustering, self organizing maps). The results of these analyses (e.g. tables of differentially expressed genes) can be linked directly to regions in a GenDB database.

7.2.1. Design overview

The EMMA platform has been designed to cover all aspects of microarray data acquisition and analysis. The general structure of a microarray experiment encourages using a modular concept for the implementation of separate specialized components. Figure 7.21 displays the main tasks that are involved in microarray experiments and how they are realized in the EMMA platform. All required components of the system are open source and available for standard UNIX/Linux systems.

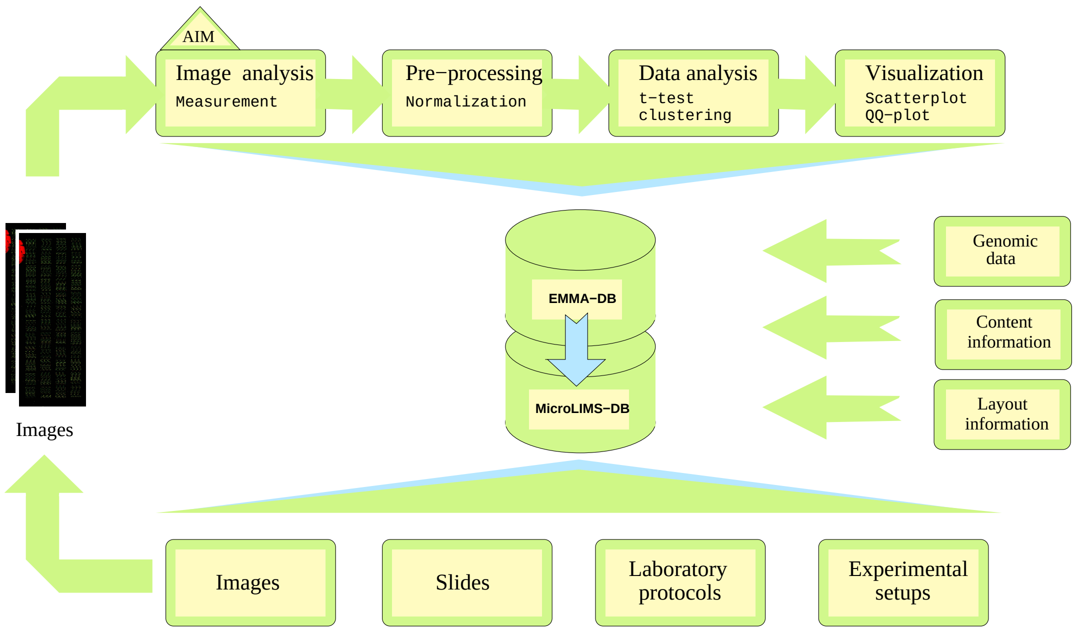


Figure 7.21.: The design of the EMMA-platform: the platform consists of the two databases EMMA-DB and MicroLIMS-DB as central data repositories which store data from many sources. Data concerning experimental design and protocols can be uploaded and filed with MicroLIMS at any time they arise in the lab (lower part of the figure). Highly customizable pipelines are provided by EMMA for the subsequent analysis of measured data (top part). Scanned images can be automatically loaded from MicroLIMS and inserted into such a pipeline.

The system is based on a central data repository (consisting of MicroLIMS and EMMA-DB) for storing all experiment related (laboratory) information and those data sources that are intimately connected with the measured data and results obtained from further analysis. Experimental setups, parameters, employed methods or procedures, and the images obtained by scanning a microarray are described in the MicroLIMS system and can be referenced by experiments stored in the EMMA database. The details about the major components of the EMMA platform are described in the following sections.

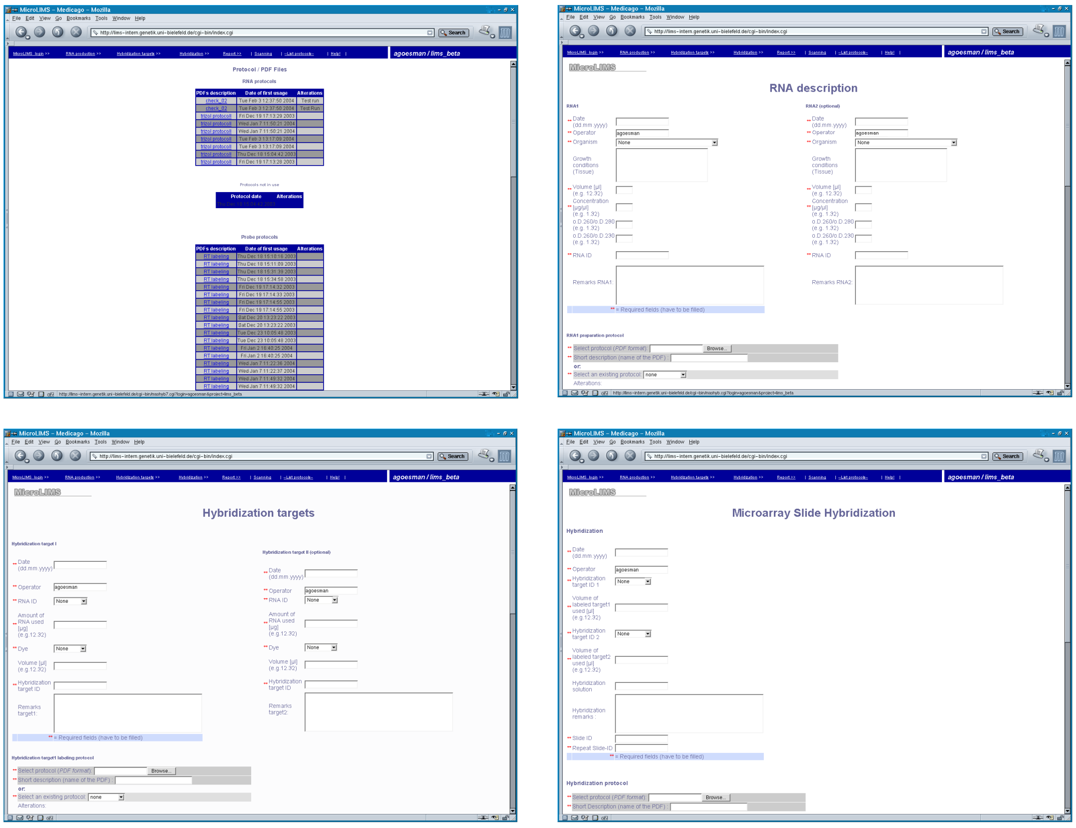
7.2.2. MicroLIMS

MicroLIMS has been developed as an integral building block of the EMMA-platform. It relies on a separate database and can be used independently. As a LIMS server it can manage all kinds of laboratory specific meta data that is acquired during a microarray experiment. MicroLIMS captures the whole laboratory work flow of RNA-purification, probe labeling, hybridization, and scanning protocols (see Fig. 7.22). It also allows uploading and downloading of images from the microarray scanner and of raw data files from image analysis software. MicroLIMS has a web-based user interface which provides data upload via a standard web browser installed on a laboratory PC. The protocol data stored in MicroLIMS and the measured data from each slide are linked via the EMMA platform in such a way that a report of the experimental setup can be displayed for each slide.

7.2.3. EMMA-DB

The EMMA-DB component represents the core module of the EMMA platform. All measured and computed data derived from the analysis of microarray experiments are stored in a relational database (e.g. MySQL or PostgreSQL). The database schema was specified in UML compliant to the MIAME (Minimum Information About a Microarray Experiment) recommendations [MGEa] and implemented with the O2DBI-II tool.

Together with user defined extensions and custom class methods, the automatically generated modules form an object-oriented API layer (see figure 7.23) that handles all access to the data stored in the database.


MicroLIMS – Medicago – Mozilla
File Edit View Go Bookmarks Tools Window Help
Search
MicroLIMS login >> | RNA production >> | Hybridization targets >> | Hybridization >> | Report >> | Scanning | --List protocols-- | Help! |
agoesman / lims_beta
MicroLIMS
Browse...

http://lims-intern.genetik.uni-bielefeld.de/cgi-bin/index.cgi

Protocol / PDF Files

RNA protocols

PDFs description	Date of first usage	Alterations
check_02	Tue Feb 3 12:37:50 2004	Test run
check_02	Tue Feb 3 12:37:50 2004	Test Run
trizol protocoll	Fri Dec 19 17:13:29 2003	
trizol protocoll	Wed Jan 7 11:50:21 2004	
trizol protocoll	Wed Jan 7 11:50:21 2004	
trizol protocoll	Tue Feb 3 13:17:09 2004	
trizol protocoll	Tue Feb 3 13:17:09 2004	
trizol protocoll	Thu Dec 18 15:04:42 2003	
trizol protocoll	Fri Dec 19 17:13:28 2003	

Protocols not in use

Protocol date	Alterations
Thu Dec 18 15:04:42 2003	

Probe protocols

PDFs description	Date of first usage	Alterations
RT labeling	Thu Dec 18 15:10:16 2003	
RT labeling	Thu Dec 18 15:11:09 2003	
RT labeling	Thu Dec 18 15:31:39 2003	
RT labeling	Thu Dec 18 15:34:58 2003	
RT labeling	Fri Dec 19 17:14:32 2003	
RT labeling	Fri Dec 19 17:14:33 2003	
RT labeling	Fri Dec 19 17:14:55 2003	
RT labeling	Fri Dec 19 17:14:55 2003	
RT labeling	Sat Dec 20 13:23:22 2003	
RT labeling	Sat Dec 20 13:23:22 2003	
RT labeling	Tue Dec 23 10:05:48 2003	
RT labeling	Tue Dec 23 10:05:48 2003	
RT labeling	Fri Jan 2 16:40:25 2004	
RT labeling	Fri Jan 2 16:40:25 2004	
RT labeling	Wed Jan 7 11:22:36 2004	
RT labeling	Wed Jan 7 11:22:37 2004	
RT labeling	Wed Jan 7 11:49:32 2004	
RT labeling	Wed Jan 7 11:49:32 2004	

http://lims-intern.genetik.uni-bielefeld.de/cgi-bin/mashyb7.cgi?login=agoesman&project=lims_beta

RNA description

RNA1

** Date (dd.mm.yyyy)	
** Operator	agoesman
** Organism	None
Growth conditions (Tissue)	
** Volume [µl] (e.g. 12.32)	
** Concentration [µg/µl] (e.g. 1.32)	
o.D.260/o.D.280 (e.g. 1.32)	
o.D.260/o.D.230 (e.g. 1.32)	
** RNA ID	
Remarks RNA1:	

RNA2 (optional)

** Date (dd.mm.yyyy)	
** Operator	agoesman
** Organism	None
Growth conditions (Tissue)	
** Volume [µl] (e.g. 12.32)	
** Concentration [µg/µl] (e.g. 1.32)	
o.D.260/o.D.280 (e.g. 1.32)	
o.D.260/o.D.230 (e.g. 1.32)	
** RNA ID	
Remarks RNA2:	

** = Required fields (have to be filled)

RNA1 preparation protocol

** Select protocol (*PDF format*):
** Short description (name of the PDF) :
or:
** Select an existing protocol: none
Alterations:

http://lims-intern.genetik.uni-bielefeld.de/cgi-bin/mashyb.cgi?login=agoesman&project=lims_beta

Hybridization targets

Hybridization target I

** Date (dd.mm.yyyy)	
** Operator	agoesman
** RNA ID	None
** Amount of RNA used [µg] (e.g.12.32)	
** Dye	None
Volume [µl] (e.g.12.32)	
** Hybridization target ID	
Remarks target1:	

Hybridization target II (optional)

** Date (dd.mm.yyyy)	
** Operator	agoesman
** RNA ID	None
** Amount of RNA used [µg] (e.g.12.32)	
** Dye	None
Volume [µl] (e.g.12.32)	
** Hybridization target ID	
Remarks target2:	

** = Required fields (have to be filled)

Hybridization target1 labeling protocol

** Select protocol (*PDF format*):
** Short description (name of the PDF) :
or:
** Select an existing protocol: none
Alterations:

Microarray Slide Hybridization

Hybridization

** Date (dd.mm.yyyy)	
** Operator	agoesman
** Hybridization target ID 1	None
** Volume of labeled target1 used [µl] (e.g.12.32)	
** Hybridization target ID 2	None
** Volume of labeled target2 used [µl] (e.g.12.32)	
Hybridization solution	
Hybridization remarks :	
** Slide ID	
** Repeat Slide-ID	

** = Required fields (have to be filled)

Hybridization protocol

** Select protocol (*PDF format*):
** Short Description (name of the PDF) :

Figure 7.22.: The MicroLIMS web interface. The user can upload protocol specific data and images at every stage of a microarray experiment: RNA purification, labeling of the probes, hybridization, and scanning.

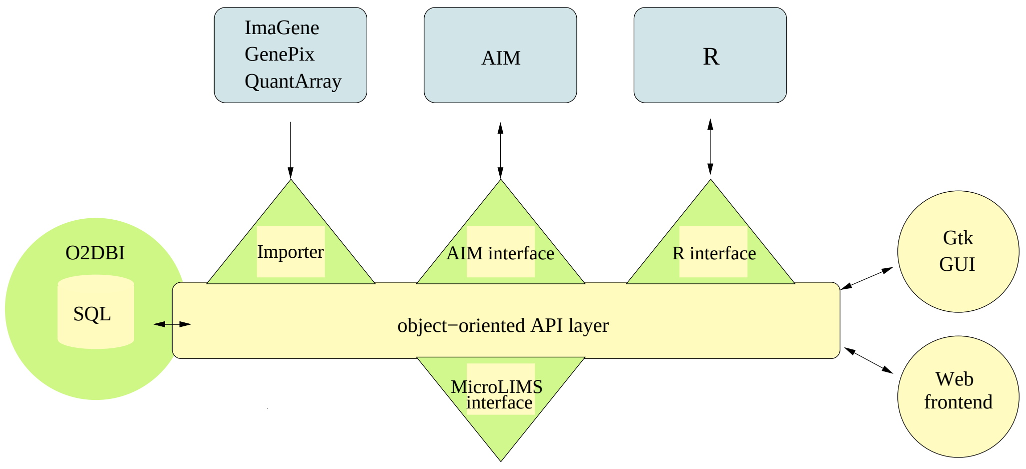


Figure 7.23.: Overview of the architecture of the EMMA-DB core components: the central object-oriented API layer serves as an bidirectional interface for the modules, which provide the functionality of the EMMA platform (e.g. web frontend, GUI and import wizards). The API layer relies on O2DBI-II which implements an abstraction layer to a relational database on the left. Additional components like the MicroLIMS system or the R packages are connected to the main layer via specialized communication interfaces plotted as triangles, while the external applications are depicted by grey rectangles. Circles denote user interfaces and arrows represent the directionality of communication (uni- or bidirectional) between components.

The EMMA-DB core has been implemented in Perl but since O2DBI-II features the implementation of a client server architecture, all data can also be accessed via a C++ client or an XML-based remote procedure call (XML-RPC) interface which allows for the communication of arbitrary O2DBI-II applications via the Internet. EMMA uses MySQL by default but can be configured to use another DBMS if required.

7.2.4. Data import/export

EMMA is capable of importing a variety of data files in different formats from other software. EMMA can currently import EMBL and FASTA files and is compatible to all widely used image analysis tools as well as general spreadsheet files.

7.2.5. Image analysis

There are already some systems for analyzing images from laser scanners. We have completely embedded the AIM software [KKSed] into the analysis pipeline. AIM is capable of automatic spot detection with little or no user interaction. You can import the scanned images uploaded to MicroLIMS, start automatic spot detection and adjust spot and grid positions from within the user interface. After adjusting spots and grids, intensity values can be recomputed automatically.

The output of other software installed on a stand-alone PC can also be imported. EMMA currently supports output files from ImaGene, GenePix, and QuantArray. Other formats can be added as they are available. All computation in the EMMA-platform is carried out using the open source statistics system R [IG96]. R already provides hundreds of efficient operations essential for statistic analysis and visualization.

7.2.6. Filtering, normalization, and calibration

Data from microarray experiments often suffers from a variety of systematic deviations like different hybridization conditions, dye efficiencies, scanner settings, and bleaching effects. These effects can distort the distribution of the data. To illustrate this, a dye-shift experiment was carried out using microarrays with 3568 ORF-specific DNA fragments of *Corynebacterium glutamicum* [HBB+03]. In this experiment, RNA from one condition is labeled with both dyes and hybridized to a microarray. In theory, one would expect the data to exactly follow the main diagonal, but often the data distribution is found to be skewed as a result of systematic effects.

To remove spots that mainly contribute to experimental noise, filtering of the data is the first step in the analysis pipeline. EMMA provides filters based on spot intensities and standard deviation specifying either threshold values or percentiles. Removing the spots belonging to the lowest x percent of the overall intensity is a widely used method.

The R-package “sma”[4] provides different methods for normalization including median signal normalization and normalization based on local regression [YDLSb]. Normalization can be computed for the whole set of spots or the user can define an arbitrary set of spots as a reference.

All available methods of the sma package compute two values for each spot: The log-ratio $M = \log_2(R/G) - c(A)$ and the log-overall intensity $A = \log_2(\sqrt{RG})$, where R and G denote intensities of the red and green channels, respectively, and $c(A)$ denotes the intensity dependent normalization function.

[4] *http://cran.r-project.org/*

Variance stabilization is an alternative to applying logarithmic transformation by computing *M* and *A*-values. Application of variance stabilization calibrates the measured data such that the variance of the derived dataset is constant over the whole range of intensities. Therefore, differential expression of highly expressed genes can be directly compared to genes showing low levels of expression. [HvHS$^+$02]

7.2.7. Testing for differentially expressed genes

One main task in analyzing microarray experiments is to find significantly up or down regulated genes. cDNA microarrays cannot measure gene expression directly but indirectly by relative transcript abundance. Abundance could also be due to technical or other biological reasons (e.g. transcript stability), but for the sake of shortness we will call genes found to have a high ratio of transcript abundance "differentially expressed".

The simplest approach to find such genes is to define an *n*-fold threshold of the calculated mean *M*-values over the replicates. The *n*-fold-approach can be improved by taking the variability of the data into account. We have decided to use Student's t-test provided by R to assess the significance of differential expression. This test can be used to compare one or two groups of replicate microarrays. The grouping of replicate arrays increases the number of replicates for each gene and thereby the reliability of the test. By default, unequal variances between the samples are assumed in the two sample case and the Welch modification to the degrees of freedom is used.

The t-test assigns a value t to each gene which can be used to rank differentially expressed genes. Additionally, a confidence indicator p is computed for each gene. The t-test assumes that the samples are normally distributed. Thus we have recently added the Wilcoxon rank sum statistic in case normal distribution is not guaranteed.

There are as well different methods for computing adjusted p-values like Bonferroni's, Holm's, and Hochberg's methods as well as the method of [WY93] (see also [DYCS02]).

As a result of the test, EMMA displays a comprehensive list of genes containing some statistics for each gene which can be exported. Another option for visualization of the results of the t-test is to output a quantile-quantile plot of the t-values as used by [DYCS02].

7.2.8. Cluster analysis

Often microarray experiments are designed as multi conditional experiments like observing changes of expression over multiple points in time or under different doses of a drug. A well known example is found in [SSZ$^+$98]. These experiments result in expression profiles for each gene.

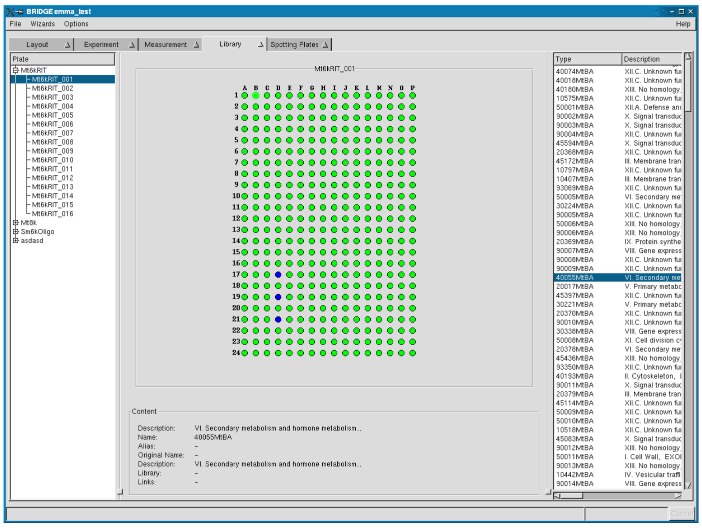
Apart from the ability to store such expression profiles in the database and to associate experimental factors with each slide, EMMA also supports cluster analysis (often only called clustering). Cluster analysis is a means for grouping genes having similar expression profiles. Many clustering algorithms have been developed and implemented. We have integrated some of the most widely used clustering algorithms for expression analysis using R-packages: k-means clustering [Mac67], hierarchical clustering [Mur85], SOMs [Koh97] as well as the PAM, CLARA and FANNY algorithms introduced by [KR90].

Additionally, we have implemented a parallel version of the PAM algorithm (called "PaPAM") in C++ and integrated it into the development version of EMMA. PaPAM utilizes multiprocessor computers to speed up cluster analysis.

7.2.9. User interfaces

EMMA is equipped with both a web-based frontend and a Gtk-based graphical user interface (GUI). The GUI is designed for local installations of EMMA on small laptop computers up to large scale servers. It features the design concept of wizards, which are a means to easily enter many parameters for data import or computation.

As displayed in figure 7.24, EMMA can be used to store and manage library data, e.g. the information about 96 or 384 well micro-titer plates and their content.



Content

Description: VI. Secondary metabolism and hormone metabolism...
Name: 40055MtBA
Alias: -
Original Name: -
Description: VI. Secondary metabolism and hormone metabolism...
Library: -
Links: -

Type	Description
40074MtBA	XII.C. Unknown fu
40018MtBA	XII.C. Unknown fu
40180MtBA	XIII. No homology.
10575MtBA	XII.C. Unknown fu
50001MtBA	XII.A. Defense and
90002MtBA	X. Signal transduc
90003MtBA	X. Signal transduc
90004MtBA	XII.C. Unknown fu
45594MtBA	X. Signal transduc
20368MtBA	XII.C. Unknown fu
45172MtBA	III. Membrane tran
10797MtBA	XII.C. Unknown fu
10407MtBA	III. Membrane tran
93069MtBA	XII.C. Unknown fu
50005MtBA	VI. Secondary me
30224MtBA	XII.C. Unknown fu
90005MtBA	XII.C. Unknown fu
50006MtBA	XIII. No homology.
90006MtBA	XIII. No homology.
20369MtBA	IX. Protein synthe
90007MtBA	VIII. Gene express
90008MtBA	XII.C. Unknown fu
90009MtBA	XII.C. Unknown fu
40055MtBA	VI. Secondary me
20017MtBA	V. Primary metabc
45397MtBA	XII.C. Unknown fu
30221MtBA	V. Primary metabc
20370MtBA	XII.C. Unknown fu
90010MtBA	XII.C. Unknown fu
30338MtBA	VIII. Gene express
50008MtBA	XI. Cell division c
20378MtBA	VI. Secondary me
45436MtBA	XIII. No homology.
93350MtBA	XII.C. Unknown fu
40193MtBA	II. Cytoskeleton,
90011MtBA	X. Signal transduc
20379MtBA	III. Membrane tran
45114MtBA	XII.C. Unknown fu
50009MtBA	XII.C. Unknown fu
50010MtBA	XII.C. Unknown fu
10518MtBA	XII.C. Unknown fu
45083MtBA	X. Signal transduc
90012MtBA	XIII. No homology.
50011MtBA	I. Cell Wall, EXO
90013MtBA	XIII. No homology.
10442MtBA	IV. Vesicular traffi
90014MtBA	VIII. Gene express

Figure 7.24.: The library interface of the EMMA Gtk GUI provides access to physical library data. A selected microtiter plate which is stored in the refrigerator is visualized as a grid of circles and the user can access the content of each well by clicking on a circle.

Since a slide is mostly not prepared by using the original library plates, EMMA also features the concept of storing spotting plates that were used for the spotting robot. The *SpottingPlate* interface shown in figure 7.25 thus displays for a selected spot the corresponding well and its content from the original library.

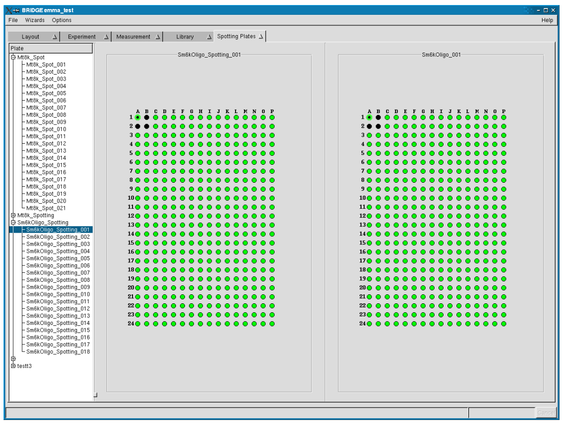
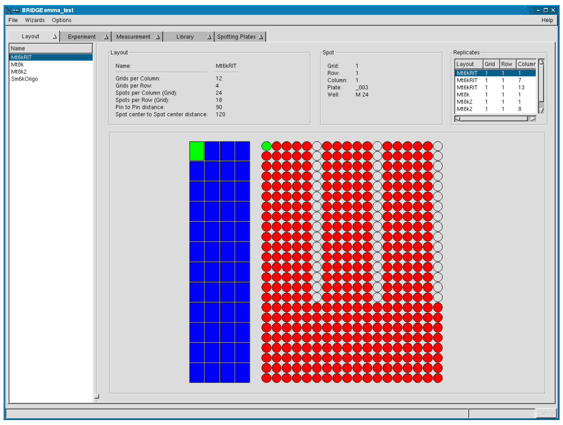


Figure 7.25.: The visualization of spotting plates can be used to organize and reproduce rearrangements of different microtiter plates. Spotting plates that were temporarily created and only used for printing a microarray can be stored and the mapping to the original plates can be visualized by clicking on a well.



Name
Mt6kRIT
Mt8k
Mt8k2
Sm6kOligo

Layout

Name:	Mt6kRIT
Grids per Column:	12
Grids per Row:	4
Spots per Column (Grid):	24
Spots per Row (Grid):	18
Pin to Pin distance:	90
Spot center to Spot center distance:	120

Spot

Grid:	1
Row:	1
Column:	1
Plate:	_003
Well:	M 24

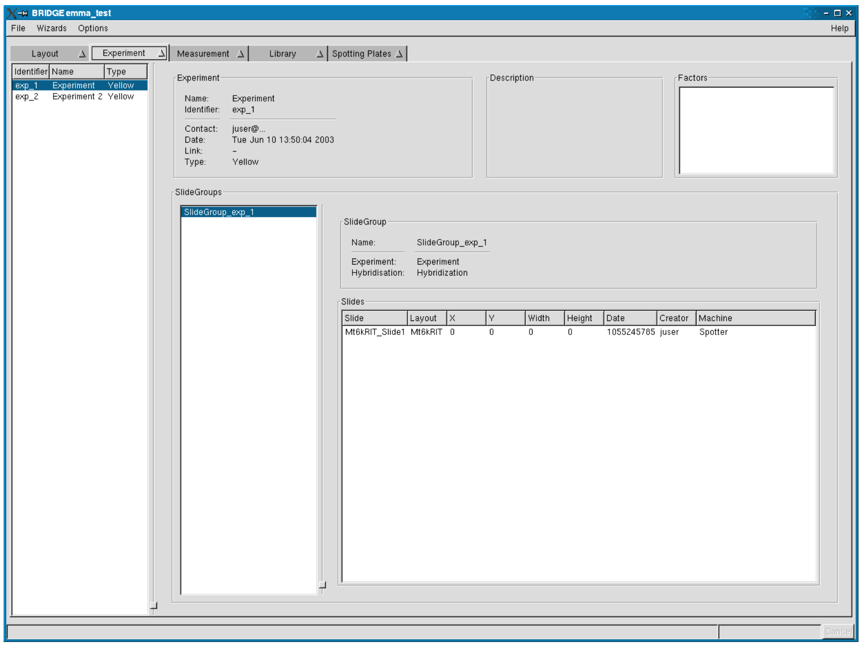
Replicates

Layout	Grid	Row	Columr
Mt6kRIT	1	1	1
Mt6kRIT	1	1	7
Mt6kRIT	1	1	13
Mt8k	1	1	1
Mt8k2	1	1	1
Mt8k2	1	1	8

Figure 7.26.: The visualization of different layouts that were used for printing sets of slides is essential for mapping the spots on a slide to their corresponding positions in microtiter plates. This assignment thus contains the information about the positions (including replicates) of a gene on a slide.

The physical layout employed for spotting a series of slides is managed and visualized using the *SlideLayout* interface of EMMA displayed in figure 7.26. Upon selection of a grid on a slide, the spots are enlarged and the user can check the content of each spotted dot.

The design of a microarray experiment can be described and modified using the *ExperimentEditor* displayed in figure 7.27. Among the most important properties for each experiment, slide groups can be defined that were employed for an experiment.



Identifier	Name	Type
exp_1	Experiment	Yellow
exp_2	Experiment 2	Yellow

Experiment

Name:	Experiment
Identifier:	exp_1
Contact:	juser@...
Date:	Tue Jun 10 13:50:04 2003
Link:	-
Type:	Yellow

SlideGroups

SlideGroup_exp_1

SlideGroup

Name:	SlideGroup_exp_1
Experiment:	Experiment
Hybridisation:	Hybridization

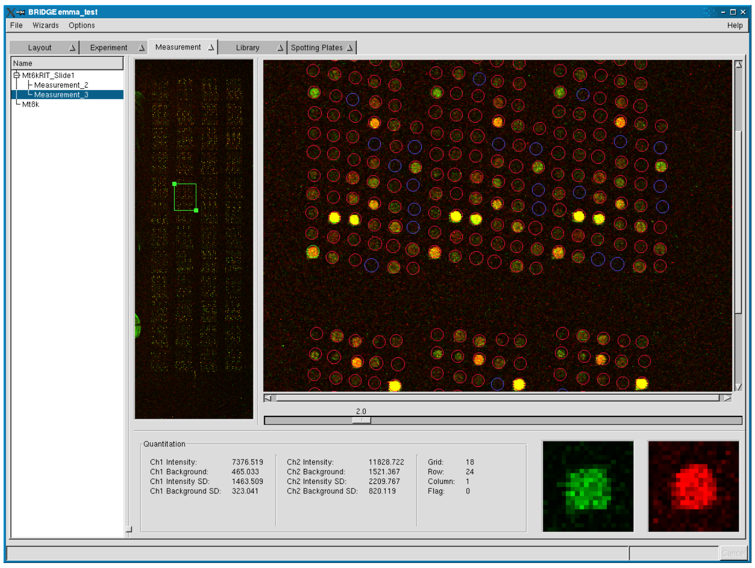
Slides

Slide	Layout	X	Y	Width	Height	Date	Creator	Machine
Mt6kRIT_Slide1	Mt6kRIT	0	0	0	0	1055245785	juser	Spotter

Figure 7.27.: The experiment editor is used for creating and managing microarray experiments. The user has to enter all available information about the experimental design and the slides that were used.

The *MeasurementBrowser* shown in figure 7.28 features the visualization of scanned images. Upon selection of a spot two separate images for both channels are displayed and the measured values are listed.

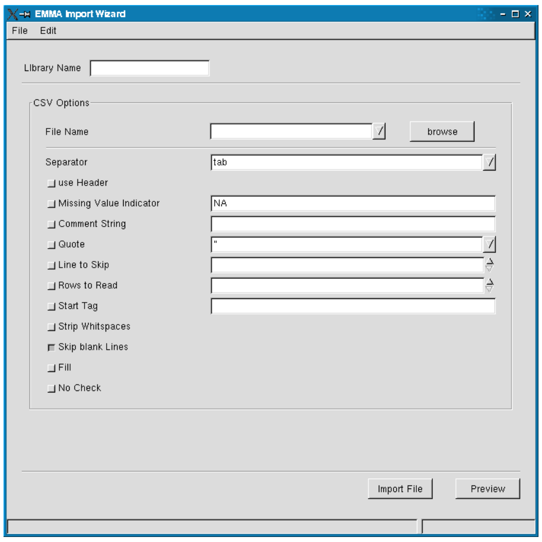
All data import is supported by a comfortable and easy-to-use *ImportWizard* as illustrated in figure 7.29. It features enhanced import facilities for predefined and arbitrary file types including a preview mode that allows the user to check the result in advance.



Quantitation

Ch1 Intensity:	7376.519	Ch2 Intensity:	11828.722	Grid:	18
Ch1 Background:	465.033	Ch2 Background:	1521.367	Row:	24
Ch1 Intensity SD:	1463.509	Ch2 Intensity SD:	2209.767	Column:	1
Ch1 Background SD:	323.041	Ch2 Background SD:	820.119	Flag:	0

Figure 7.28.: Measurements that were created using an image analysis software can be imported and visualized. Slide images can be inspected and by clicking on a spot, both channels and the measured values are displayed.



Library Name

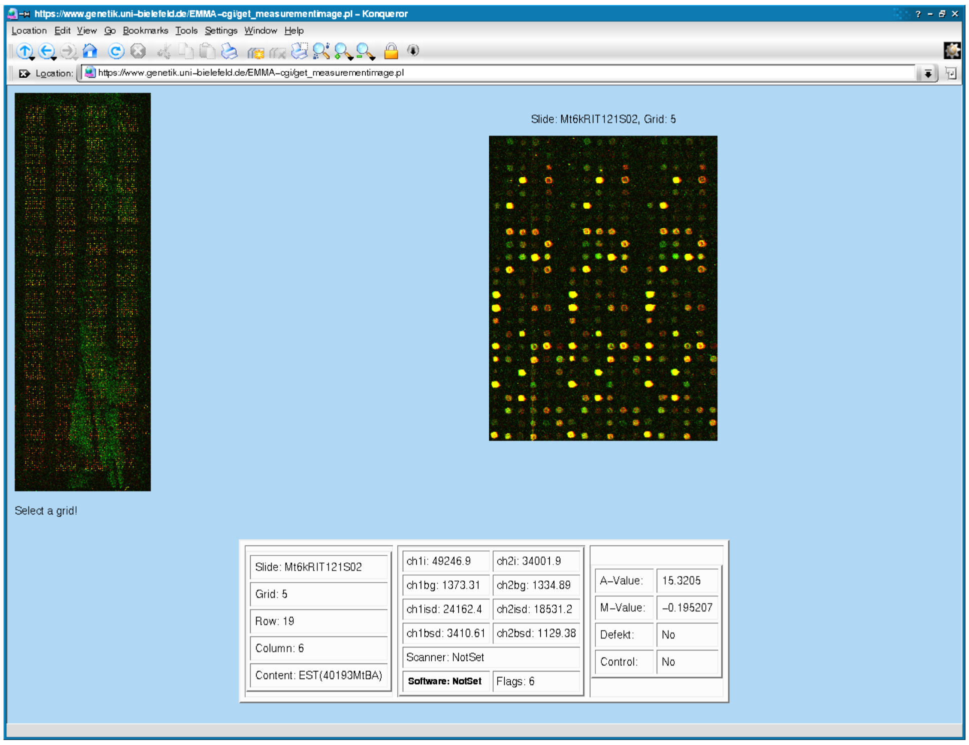
CSV Options

File Name	
Separator	tab
☐ use Header	
☐ Missing Value Indicator	NA
☐ Comment String	
☐ Quote	"
☐ Line to Skip	
☐ Rows to Read	
☐ Start Tag	
☐ Strip Whitspaces	
☑ Skip blank Lines	
☐ Fill	
☐ No Check	

Figure 7.29.: EMMA provides a comfortable import wizard for uploading many different kinds of data. The parser can be adjusted for reading different file types and features a number of options for customizing the importer.

The web interface allows remote access to a server installation of EMMA. It provides the full functionality of the program while enabling remote users to access shared data on a central server. For using the web interface via the Internet only a standard web browser is required. The screenshots below show selected interfaces of the EMMA web frontend.

As illustrated in figure 7.30, the user can view the imported slides and zoom into each grid. The spot coordinates (slide, grid, row, column) and measured intensities (intensity, background, standard deviation, and background standard deviation for both channels) can be displayed by clicking on a spot. Furthermore, the computed A and M values and the status of the spot (defect/control) are listed.



https://www.genetik.uni-bielefeld.de/EMMA-cgi/get_measurementimage.pl – Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location:
https://www.genetik.uni-bielefeld.de/EMMA-cgi/get_measurementimage.pl

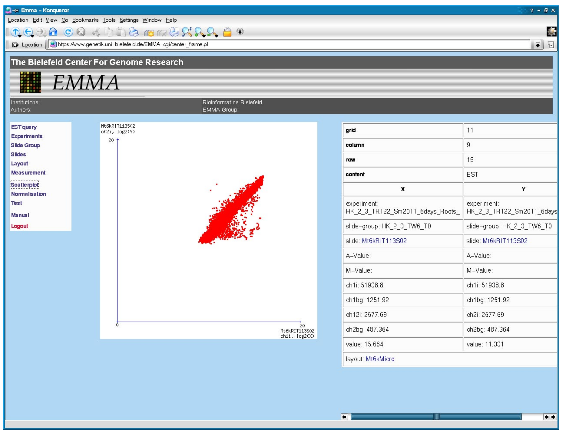
Slide: Mt6kRIT121S02, Grid: 5

Select a grid!

Slide: Mt6kRIT121S02	ch1i: 49246.9	ch2i: 34001.9		
Grid: 5	ch1bg: 1373.31	ch2bg: 1334.89	A–Value:	15.3205
Row: 19	ch1isd: 24162.4	ch2isd: 18531.2	M–Value:	–0.195207
Column: 6	ch1bsd: 3410.61	ch2bsd: 1129.38	Defekt:	No
Content: EST(40193MtBA)	Scanner: NotSet		Control:	No
	Software: NotSet	Flags: 6		

Figure 7.30.: The visualization of slides and measured spot intensities is also available in the web frontend. By clicking on a spot, the measured intensities are displayed.

The EMMA web frontend features an easy-to-use wizard for creating arbitrary scatter plots (see figure 7.31). After selecting a dataset and the parameters for the plot (e.g. type of plotted values, x/y axis), an interactive plot is generated that can be used to navigate through the spots.

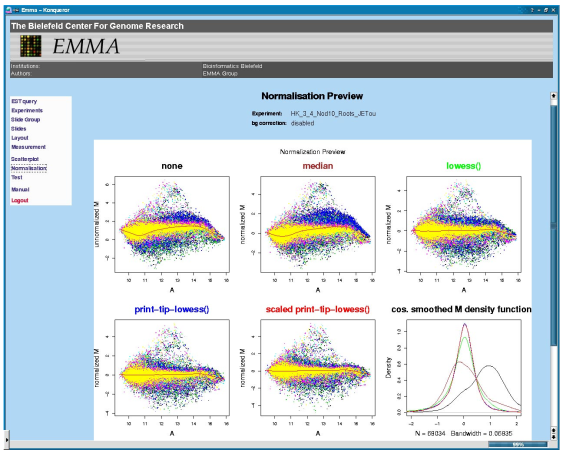


Institutions: Bioinformatics Bielefeld
Authors: EMMA Group

grid	11
column	9
row	19
content	EST
X	Y
experiment: HK_2_3_TR122_Sm2011_6days_Roots_	experiment: HK_2_3_TR122_Sm2011_6days
slide-group: HK_2_3_TW6_T0	slide-group: HK_2_3_TW6_T0
slide: Mt6kRIT113S02	slide: Mt6kRIT113S02
A-Value:	A-Value:
M-Value:	M-Value:
ch1i: 51938.8	ch1i: 51938.8
ch1bg: 1251.92	ch1bg: 1251.92
ch12i: 2577.69	ch2i: 2577.69
ch2bg: 487.364	ch2bg: 487.364
value: 15.664	value: 11.331
layout: Mt6kMicro	

Figure 7.31.: A sample scatterplot created with the EMMA web frontend. Such plots can be created for different data sets and parameters. By clicking on a spot, the corresponding data (experiment, intensities, layout, etc.) are displayed.

Different methods for normalization can be applied to the measured microarray data. Before storing the normalized values, a preview of the results can be displayed for the available normalization methods as illustrated in figure 7.32.



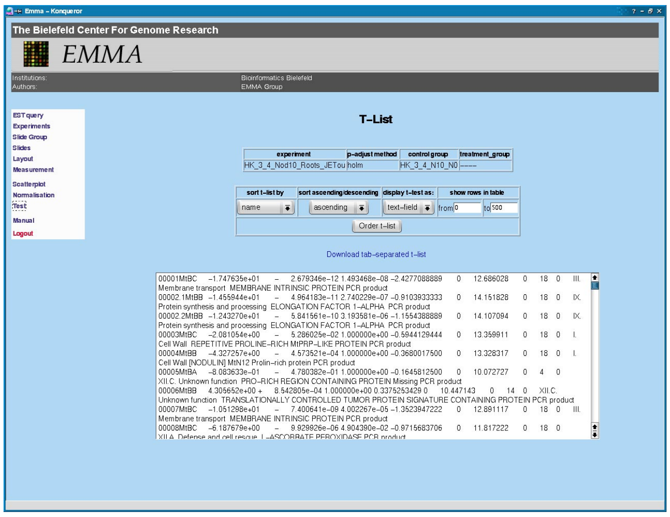
Institutions: Bioinformatics Bielefeld
Authors: EMMA Group

Normalisation Preview

Experiment: HK_3_4_Nod10_Roots_JETou
bg correction: disabled

Figure 7.32.: A preview of normalization results shows how systematic errors can be corrected by different methods. For the sample data shown above a median, lowess, print-tip lowess, and scaled print-tip lowess normalization was performed and compared to the none normalized data.

The EMMA web frontend currently provides data analysis by employing Student's t-test. Figure 7.33 shows a sortable list resulting from such a test.



Institutions: Bioinformatics Bielefeld
Authors: EMMA Group

T–List

experiment	p–adjust method	control group	treatment_group
HK_3_4_Nod10_Roots_JETou	holm	HK_3_4_N10_N0	––––

sort t–list by	sort ascending/descending	display t–test as:	show rows in table	
name	ascending	text–field	from 0	to 500

```
00001MtBC    –1.747635e+01    –    2.679346e–12 1.493468e–08 –2.4277088889    0    12.686028    0    18   0    III.
Membrane transport  MEMBRANE INTRINSIC PROTEIN PCR product
00002.1MtBB  –1.455944e+01    –    4.964183e–11 2.740229e–07 –0.9103933333    0    14.151828    0    18   0    IX.
Protein synthesis and processing  ELONGATION FACTOR 1–ALPHA  PCR product
00002.2MtBB  –1.243270e+01    –    5.841561e–10 3.193581e–06 –1.1554388889    0    14.107094    0    18   0    IX.
Protein synthesis and processing  ELONGATION FACTOR 1–ALPHA  PCR product
00003MtBC    –2.081054e+00    –    5.286025e–02 1.000000e+00 –0.5944129444    0    13.359911    0    18   0    I.
Cell Wall  REPETITIVE PROLINE–RICH MtPRP–LIKE PROTEIN PCR product
00004MtBB    –4.327257e+00    –    4.573521e–04 1.000000e+00 –0.3680017500    0    13.328317    0    18   0    I.
Cell Wall [NODULIN] MtN12 Prolin–rich protein PCR product
00005MtBA    –8.083633e–01    –    4.780382e–01 1.000000e+00 –0.1645812500    0    10.072727    0    4    0
XII.C. Unknown function  PRO–RICH REGION CONTAINING PROTEIN Missing PCR product
00006MtBB    4.305652e+00 +    8.542805e–04 1.000000e+00 0.3375253429 0    10.447143    0    14    0    XII.C.
Unknown function  TRANSLATIONALLY CONTROLLED TUMOR PROTEIN SIGNATURE CONTAINING PROTEIN PCR product
00007MtBC    –1.051298e+01    –    7.400641e–09 4.002267e–05 –1.3523947222    0    12.891117    0    18   0    III.
Membrane transport  MEMBRANE INTRINSIC PROTEIN PCR product
00008MtBC    –6.187679e+00    –    9.929926e–06 4.904390e–02 –0.9715683706    0    11.817222    0    18   0
XII.A. Defense and cell rescue  L–ASCORBATE PEROXIDASE PCR product
```

Figure 7.33.: A t-list result displayed in the EMMA web frontend shows significantly up or down regulated genes. This list can be sorted according to different settings and exported into a flat file.

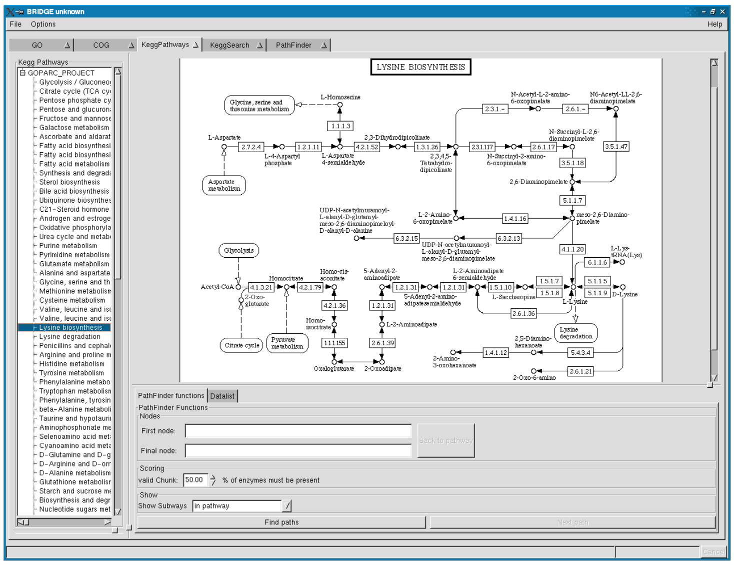
Comprehensive online documentation and online help is also available with both interfaces to guide the user through the setup and operation of the software and through setting parameters of analysis.

7.3. GOPArc

GOPArc (Gene Ontology and Pathway Architecture, unpublished) has been designed as a combined architecture for the analysis and visualization of Gene Ontologies, functional classifications and pathway data. The current prototype implementation contains a GO and COG browser, a KEGG pathway browser, a search interface, and the PathFinder system. Different types of data (e.g. annotated genes, expression profiles, etc.) can be mapped onto the pathways and functional categories thus representing meta views and additional information (see sample applications in chapter 9).

7.3.1. Metabolic pathways

The implementation of a component for the analysis and visualization of metabolic pathways was based on the PathFinder system and extends its functionality in several ways. The complete set of the KEGG metabolic pathways is now imported into the database described in [GHM$^+$02] as directed graphs. For navigating the KEGG maps, a completely redesigned Gtk interface (see figure 7.34) was implemented that allows an interactive use.



First node:

Final node:

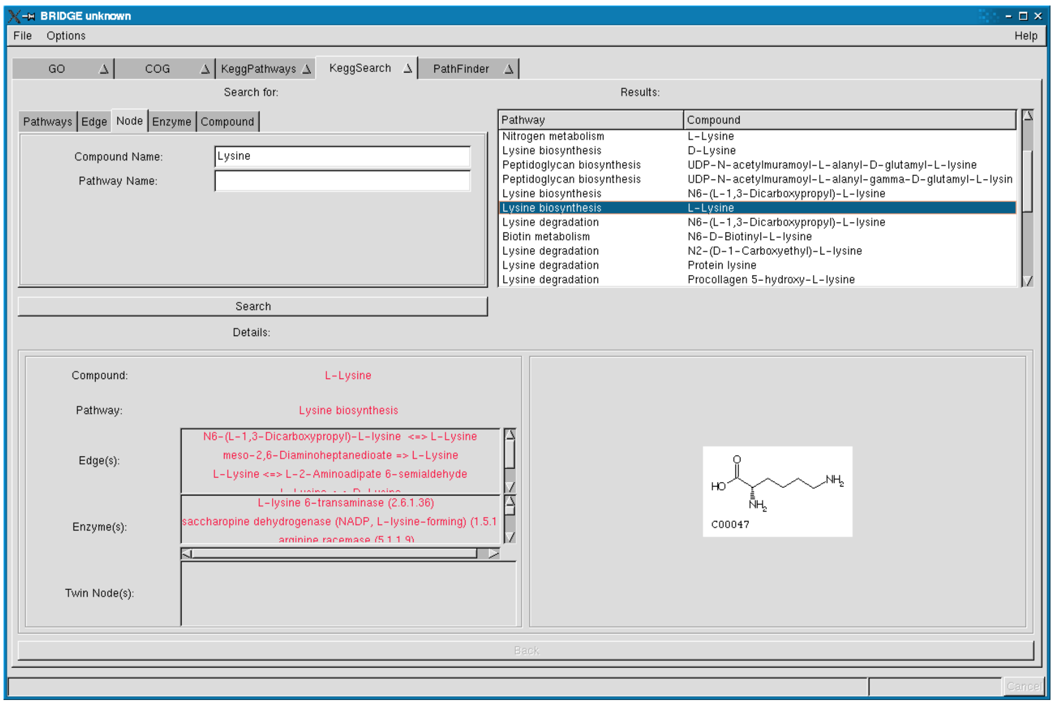
valid Chunk: 50.00 % of enzymes must be present

Show Subways: in pathway

Figure 7.34.: The KEGG browser can be used for visualizing the KEGG metabolic pathways. The interactive maps provide links to the enzyme and compound databases. Pathways can also be analyzed by using the search functionality of the PathFinder system which has been integrated.

Each rectangle containing an EC number can be colorized in a user defined way, e.g. marking the annotated enzymes of an organism is possible. Expression levels can be visualized as boxes surrounding an EC number and an additional plot can be opened for zooming and displaying the exact ratios.

Furthermore, a search interface was implemented that allows browsing the database and searching for pathways, nodes, edges, enzymes, and compounds as illustrated in figure 7.35.



Compound Name: Lysine

Pathway Name:

Pathway	Compound
Nitrogen metabolism	L-Lysine
Lysine biosynthesis	D-Lysine
Peptidoglycan biosynthesis	UDP-N-acetylmuramoyl-L-alanyl-D-glutamyl-L-lysine
Peptidoglycan biosynthesis	UDP-N-acetylmuramoyl-L-alanyl-gamma-D-glutamyl-L-lysin
Lysine biosynthesis	N6-(L-1,3-Dicarboxypropyl)-L-lysine
Lysine biosynthesis	L-Lysine
Lysine degradation	N6-(L-1,3-Dicarboxypropyl)-L-lysine
Biotin metabolism	N6-D-Biotinyl-L-lysine
Lysine degradation	N2-(D-1-Carboxyethyl)-L-lysine
Lysine degradation	Protein lysine
Lysine degradation	Procollagen 5-hydroxy-L-lysine

Compound: L-Lysine

Pathway: Lysine biosynthesis

Edge(s):
N6-(L-1,3-Dicarboxypropyl)-L-lysine <=> L-Lysine
meso-2,6-Diaminoheptanedioate => L-Lysine
L-Lysine <=> L-2-Aminoadipate 6-semialdehyde

Enzyme(s):
L-lysine 6-transaminase (2.6.1.36)
saccharopine dehydrogenase (NADP, L-lysine-forming) (1.5.1
arginine racemase (5.1.1.9)

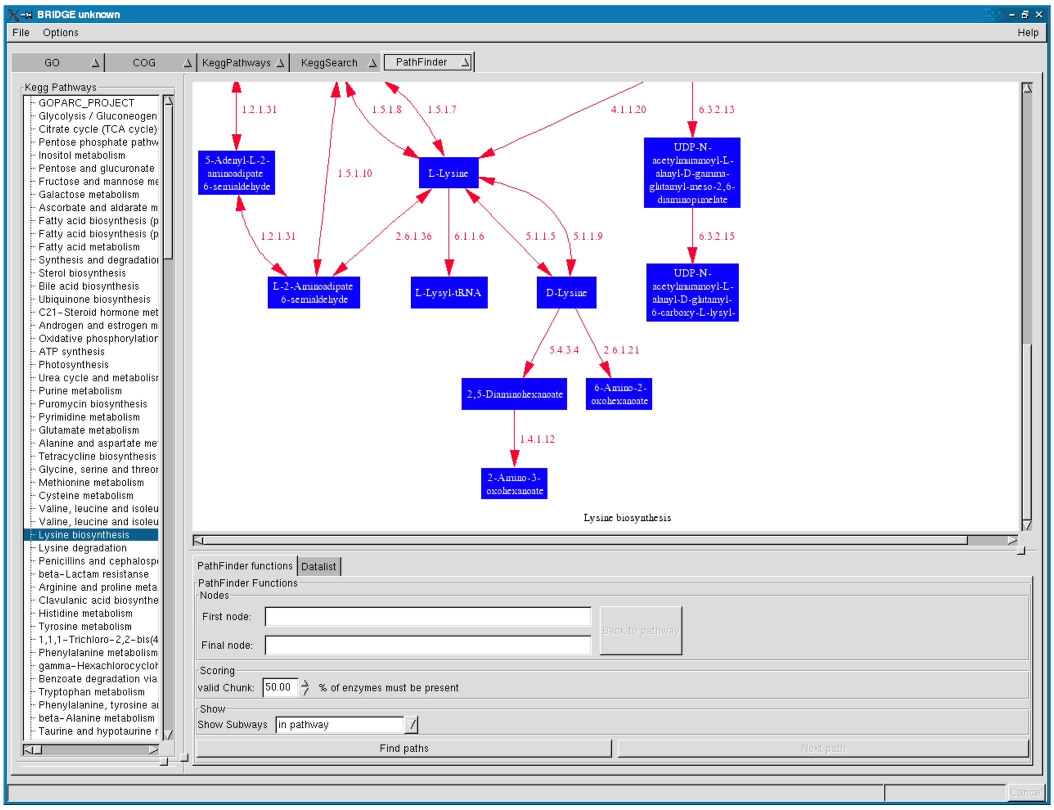
Twin Node(s):

Figure 7.35.: The KEGG-Search interface can be used to browse the database and search for specific elements, e.g. special enzymes or chemical compounds. Search results are always linked to their corresponding data and chemical reactions are displayed in detail. For most of the compounds an image of the structural formula is also available.

The KEGG-Search interface also displays the chemical formula of compounds and provides extensive hyperlinks to related data.

For the PathFinder system we have implemented a separate graphical user interface shown in figure 7.36 which now uses the graph drawing software GraphViz [5] instead of xvcg [San95] for dynamically visualizing a metabolic pathway.

[5] *http://www.research.att.com/sw/tools/graphviz/*



valid Chunk: 50.00 % of enzymes must be present

Show Subways: in pathway

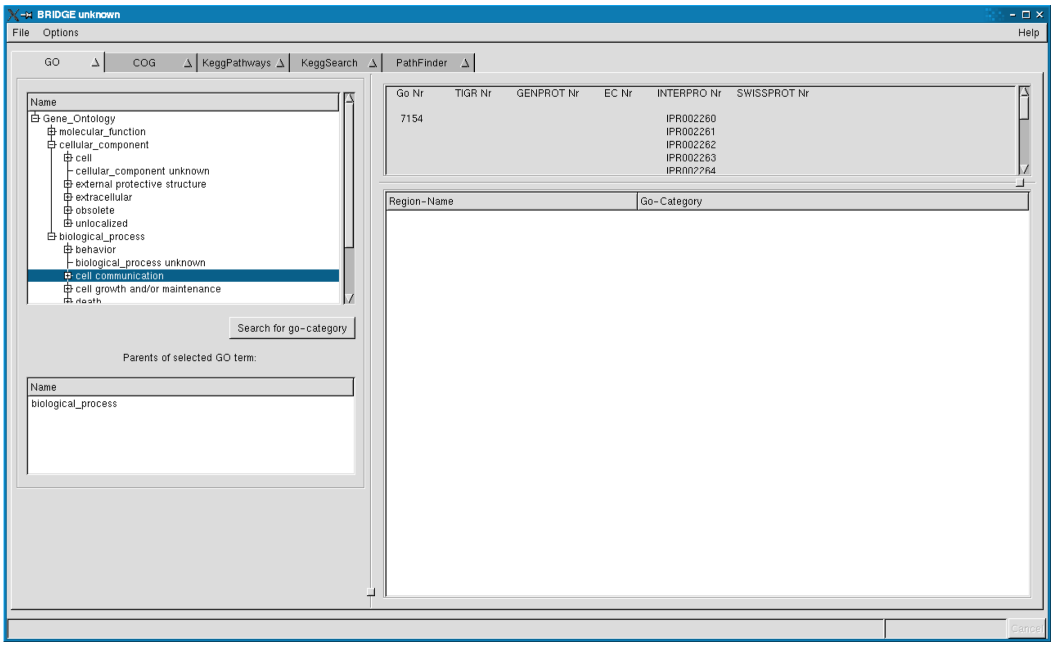
Figure 7.36.: The PathFinder interface provides dynamically generated interactive maps for the KEGG metabolic pathways. Since all visualization is based on graph drawing algorithms, arbitrary pathways that are stored as a graph can be displayed in addition to the KEGG maps.

Furthermore, the displayed maps were enriched with interactive navigation facilities and extended by enhanced visualizations.

Both the KEGG and the PathFinder browser provide the functionality for analyzing chunks and subways in a pathway. Therefore, the user has to select a starting node and an end node by clicking on a compound in the pathway map. Chunks and subways are thus computed on demand and finally highlighted in the selected pathway.

7.3.2. Functional categories

For the analysis of functional categories the publicly available GO database (see also section 3.4.5) was imported into the existing pathway database and the database schema was translated into an O2DBI description for generating the API modules. These modules were then extended by additional special purpose functions. The graphical user interface displayed in figure 7.37 features a tree-view for navigating the GO categories, TIGR roles, or the GenProt classification.



Name
- Gene_Ontology
  - molecular_function
  - cellular_component
    - cell
    - cellular_component unknown
    - external protective structure
    - extracellular
    - obsolete
    - unlocalized
  - biological_process
    - behavior
    - biological_process unknown
    - cell communication
    - cell growth and/or maintenance
    - death

Name
biological_process

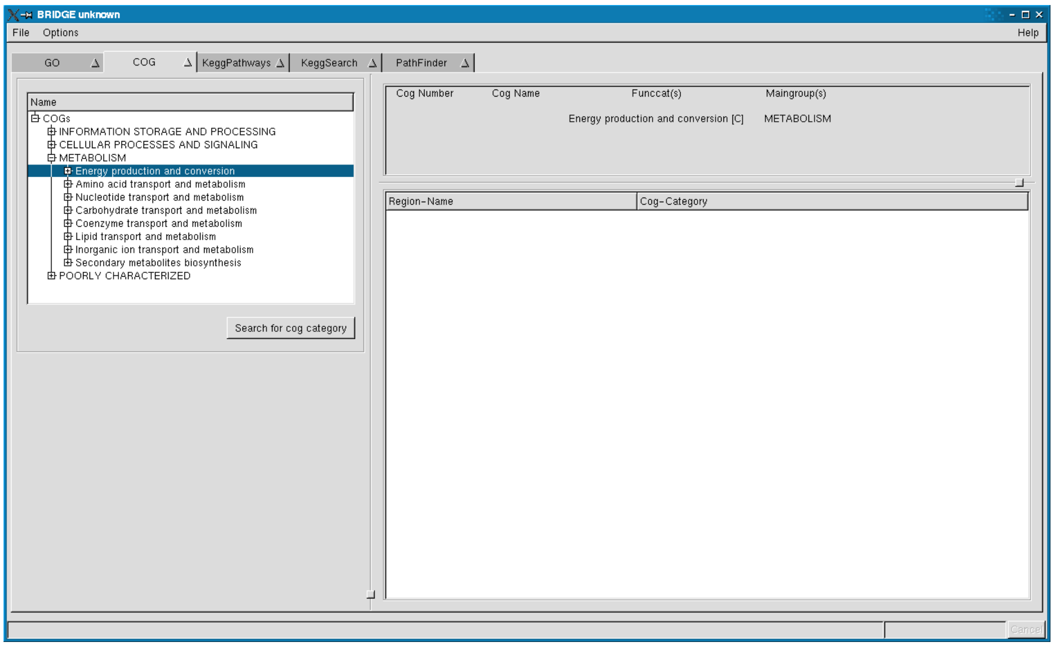
Go Nr	TIGR Nr	GENPROT Nr	EC Nr	INTERPRO Nr	SWISSPROT Nr
7154				IPR002260	
				IPR002261	
				IPR002262	
				IPR002263	
				IPR002264	

Region-Name	Go-Category

Figure 7.37.: The GO browser features a tree-view for navigating different functional categories. Since the Gene Ontologies are represented as a DAG, a single node can have several parent nodes which in such case are displayed in a separate list.

Upon selection of an item in the tree, additional information about the category is listed at the top and potential cross-references (mappings) to other categories are displayed. As a sample application, this interface can be integrated into the GenDB system and the user can be provided with a list of candidate genes for a selected category.

A similar functionality was also implemented for browsing the COG categories (see section 3.4.6) as illustrated in figure 7.38. Here, the user can select an entry and obtain all genes that may belong to that category.



Name
- COGs
  - INFORMATION STORAGE AND PROCESSING
  - CELLULAR PROCESSES AND SIGNALING
  - METABOLISM
    - Energy production and conversion
    - Amino acid transport and metabolism
    - Nucleotide transport and metabolism
    - Carbohydrate transport and metabolism
    - Coenzyme transport and metabolism
    - Lipid transport and metabolism
    - Inorganic ion transport and metabolism
    - Secondary metabolites biosynthesis
  - POORLY CHARACTERIZED

Cog Number	Cog Name	Funccat(s)	Maingroup(s)
		Energy production and conversion [C]	METABOLISM

Region-Name	Cog-Category

Figure 7.38.: The COG browser can be used for navigating the clusters of orthologous groups. Upon selection of a specific category, additional descriptions are displayed (e.g. function description, main groups) and all regions that belong to the selected category are listed.

Such a list can thus serve as a starting point for the annotation of genes specific for a selected category. Once all corresponding regions of a chosen functional category have been extracted and visualized, the user can simply navigate to such a selected region and e.g. display it in the main window of the GenDB system.

CHAPTER 8

Implementation

Based on all prerequisites mentioned so far, the BRIDGE system has been developed as a **B**ioinformatics **R**esource for the **I**ntegration of heterogeneous **D**ata from **G**enomic **E**xplorations. The BRIDGE system itself has been designed as a general framework that allows incorporating different components for special tasks (e.g. a genome annotation component or an expression data analysis module).

Perl has been selected as the primary implementation language since it allows using a multitude of well-tested existing Perl modules from the BioPerl project. The widespread use of Perl in bioinformatics thus enables many researchers to use the BRIDGE system as a platform for their implementation of further genome analysis pipelines. To be able to offer an API to the outside world, the system requires a persistent storage layer. A relational storage backend (here MySQL) was selected, which provides a fast, reliable, and well tested storage subsystem.

The development of all integrated components was always driven by the postulation that each module should also be applicable as a stand-alone tool. Due to this versatility and extensibility, the system is beginning to show its usability as an open platform for systems biology that is ready for tomorrow's research tasks.

The following chapters describe the most important features for each implemented component of the BRIDGE system.

8.1. Project-Management

Due to the BRIDGE system architecture described in section 6, we are now facing a situation where we have well-designed and full-featured systems for special purpose tasks (e.g. a genome annotation system for sequence analysis and a plug-in for the visualization of metabolic pathways) but none of the components “is aware of” each other. Hence we have to provide a means of control and a mechanism to connect two or more of these systems. Therefore we have implemented a *General Project-Management System* (GPMS) that organizes all data into projects. For example, a GenDB project is created for the sequence analysis and annotation of a genome with the GenDB software package or an EMMA project is set up for the microarray experiments. The integration of several such projects can then be achieved by defining so called *meta projects* that contain all required components (e.g. GenDB, EMMA, and GOPArc or two GenDB projects for genome comparison). In addition to the individual projects, the GPMS also stores information about all users that participate in a project. In order to gain access to a project’s data, a user has to become a member of the project. As illustrated in figure 8.5, each member has a well defined role (e.g. “Annotator”) that is further associated with different rights (e.g “is allowed to annotate”, “can edit sequence”). These rights are then translated into SQL privileges (e.g. for read or write access) and automatically enforced by the RDBMS when a user tries to access the *DataSources* of a project.

8.1.1. Design goals and specification

In this section, the concept for a *General Project-Management System* is described and the basic elements are explained. The system design was modeled using the Unified Modeling Language (UML).

Basic definitions for the *General Project-Management System*

For the design of the *General Project-Management System*, specific elements can be identified that represent real world objects or reflect the relationships between individual components involved in a project. These core objects are defined below in order to clarify their use in the following sections.

User A *User* simply represents an individual person that has at least a name, an account, and an e-mail address.

Project A *Project* indentifies a specific scope for research or ongoing work, e.g. a *Project* can be defined for the annotation of a newly sequenced bacterial genome. In most cases a

Project is defined for and related to a special software application, e.g. a genome annotation system.

Member A *User* has to become a *Member* of a *Project* for accessing the *Project's* data. Thus, a *Project* has a number of associated *Users* and each *User* "knows" about the *Projects* she/he is involved in.

Role The level of access to a *Project's* data can be specified by assigning well-defined *Roles* to each *Member*. A *Role* thus represents a set of access privileges or permissions (see *DB_Privileges* below).

Right Since most database management systems use their own access control mechanisms, access is not granted directly based on these privileges. The *General Project-Management System* therefore features the definition of *Rights* as free text descriptions that reflect a specific task for which a certain level of data access is required (e.g. `basic access` or `annotate`).

Project_Class Since all data access control should follow the same rules for every *Project* of the same application type, *Roles* and *Rights* are not defined for an individual *Project* but for a *Project_Class*. For example, the *Project_Class* GenDB uses the *Roles* guest, annotator, maintainer, developer, and chief for all genome annotation projects (see section A.1).

DB_Privileges *DB_Privileges* represent privileges that are used by a specific RDBMS for controlling access to individual databases. Thus all *Rights* have to be mapped onto appropriate *DB_Privileges*.

DataSource A *DataSource* describes a storage backend for *Project* related data that is located on a *Host* (e.g. a database server). This can be either a database (*DB*) stored on a DB server machine or an *ApplicationServer* that provides data e.g. via web services. A *DB* can be further specified by the type of the database management system (*DBMS_Type*).

DataSource_Type A special *DataSource_Type* can be used to determine the specific type of a *DataSource*. The *DataSource_Type* contains information about the internal structures of a *DataSource* (e.g. tables of a database) and thus all *DB_Privileges* refer to a corresponding *DataSource_Type*.

8.1.2. Specification of the *General Project-Management System*

Based on the definitions described so far, we have designed a *General Project-Management System* that organizes application specific data into *Projects*. We have also integrated the administration of *Users* and modeled the relationship of *Users* to *Projects* as *Roles*. Figure 8.1 illustrates the central components of the GPMS and their relationships in UML.

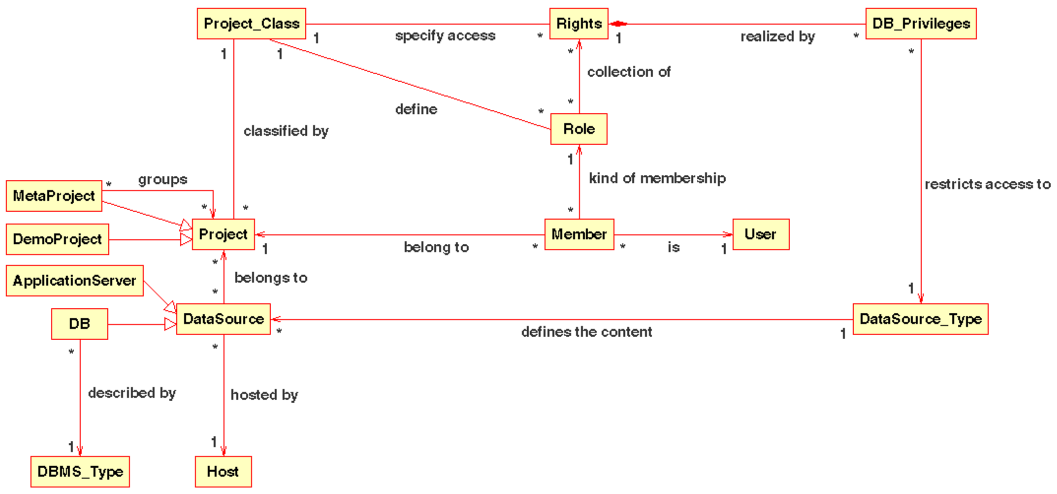


Figure 8.1.: The UML description of the *General Project-Management System* illustrates the role of each implemented class. A *User* has to become a *Member* of a *Project* in order to gain access to the *Project*'s *DataSources*. All access is controlled by special *Rights* which are associated with individual *Roles*. Each *Right* is realized by database specific privileges.

User and *Project* are the two core objects. Each *Project* specifies a separate well-defined scope that is uniquely identified by its name. Different *DataSources* that should be accessed in such a *Project* are referenced by a *Project* which allows several *Projects* to share the same *DataSources*. In addition to individual *Projects*, a *MetaProject* can be used to group several *Project* instances to form a new *Project* in order to correlate their data.

As another central component, *Users* are modeled in a very simple manner: They can be identified by their unique login name or e-mail address and are thus organized as separate objects. Their individual *Roles* in single *Projects* are defined by associating a *User*, a *Project*, and a *Role* in a *Member* object. In order to simplify the assignment of *Roles*, an additional class (*Project_Class*) has been included that can be used to group *Projects* of the same type (these are normally used by the same type of application). Thus *Roles* are only defined once for each *Project_Class* and not individually for each *Project*. A *Role* is further associated with a list of *Rights* that specify the level of access a *User* with this *Role* has for a *Project*. Each *Right* is defined by a comprehensive name which explains its semantics. It is associated with a list of *DB_Privileges* and thus mapped onto specific database privileges (e.g. *select* privilege in SQL). As an example, more than 30 different GenDB projects are currently maintained together by a single *Project_Class* in order to organize all data accesses in a consistent way.

8.1.3. Implementation

This chapter outlines the details of the implementation of the *General Project-Management System*. The data model for the relational database is described and the attributes of each class are explained. Sample descriptions for the definition of *Roles* and *Rights* complement this section by illustrating the implementation of fine-grained access control.

Database schema for the *General Project-Management System*

The *Project-Management System* was developed as an object-oriented application based on the data model described in the previous section. Since the GPMS requires a persistent storage backend the O2DBI-II system was used for the implementation. `MySQL` is currently used as the database backend but other relational database management systems can be used as they are supported by the O2DBI-II software. Figure 8.2 displays the current database schema for the GPMS database (GPMSDB) as it has been generated by O2DBI-II.

8.1.4. Class descriptions

The following sections briefly describe the relevant classes of the data schema (see figure 8.2). The five core classes *Project*, *User*, *Member*, *Role*, and *DataSource* are complemented by several simple classes that store additional information.

Project *Projects* are the central components of the GPMSDB. They connect all relevant classes. *Projects* have a unique name, an additional description, special configurations (e.g. global project settings), and a list of *DataSources*. Each *Project* belongs to a specific *Project_Class* that is described in the following paragraph. The general class *Project* has been extended by several subclasses that can be used to model individual properties and features of special types of *Projects* (e.g. a *Project::GENDB* defines an additional genetic code).

Project_Class A *Project_Class* arranges *Projects* of the same type into groups. Additionally, a *Project_Class* determines the available *Roles* and *Rights* for the individual *Projects*.

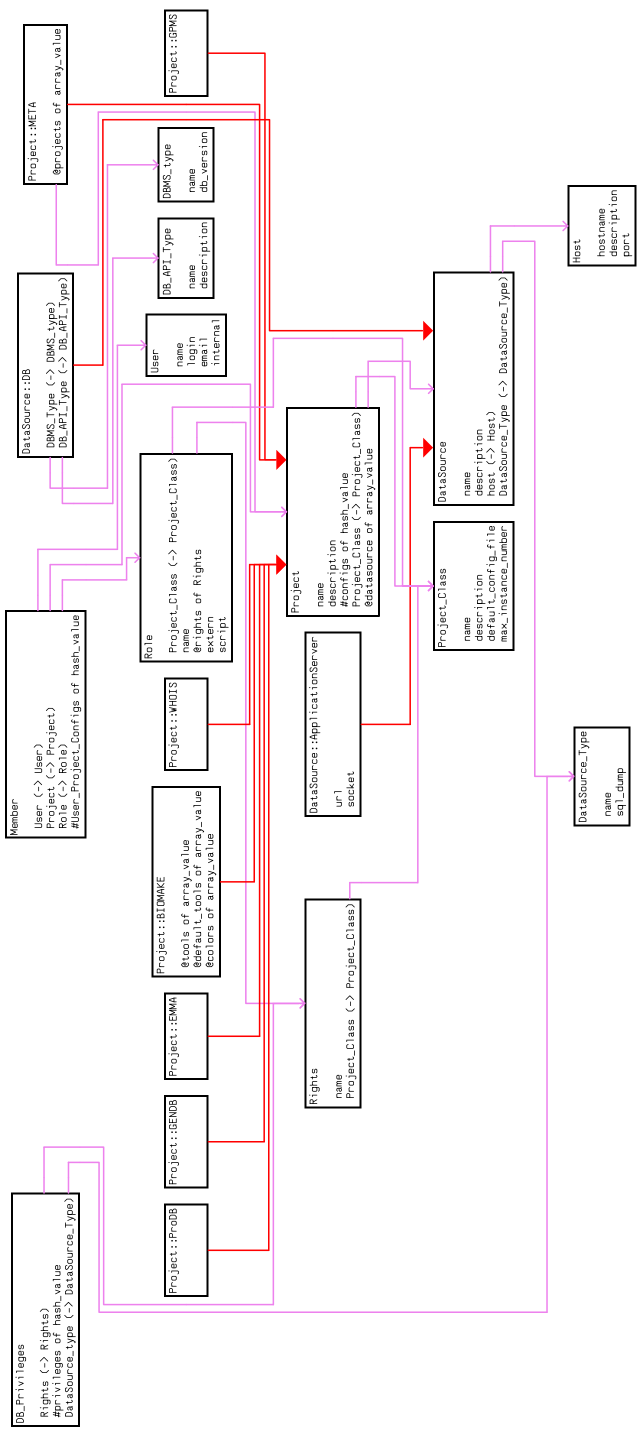


Figure 8.2.: The database schema of the *General Project-Management System*. Pink lines mark references between two classes and red lines show inherited class relationships.

Hence *Roles* and *Rights* do not have to be defined for every *Project* but only once for all *Projects* of the same *Project_Class*. This approach reduces the complexity of the system and the required work when administering *Projects*. Furthermore, a concise representation of all *Project* related data is achieved and a consistent use of *Roles* and their corresponding permissions can be ensured. A *Project_Class* has a name and a description. Optionally, an instance of a *Project_Class* can be associated with a configuration file that specifies standard configuration parameters.

User Basic user data is stored in *User* objects. Each *User* object has to provide a name, a login, and an e-mail address. A special flag can be set to determine whether the *User* is an internal or external user (e.g. to allow access only via web frontends running on a specific host).

Member A *User* has to become a *Member* of a *Project* in order to gain access to a *Project's* data. Therefore, a *Member* object relates a *User* to a *Project*. Each *Member* is also assigned a *Role* that defines the level of access for that *Project* (see description of **Role** below). Individual configuration items for a *Project* can be stored in the User_Project_Configs.

Role Each *Member* of a *Project* has a specific *Role* that determines the level of access and the *User's* permissions for that *Project*. For example, access can be granted on a very low level with read-only permissions or with complete access to all data, even with the right to delete everything. Therefore, *Roles* have a list of *Rights* that further define such privileges. It is important to note that *Roles* are defined for a *Project_Class*, not for each single *Project*. An additional "extern" flag can be used to allow setting this *Role* via the web frontend by external project managers. The "script" attribute can be used to store some code that is automatically executed for newly created *Members* with this *Role*.

Rights *Rights* determine the permissions of a *Member* for a specific *Project*. *Rights* are associated with database privileges. If a *User* is added to a *Project* as a *Member*, all the privileges determined by the *Rights* of the corresponding *Role* will be granted to the *User*. *Rights* are defined for a *Project_Class*, not for a single *Project*. This approach does not only simplify the maintenance of *Projects*, it also ensures a consistent use of *Roles* for all *Users* that have access to *Projects* of a specific *Project_Class*.

DB_Privileges DB_Privileges refer to a *Right* and represent the DBMS-specific access privileges. They are always defined for a specific *DataSource_Type* (see below).

DataSource Applications often need to access and to store persistent external data. These are provided as a *DataSource* which is attached to a *Project*. *DataSources* are characterized by a name, a description, a *Host*, and a *DataSource_Type*. The class “DataSource” has been extended by two special types of data-sources, a database (*DB*) and a so called *Application Server* (see description of **DataSource::ApplicationServer** below).

Host Information about the host of a typical *DataSource* are stored in *Host* objects. Each *Host* has a name, a port, and a description.

DataSource_Type The *DataSource_Type* determines the specific type of a *DataSource*. For databases, the *DataSource_Type* can also contain a reference to a file that defines the database schema so that a newly created database can be initialized with all table structures automatically after creation. A *Project* may only have **one** *DB* and **one** *ApplicationServer* of each *DataSource_Type* so that the connection to the correct *DataSource* can be established automatically.

DataSource::ApplicationServer An ApplicationServer is a generic *DataSource* that can provide data for an application. The class inherits all attributes of the *DataSource* class and has an additional url and a socket. It is important to note that a *Project* may only have **one** ApplicationServer of each *DataSource_Type* (see description of **DataSource_Type**).

DataSource::DB This class represents all kinds of databases and extends the *DataSource* class. In addition to all inherited attributes of the *DataSource* class, a *DataSource::DB* is described by a *DBMS_Type* and a *DB_API_Type*. It is important to note that a *Project* may only have **one** *DataSource::DB* of each individual *DataSource_Type* (see description of **DataSource_Type**).

DBMS_Type The *DBMS_Type* describes the database management system that is used to store the databases (e.g. `MySQL`). A *DBMS_Type* has a name and a version number.

DB_API_Type The *DB_API_Type* is a special class that can be used to define an `API` for accessing a database (*DataSource::DB*). For our purposes this is in most cases an O2DBI-I or an O2DBI-II interface.

Project::Meta *Project::Meta* is a generic *Project* subclass that combines several arbitrary *Projects* to a new *Project*. A meta-*Project* provides a list of member-*Projects* and inherits all the attributes of *Project*.

Project::GPMS *Project::GPMS* is a special subclass of *Projects* that can be used to refer to other *General Project-Management Systems*.

Project::GENDB *Project::GENDB* is a special subclass for *Projects* that use the GenDB system for the annotation of microbial genomes.

Project::EMMA *Project::EMMA* is a special subclass for microarray *Projects* using the EMMA software.

Project::ProDB *Project::ProDB* is a special subclass for proteomics *Projects* using the ProDB software [WRB$^+$03].

Project::BIOMAKE The *Project::BIOMAKE* class provides additional features for using the BIOMAKE software that can be employed for the automatic analysis of ESTs and sequencing reads.

Other software can be integrated into the GPMS by simply implementing a new subclass that contains the necessary code for initializing a connection to the corresponding data source. For all O2DBI-II projects, the default methods of the parent class can be used.

8.1.5. Interfaces

In this section different ways for accessing and using the *General Project-Management System* are described. In addition to the API that allows programmers to directly manipulate all objects stored in the database, a number of scripts for maintaining the system and a Gtk graphical user interface for the management of *Users* and *Project Members* are provided. A simplified web frontend was also implemented that supports a restricted user management for “external” maintainers of *Projects*. Thereby, project leaders of other research groups (here: those who are not located at Bielefeld University) can maintain the list of *Users* that should have access to specific *Projects*.

GPMS scripts

The *General Project-Management System* scripts provide a flexible way for configuring and maintaining *Projects* and their *Members*. Recurrent or frequent tasks can be automated by combining several scripts according to the individual needs of GPMS administrators. All scripts listed in table 8.1 can be used to initially set up the system and for maintaining projects, users, and their memberships.

The scripts listed in table 8.1 are executed using the wrapper script **gpms** which sets the installation specific environment variables. Executing this script without parameters will list all available scripts and print a usage message. As an example, a new user is added via the following commandline:

```
gpms add_user -l juser -f 'Joe User' -e Joe.User@CeBiTec.Uni-Bielefeld.DE
```

A new user can be added by specifying a login name, a full name and an optional e-mail address.

Using an *Application_Frame*

In addition to the standard classes of the *General Project-Management System* described in section 8.1.4, a general framework was implemented that simplifies the necessary steps for accessing a project's data. Such an *Application_Frame* uses the GPMS for accessing the *DataSources* of a *Project* and it also provides a number of useful methods that are often needed by the end applications. A detailed description of these methods can be found in appendix A.2.

adding datasets

name	description
add_host	add a new *Host* to the GPMS
add_datasource_type	register a new *DataSource* to the GPMS
add_db_api_type	register a new *API_Type* to the GPMS
add_dbms_type	create a *DBMS_Type* in the GPMS
add_db	create a new *Database* in the GPMS
add_project_class	create a new *ProjectClass* in the GPMS
add_project	create a new *Project* in the GPMS
add_datasource2project	add a *DataSource* to a *Project*
add_project_config	add configurations to a *Project*
add_role	read and store the *Role* definitions for a *ProjectClass*
add_rights	parse a *Right* definition-file and store it in the GPMS
add_user	register a new *User* in the GPMS
add_member	add an existing *User* as a new *Member* to a *Project*
add_meta_project	create a new *MetaProject* in the GPMS
add_project2meta_project	add a *Project* to a *MetaProject*

deleting data

name	description
del_host	remove a *Host* from the GPMS
del_datasource_type	delete a *DataSource_Type* from the GPMS
del_datasource	remove a *DataSource* from the GPMS
del_db_api_type	delete a *DB_API_Type* from the GPMS
del_dbms_type	remove a *DBMS_Type* from the GPMS

del_project_class	delete a *ProjectClass* from the GPMS
del_project	delete a *Project* from the GPMS
del_project_config	remove configurations from a *Project*
del_role	remove roles from a *ProjectClass*
del_rights	delete rights from a *ProjectClass*
del_user	remove a *User* from the GPMS
del_member	remove a *User* from a *Project*

other scripts

name	description
change_member_role	change the *Role* of an existing *Member*
export_members	print a list of all *Member*s of a *Project* or all *Member*s of all *Project*s of a *ProjectClass* to a file
rem_datasource_from_project	remove a *DataSource* from a *Project*
rem_project_from_meta_project	remove a *Project* from a *MetaProject*
list_project_members	print a list of all *Members* of a *Project*
list_projects	print a list of all *Project*s and *Roles* available for the *Project*
list_user_projects	display a list of all *Projects* that can be accessed by a *User*
list_extern_user	print list of all extern *Users*
gui	start the graphical user interface to maintain the GPMS

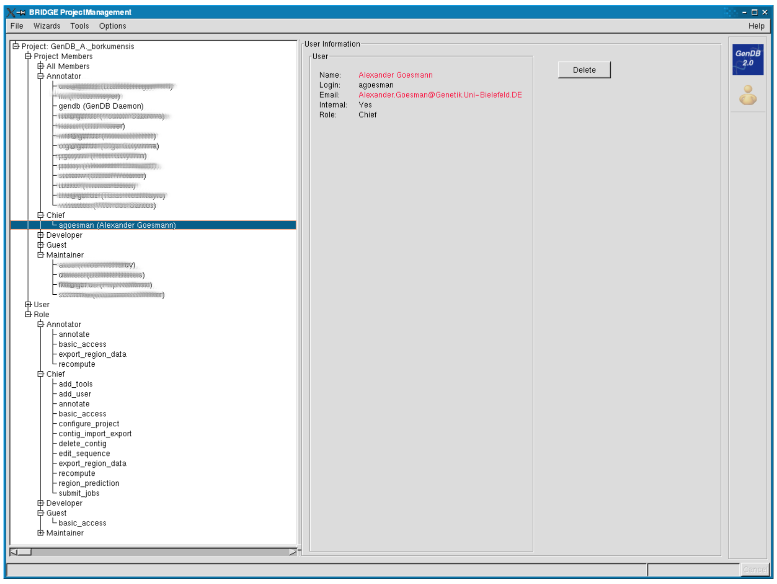
Table 8.1.: All currently implemented scripts for manipulating the GPMS. Executing a script without parameters will print a detailed description and a complete list of available options. Most of the scripts for adding and deleting data require special database permissions, e.g. CREATE, DROP, GRANT privileges in MySQL.

Graphical user interfaces for maintaining the GPMS

In addition to the scripts the *General Project-Management System* can be maintained via a graphical user interface implemented in Perl Gtk (see figure 8.3).

The tree-view on the left displays all *Members* of a selected *Project* sorted by the *Roles* that were defined for the corresponding *Project_Class*. Another subtree shows the *Rights* that have been defined for each *Role*. The tree also contains a list of all *Users* that are registered in the GPMS. New *Users* and *Members* can be added via a simple input form.

Figure 8.4 shows four screenshots of different operations that can be performed with the web frontend. All *Members* of a *Project* can be listed, new *Users* and *Members* can be added, or existing *Members* can be removed from a *Project*. It is also possible to change the *Role* of an existing *Member*. For reasons of security, only those *Roles* can be assigned that do not include administrative privileges and are therefore marked as *extern*.



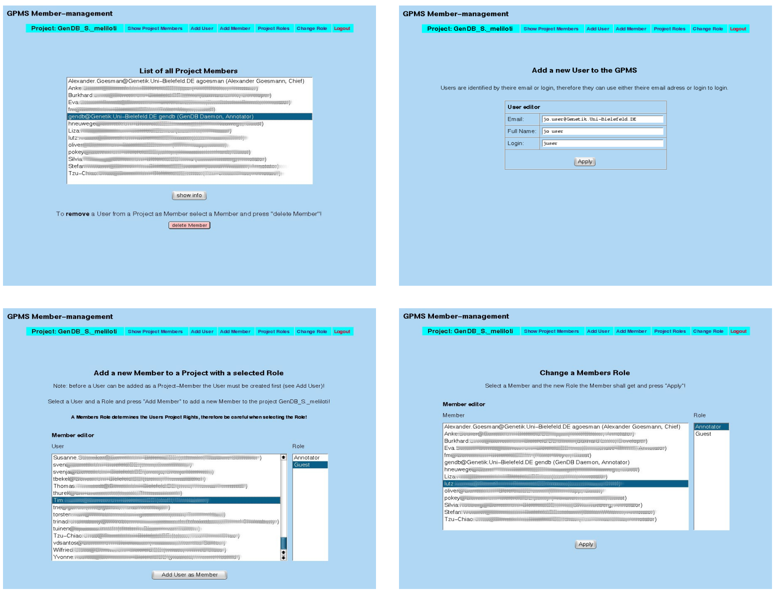
```
⊟ Project: GenDB_A._borkumensis
  ⊟ Project Members
    ⊞ All Members
    ⊟ Annotator
      ├ [redacted]
      ├ gendb (GenDB Daemon)
      └ [redacted]
    ⊟ Chief
      └ agoesman (Alexander Goesmann)
    ⊞ Developer
    ⊞ Guest
    ⊟ Maintainer
      └ [redacted]
  ⊞ User
  ⊟ Role
    ⊟ Annotator
      ├ annotate
      ├ basic_access
      ├ export_region_data
      └ recompute
    ⊟ Chief
      ├ add_tools
      ├ add_user
      ├ annotate
      ├ basic_access
      ├ configure_project
      ├ contig_import_export
      ├ delete_contig
      ├ edit_sequence
      ├ export_region_data
      ├ recompute
      ├ region_prediction
      └ submit_jobs
    ⊞ Developer
    ⊟ Guest
      └ basic_access
    ⊞ Maintainer
```

User Information

User

Name:	Alexander Goesmann
Login:	agoesman
Email:	Alexander.Goesman@Genetik.Uni-Bielefeld.DE
Internal:	Yes
Role:	Chief

Figure 8.3.: The Gtk frontend of the *General Project-Management System* can be used to maintain *Users* and *Projects* and provides an immediate overview.



List of all Project Members

Alexander.Goesman@Genetik.Uni–Bielefeld.DE agoesman (Alexander Goesmann, Chief)
Anke.[redacted]
Burkhard.[redacted]
Eva.[redacted]
fm[redacted]
gendb@Genetik.Uni–Bielefeld.DE gendb (GenDB Daemon, Annotator)
hneuwege[redacted]
Liza.[redacted]
lutz.[redacted]
oliver[redacted]
pokey[redacted]
Silvia.[redacted]
Stefan.[redacted]
Tzu–Chiao.[redacted]

To **remove** a User from a Project as Member select a Member and press "delete Member"!

Add a new User to the GPMS

Users are identified by theire email or login, therefore they can use either theire email adress or login to login.

User editor	
Email:	jo.user@Genetik.Uni-Bielefeld.DE
Full Name:	jo user
Login:	juser

Add a new Member to a Project with a selected Role

Note: before a User can be added as a Project–Member the User must be created first (see Add User)!

Select a User and a Role and press "Add Member" to add a new Member to the project GenDB_S._meliloti!

A Members Role determines the Users Project Rights, therefore be careful when selecting the Role!

Member editor

User: Susanne.[redacted] sven[redacted] svenja[redacted] tbekel[redacted] Thomas.[redacted] thurek[redacted] Tim.[redacted] tne[redacted] torsten.[redacted] trinad.[redacted] tuinen[redacted] Tzu–Chiao.[redacted] vdsantos[redacted] Wilfried.[redacted] Yvonne.[redacted]

Role: Annotator, [✓ Guest]

Change a Members Role

Select a Member and the new Role the Member shall get and press "Apply"!

Member editor

Member: Alexander.Goesman@Genetik.Uni–Bielefeld.DE agoesman (Alexander Goesmann, Chief) Anke.[redacted] Burkhard.[redacted] Eva.[redacted] fm[redacted] gendb@Genetik.Uni–Bielefeld.DE gendb (GenDB Daemon, Annotator) hneuwege[redacted] Liza.[redacted] lutz.[redacted] oliver[redacted] pokey[redacted] Silvia.[redacted] Stefan.[redacted] Tzu–Chiao.[redacted]

Role: [✓ Annotator], Guest

Figure 8.4.: The web frontend of the *General Project-Management System* can be used to manage *Users* and *Projects*.

8.1.6. Administering users

For the implementation of the BRIDGE system, the *General Project-Management System* was used to manage *Users* and *Projects*. In order to gain access to a project's data, a user has to become a *Member* of the *Project*. As illustrated in figure 8.5, each *Member* has a well defined *Role* (e.g. "Annotator") that is further associated with different *Rights* (e.g. "is allowed to annotate", "can edit sequence"). These *Rights* are then translated into SQL privileges (e.g. for read or write access) and granted to the *Member*.

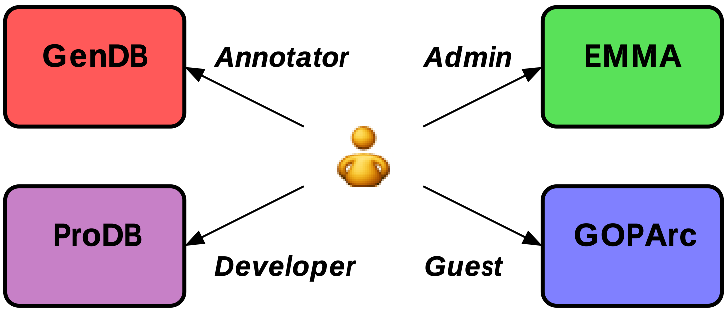


Figure 8.5.: Different *Roles* of a *User* for several *Projects*. For example, the *User* can be an "Annotator" for a GenDB *Project* and therefore has – beyond basic access privileges – the *Right* to create new annotations.

8.1.7. Accessing the data

Using the *Project-Management System* also allows to hide the *DataSource* from the *User*: only the *General Project-Management System* knows where to find the right database of a GenDB *Project* that is needed by the application (see figure 8.6). The *Project* itself contains the information about the corresponding *DataSource* and can establish the connection. Neither the user nor the graphical frontend of the application needs to know where the data originates from.

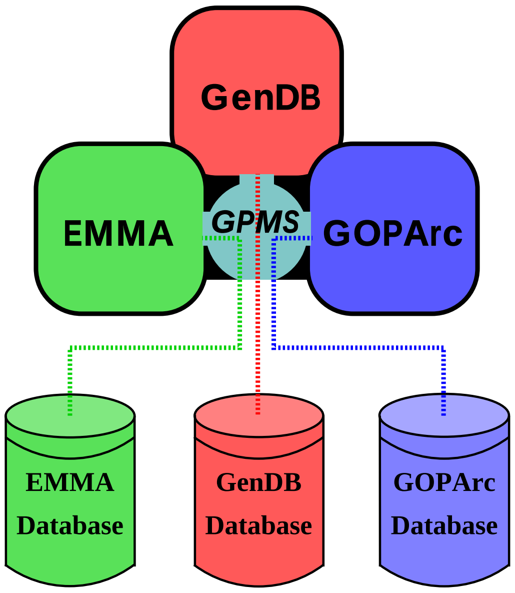


Figure 8.6.: All data access is controlled by the *Project-Management System*. The application does not need to know the *DataSources* of a *Project* since the *General Project-Management System* stores all information needed to establish connections (e.g. to relational database management systems).

In addition to the purposes just mentioned, the GPMS stores individual settings and configurations for *Projects* and their *Members* (e.g. settings for colors, etc.). The *Project-Management System* also includes a module for the session management of web-based frontends that can be used to prevent uncontrolled access to confidential project data.

8.2. BRIDGE

This section describes the implementation of the BRIDGE layer. Instead of simply linking different data sources, the BRIDGE system provides direct access to remote objects. This approach also allows the implementation of individual algorithms that can be derived directly from pseudo code descriptions.

8.2.1. Extension of O2DBI

As already shown in figure 6.4, BRIDGE was designed as a separate layer on top of the O2DBI-II server classes. In addition to all O2DBI-II server methods (auto-generated and manually added methods), the BRIDGE layer provides the auxiliary functionality that is necessary for retrieving "external" objects that are referenced in an initially loaded project. As an example, this mechanism is illustrated in figure 8.7 where an *EMMA::Spot* object references a *GenDB::CDS* region.

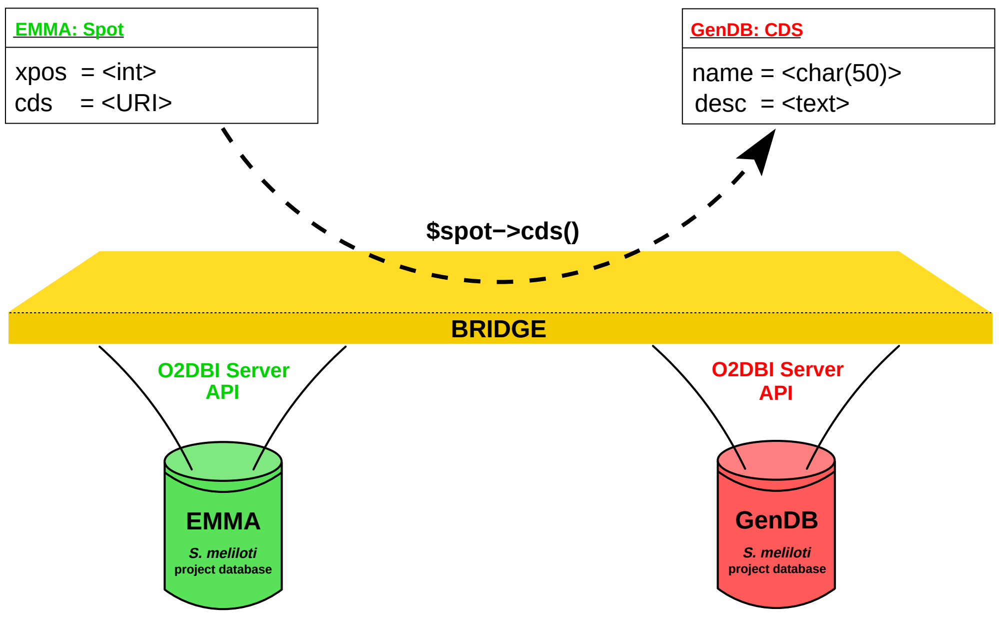


Figure 8.7.: A sample application of the BRIDGE layer: a *CDS* region object referenced by a *Spot* is initialized upon request and returned to the application.

For providing this functionality, this layer needs to communicate with an instance of the *General Project-Management System* that "knows" other projects which contain external objects referenced by the original project data.

The implementation of this layer required the definition of unique identifiers and some extensions of the O2DBI-II system: first of all, objects that should be referenced require a *Unique ID* for identifying them. This number has to be unique for the datasource (e.g. a MySQL database, see also section 8.5) that stores the objects. For obtaining these *Unique IDs*, the O2DBI-II system provides an additional method

```
get_new_uniqueID()
```

that generates a new random *Unique ID*. For the current implementation, random numbers were selected as they are superior to incremental ids and offer more safety: inadvertent misinterpretation of objects can be avoided by picking random numbers from a sufficiently large space (e.g. 2^{64}). While especially small incremental ids would reappear in any other datasource that stores at least as much objects as the original datasource, the attempt to access an object with a wrong random number will generate an error in (almost) all cases.

Classes with objects that can be referenced have an additional flag (`Ext_Referenceable`) for specifying this property and they are called *external referenceable*. Setting this flag also implies that the default O2DBI-II constructor method

```
$class->create(args...)
```

automatically assigns a *Unique ID* after successfully creating a new object. Therefore, the class has an auxiliary private method

```
$class->_create_with_unique_id(uniqueID, args...)
```

that sets the *Unique ID*. Using the standard constructor method thus ensures that each *external referenceable* object obtains a valid *Unique ID*. In addition to this, these classes lack the standard getter/setter method for directly reading or writing the *Unique ID*. The missing setter method for this special attribute thus guarantees that an object keeps its *Unique ID* forever once it has been assigned in the constructor.[1] Instead of a public getter method, objects of these classes have a private method

```
$obj->_get_unique_id()
```

for reading the *Unique ID*. Another new generic method

```
$master->fetchby_unique_id(uniqueID)
```

is also provided by the O2DBI-II for retrieving an object for a given *Unique ID*. Usually, this method is not used directly since the functionality for retrieving "external" objects is encapsulated by the BRIDGE layer (explained later in this chapter).

On the other hand, the O2DBI-II system has been extended by a new common attribute type `Ext_Reference` (*external reference*) that usually contains a reference to an "external

[1]In some cases where additional *Unique IDs* must be assigned for already existing objects (e.g. because the database has been used for some time without *Unique IDs*), these identifiers can only be added by special scripts that directly manipulate the database

object". It can be used for referencing objects of other *external referenceable* classes that already have a *Unique ID*. This attribute contains a URI (Uniform Resource Identifier) for directly accessing the referenced object that is defined as follows:

- Syntax:
  o2xr://<namespace>/<projectname>/<datasourcetype>?uid=<uid>
  where:
- *o2xr* is the name of this schema, similar to `http` or `mailto`. *o2xr* is an abbreviation for *O2DBI eXternal Reference*.
- *namespace* denotes an (worldwide unique) instance of a project management (GPMS). Since we do not want to establish a global registration service for all available instances of the GPMS, we cannot guarantee the uniqueness of the namespace. But using for example the name of an institute where the *General Project-Management System* is installed would ensure a "sufficient uniqueness" (see example below). Nevertheless, it would not be a big problem to resolve such potential conflicts by either changing the name of one instance or by adding a special rule for resolving such conflicts (see implementation of wildcard mechanisms below).
- *projectname* is the name of a project managed by the GPMS given in the namespace.
- *datasourcetype* specifies the type of the datasource that belongs to the project. Since each project can have several datasources (but only a single datasource of each type that has been defined in the GPMS), both, the project name and the datasourcetype are required in the URI.
- *uid* denotes the type of the *Unique ID* (a kind of argument type). Potential other *Unique IDs* that may be added in future versions can thus be identified by using another type.
- The <*uid*> finally contains the *Unique ID* for identifying an individual object.
- For example, the complete URI of a GenDB object stored in a database at the Center for Biotechnology (CeBiTec), Bielefeld University could have the following format:
  o2xr://CeBiTec.Uni-Bielefeld.DE/GenDB_Demo/GENDB?uid=12345

These *o2xr* URIs were specified according to RFC2396[2] where the special characters `; / ? : @ & = + $` are reserved as separators while the symbols `- _ . ! ~ * ' ( )` can be used in addition to letters and numbers. On the other hand, the characters `{ } | \ ^ [ ] '` should not be used at all or only as quoted characters. As an extension to

[2] *http://www.ietf.org/rfc.html*

directly resolving the *o2xr* URIs, a configurable locator can be used to map the pair of (<namespace>, <datasourcetype>) onto a real GPMS instance and a real datasoure type. This mechanism also allows wildcards such as

- (*, <datasourcetype>) → (GPMS instance, datasource type) and
- (<namespace>, *) → ((GPMS instance, *).

For several matching patterns, additional rules have to be defined for resolving the URI (e.g. by defining the order).

It is clear that the external reference attribute is never allowed as a mandatory attribute in the default constructor of any object since the URI is not neccessarily defined for all objects that belong to a class with an external reference.

On the practical side, these URIs link to uniquely identified O2DBI-II objects stored in a datasource of a project managed by a *General Project-Management System*. For implementing the retrieval mechanism, the O2DBI-II system was extended by an additional table that associates a *Unique ID* with its corresponding internal O2DBI-II object:

O2XR

_id	**object_id**	**unique_id**	**_object_class**
1	1	95876632	GENDB::DB::Region::Source::Contig
2	2	83482798	GENDB::DB::Region::CDS
...	...	...	...

Table 8.2.: Additional O2DBI-II table for linking *Unique IDs* to O2DBI-II objects in each database. Since a data model (and the implemented classes) can change over time, the corresponding class of a referenced object is **not** stored in the URI but in the database where uids are associated with O2DBI-II objects.

The class type of a referenced object is also stored in a separate column (_object_class) for obtaining an object of the correct type. If the data model changes (e.g. when the class hierarchy is modified) only this column has to be updated but the URIs remain valid for the referenced object. This table has a unique index on column unique_id for retrieving an object and a combined unique index on column object_id and _object_class for retrieving the *Unique ID*. Upon deletion of an object, the entry in the O2XR table remains but the object is marked as deleted by setting the object_id field to `NULL`.

8.2.2. BridgeFunc

The functionality of the BRIDGE layer itself is currently implemented in a single basic Perl class (*BridgeFunc*) that extends the standard functionality of the O2DBI-II server classes. For reasons of convenience and modularity, this core module is accompanied by three subclasses which provide a number of useful methods for managing *Projects*, *Namespaces*, and *Application_Frames*:

- *BridgeFunc::AppFrames*
  This class can be used for managing *Application_Frames* in the *BridgeFunc* layer. Each specific data source (here, a *ProjectManagement::DataSource* as used by the *General Project-Management System*) is normally accessed by a single specific subclass of an *Application_Frame*, i.e. a GenDB data source is accessed by using its corresponding *Application_Frame::GENDB*. This module keeps track of the associations between *Application_Frame* classes and their corresponding data sources. It also processes the O2DBI-II master modules and overwrites the attribute handlers for classes that contain external references by modifying the Perl symbol table. The API is described in appendix A.5.4.

- *BridgeFunc::Namespaces*
  This module can be used to handle different *Namespaces*. Each individual *Namespace* is normally associated with an unique instance of a GPMS installation. Data from external *Projects* which have their own local GPMS can be accessed by registering the corresponding *Namespace* and its *Application_Frame::GPMS*. Access to external data sources may require separate authorization or at least a guest account. *Projects* can be added or removed and a *Namespace* can be queried for *Projects* via different methods (see appendix A.5.3 for further details).

- *BridgeFunc::Projects*
  This class is used to handle *Projects* that are managed by a *General Project-Management System*. In this *BridgeFunc* context a *Project* is defined as a subclass of *Project-Management::Project*. Basically, this class is used by *BridgeFunc::Namespaces* as a helper module for managing the *Projects* of a *Namespace*. *Projects* can be added or removed and it is possible to retrieve a *Project* object for a given name or *Application_Frame*. The complete API of this class can be found in appendix A.5.2.

In addition to all standard and manually added methods of the O2DBI-II server that can be used in exactly the same way as when using them without a BRIDGE layer, *BridgeFunc* provides the following methods:

- new()
  This default constructor method initializes a new BRIDGE layer object.

- register_AppFrame(<*namespace*>, <*Application_Frame*>)
  An *Application_Frame* object contains information about the user, the O2DBI master objects and some other current configurations (see documentation of the *General Project-Management System* for more details). Upon manual creation of an *Application_Frame*, it can be registered in the BRIDGE layer thus providing the connection to a project's data source.

- register_AppFrame_Type(<*Application_Frame_Type*>, <*DataSource_Type*>)
  Each available GPMS data source is accessed by an individual *Application_Frame*. Since the *BridgeFunc* layer does not have any *a priori* knowledge about requested data sources and their corresponding *Application_Frames* the *Application_Frame_Types* that are required by an application have to be registered initially.

- remove_AppFrame(<*namespace*>, <*projectname*>)
  This method simply removes an already registered *Application_Frame*.

- get_AppFrame(<*namespace*>, <*projectname*>)
  An *Application_Frame* object is returned for a given *namespace* and *projectname*. If no such *Application_Frame* exists, the BRIDGE system will try to initialize an apropriate *Application_Frame* using its current settings.

- get_namespace_project([<*Application_Frame*>, <*O2DBI-II master*>]) This method returns the *namespace* and the *projectname* for a registered *Application_Frame* or an *O2DBI-II master*.

- get_Object(<*URI*>)
  This method tries to resolve a given *URI* and, if possible, the corresponding object is initialized and returned.

- get_URI(*$object*)
  Vice versa this method tries to return the complete corresponding URI for a given object.

The methods for retrieving an object and for getting the URI for an object are added as code references to the O2DBI-II master module for providing this special functionality. But before any external objects can be accessed, the references (URIs) have to be stored. This can be done in a very simple way as illustrated by the following example:

```
...

use BridgeFunc;
use GPMS::Application_Frame::GPMS;
use GPMS::Application_Frame::EMMA;
use GPMS::Application_Frame::GENDB;
```

```
...

# some variables that have to be defined
my ($user, $password, $gendb_project_name, $emma_project_name) = ();

my $gpms = GPMS::Application_Frame::GPMS->new($user, $password);

die "Unable to contact GPMS!" unless (ref $gpms);

my $gendb_AppFrame = GPMS::Application_Frame::GENDB->new($user,
                                                         $password,
                                                         $gpms->gpms_master);
$gendb_AppFrame->project($gendb_project_name);

my $emma_AppFrame = GPMS::Application_Frame::EMMA->new($user,
                                                       $password,
                                                       $gpms->gpms_master);
$emma_AppFrame->project($emma_project_name);

my $bridgefunc = BridgeFunc->new($gpms, 'cebitec.uni-bielefeld.de');
$bridgefunc->register_AppFrame('cebitec.uni-bielefeld.de', $gendb_AppFrame);
$bridgefunc->register_AppFrame('cebitec.uni-bielefeld.de', $emma_AppFrame);

print 'Fetching CDS... ';
my $genes;
foreach my $cds (@{$gendb_AppFrame->application_master->Region->CDS->fetchall}) {
    $genes->{$cds->name} = $cds;
}
print "Done!\n";

foreach my $seq (@{$emma_AppFrame->application_master->Sequence->fetchall}) {
    if (defined ($cds->{$seq->name})) {
        print 'Linking sequence '.$seq->name."\n";
        $seq->GenDB_Region($genes->{$seq->name});
    }
    else {
        print 'Skipping sequence '.$seq->name."\n";
    }
}
```

In a more abstract manner, the procedure implemented above can be described as follows: in the first step the connection to a local GPMS is established and the corresponding *Application_Frame* is initialized. Afterwards, the *Application_Frames* for a GenDB and EMMA project are created. Whenever an *Application_Frame* is registered to the newly created BridgeFunc layer, the getter/setter methods for external reference attributes are overwritten by methods of the BridgeFunc layer. Calling a setter method like

```
$seq->GenDB_Region(...)
```

for an attribute (here GenDB_Region) that contains an external reference with the external referenced object as its argument thus executes

```
BridgeFunc->get_URI()
```

and stores the obtained URI string in the database.

Finally, a sample script application that illustrates the usability of the BRIDGE system is shown in the following source code:

```
...

use BridgeFunc;

use GPMS::Application_Frame::EMMA;
use GPMS::Application_Frame::GENDB;
use GPMS::Application_Frame::GPMS;

...

# some variables that have to be defined
my ($user, $password, $emma_project_name) = ();

# initialize a connection to the local Project Management System
my $gpms_appframe = GPMS::Application_Frame::GPMS->new($user, $password);

die "Unable to contact GPMS!" unless (ref $gpms_appframe);

# initialize an Application_Frame for the current project
my $emma_appframe = GPMS::Application_Frame::EMMA->new($user,
                                                       $password,
                                                       $gpms_appframe->gpms_master);

# try to initialize a project for the given name
$emma_appframe->project($emma_project_name);

# initialize the BRIDGE layer
my $bridgefunc = BridgeFunc->new($gpms_appframe, ’cebitec.uni-bielefeld.de’);

# register the Application_Frames for the local namespace
$bridgefunc->register_AppFrame(’cebitec.uni-bielefeld.de’, $emma_appframe);
$bridgefunc->register_AppFrame_Type(’GPMS::Application_Frame::GENDB’,
                                    ’GENDB::DB’);

# loop through all EMMA sequences and check if there is a reference to a GenDB region
foreach my $seq (@{$emma_appframe->application_master->Sequence->fetchall}) {
    my $region = $seq->GenDB_Region;
    if (ref $region) {
        print ’Sequence ’.$seq->name().’ is linked to CDS ’.$region->name().’\n’;



        my $annotation = $region->latest_annotation->function();
        # check if we have a latest annotation and a CDS
        if (ref $annotation && $region->isa(’GENDB::DB::Region::CDS’) {
            print ’Gene name: ’.$annotation->name()
                .’Gene product: ’.$annotation->geneproduct()
                .’EC: ’.$annotation->EC_number();
        }
    }
    else {
        print ’Sequence ’.$seq->name.’ is not linked to a GenDB region.\n’;
    }
}
```

In this example, only the *Application_Frames* for the *General Project-Management System* and the EMMA project are initialized. Instead of registering an *Application_Frame* for GenDB, the corresponding *Application_Frame_Type* for GenDB projects is registered. Calling

```
$seq->GenDB_Region()
```

thus reads the URI but instead of returning a string, the overloaded method

```
get_object()
```

of the BridgeFunc layer is executed and the corresponding object is initialized and returned.

As another useful extension, a versioning system has been introduced for the O2DBI-II server classes and the corresponding database. Since the use of external references increases the danger of inconsistencies between different O2DBI-II versions, this feature can help to detect and handle such conflicts.

8.3. BRIDGE GUI

The design of the BRIDGE platform encourages using a plug-in architecture for the integration of specialized components into a common graphical user frontend. Similar to many modern applications, these plug-ins can be embedded into a main standard graphical user interface that provides more general functionality. It is common for modern graphical user interfaces to have widely used features such as menus for accessing different functions, a status bar where messages are displayed, a progress bar to indicate running processes, interfaces for changing user settings (options), and of course some kind of a help sytem for assisting the user (see figure 8.8 for an example).

Normally, menus, bars, options, etc. are globally defined and implemented in the toplevel window of an application. But for a plug-in architecture, a more flexible approach is required. The main application framework also has to provide a concept and mechanisms for the integration and communication of its embedded specialized components. For instance, a plug-in should be able to indicate that a task is in progress and thus it may be useful to inform the user and all other modules of the application about this task. This could also be indicated by globally changing the shape of the cursor until the task is finished. Since the user should always be able to abort time-consuming actions in order to regain full control over the application, a special *Cancel* button could be provided for interrupting such tasks.

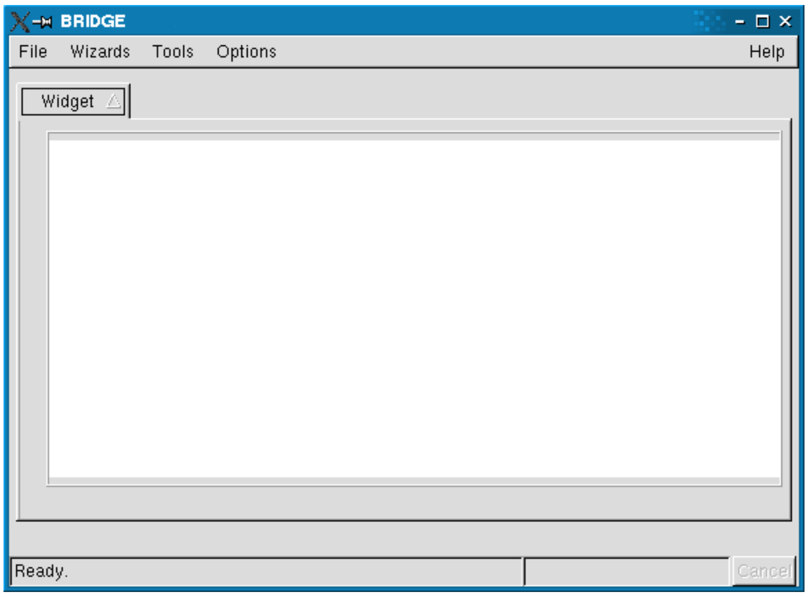


Figure 8.8.: This screenshot of a standard Gtk GUI shows some common features of modern graphical user interfaces: a menu bar with different menus, a status bar, a progress bar, and access to a help system.

In order to facilitate the implementation of this functionality, a common framework was developed which provides a number of specialized Gtk widgets. All modules described in the following sections are not part of one or another specialized component (e.g. GenDB or EMMA) since they provide more general features. Instead, they are maintained independently so that they can be used as well by other applications that are not part of the BRIDGE system.

8.3.1. StatusWidget

The *StatusWidget* was designed as a basic container for all user interfaces that require the functionality described above. Since a *StatusWidget* is a subclass of a Gtk::VBox, derived widgets of this class can be nested and packed into each other. By registering a subwidget, all nested *StatusWidgets* inherit the same signals[3] which can then be connected to a uniform callback (usually a simple subroutine) in the main application. For example, a global status bar and a progress bar that are provided by the main application can be employed by all *StatusWidgets*. Signals that are emitted by a nested *StatusWidget* are passed back through all parent *StatusWidgets* until they are received by a toplevel widget which handles that

[3]See *http://www.gtk.org/* for details about signals and callbacks.

signal. Thus, a *StatusWidget* knows nothing about its current context, it simply emits one of its signals and the main application has to provide the desired behavior. The current implementation of the *StatusWidget* provides the following signals:

- message – send a message, e.g. for putting a text onto the status bar
- init_progress – initialize a time consuming process, e.g. initialize the progress bar
- update_progress – update the status of the current process
- end_progress – stop the current process, e.g. stop progress bar and reset it
- change_cursor – request a change of the cursor for all windows of the application

These signals are inherited by all instances of a *StatusWidget* and emitted recursively until they are caught by the toplevel window. In addition to this, the *StatusWidget* has a special method for interrupting lengthy processes that were initiated by the *init progress* signal. Before updating the progress bar, it is checked whether the user has requested to cancel the current process. Additional signals for derived widgets can be defined by adding them in a special Gtk subroutine:

```
sub GTK_CLASS_INIT {
    my ($class) = @_;

    # define some additional individual signals
    my %signals = ('region_selected' => ['first', 'void', 'gint'],
                   'scrolled'        => ['first', 'void', 'gint', 'gint'],
                   'region_marked'   => ['first', 'void', 'gint', 'gint', 'GtkString']);

    # add the signals
    $class->add_signals(%signals);
}
```

In addition to such signals, a subclass of *StatusWidget* can also provide its own menus and menu items which are then displayed in the menu bar of the main window:

```
sub get_menu {
    my ($self) = @_;

    # define the menu entries
    my @menu = ({'path' => "/Options/Show tooltips",
                 'type' => '<ToggleItem>',
                 'accelerator' => '<Control>t',
                 'callback' => sub { $self->_toggle_tooltips($_[0]->active) }
                });

    # get the menus for all child widgets of the main StatusWidget
    my @child_menus = $self->SUPER::get_menu;
    # add them to the menu
    push(@menu, @child_menus);

    return @menu;
}
```

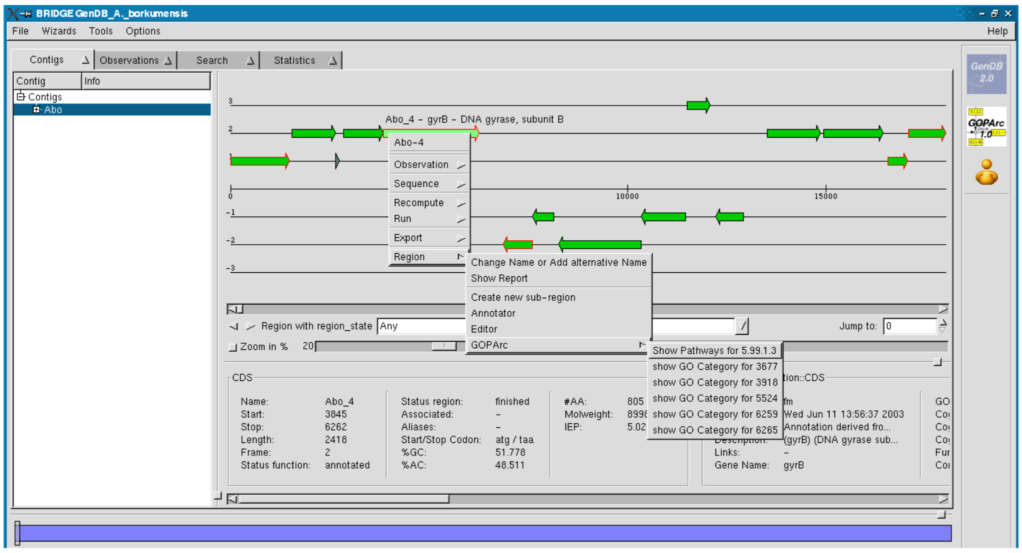
Furthermore, tooltips can be added to each single widget of a *StatusWidget* that get their help messages from a special help repository of the application. All available methods of a *StatusWidget* can be found in appendix A.5.5.

8.3.2. MenuCreator

The class *MenuCreator* was implemented to facilitate a more flexible and dynamic use of Gtk menu bars. In general, Gtk provides two different ways for constructing menu bars, either by creating the menu bar, menus, and all menu items via their standard constructor methods or by simply describing a *Gtk::ItemFactory* (see `get_menu` method above). Thus the latter method provides an ideal way for describing individual components of a menu bar in their corresponding separate components instead of defining the complete menu bar statically in a global main window. Since a standard Gtk menu bar cannot be modified after its creation using the *Gtk::ItemFactory*, the *MenuCreator* was implemented to provide this functionality. It simply reconstructs the menu bar for a given *Gtk::ItemFactory* description including all accelerators (shortcuts for special functions like CTRL-S), removes the old menu bar, and replaces it by the newly constructed menu bar. See appendix A.5.6 for a complete description of the API for the class *MenuCreator*.

8.3.3. ContextMenuInterface

The *ContextMenuInterface* has been developed as a framework for building context sensitive menus which adapt their menu items according to the object that they were opened for (see figure 8.9). A *ContextMenuInterface* is a simple interface that provides some basic functionality for these special types of menus. All modules derived from this interface have to implement the method `_open_menu`.



Name:	Abo_4	Status region:	finished	#AA:	805		fm
Start:	3845	Associated:	-	Molweight:	8998		Wed Jun 11 13:56:37 2003
Stop:	6262	Aliases:	-	IEP:	5.02		Annotation derived fro...
Length:	2418	Start/Stop Codon:	atg / taa			Description:	(gyrB) (DNA gyrase sub...
Frame:	2	%GC:	51.778			Links:	-
Status function:	annotated	%AC:	48.511			Gene Name:	gyrB

Figure 8.9.: Context sensitive menus can be opened by clicking with the right mouse button onto objects such as regions, observations, etc. in the GenDB frontend. In this example, the GOPArc module has also registered some additional external methods that are now available via the menu items at the bottom of the context menu that was opened for one of the regions.

In addition to those methods in the menu that are provided directly in the module that created the object, other "external" modules can add more menu items depending on the context (i.e. object) that was selected by the user (see also section 8.3.8). The API for the *ContextMenuInterface* can be found in appendix A.5.7.

8.3.4. PopoutBook

A special widget that was introduced for the implementation of the BRIDGE system is the *PopoutBook*. It is an extended version of the standard *Gtk::Notebook* since all pages contained in this notebook can be "popped-out" into separate windows by clicking on an arrow button on top of each folder (see figure 8.10). Closing such a window will thus reintegrate the page into its parent notebook.

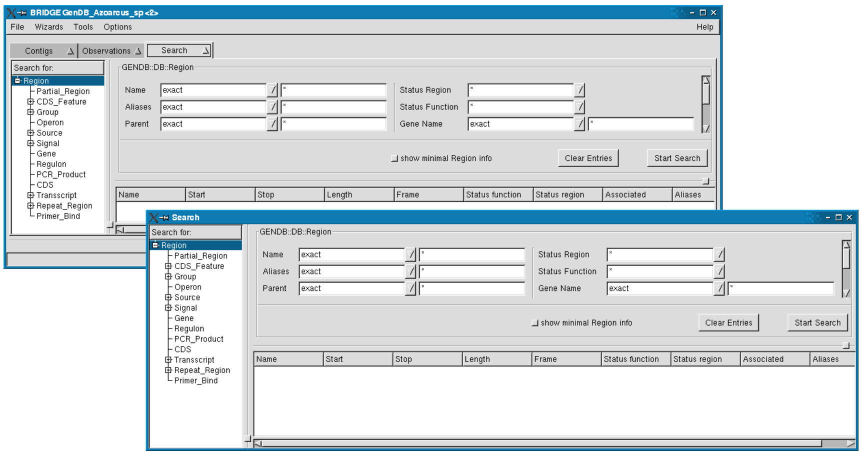


Figure 8.10.: A PopoutBook widget can be can switched out of the notebook in order to be displayed in a new separate window. Closing the window will put the widget back into the PopoutBook.

Using this kind of widget for complex applications allows a most flexible layout of frequently used GUI elements that are well suited to the individual needs of a user and the tasks that have to be performed. For easy navigation through the pages of a notebook, a special optional menu showing all available pages can be opened by clicking on a folder. Detailed information about all methods of a *PopoutBook* are in appendix A.5.8.

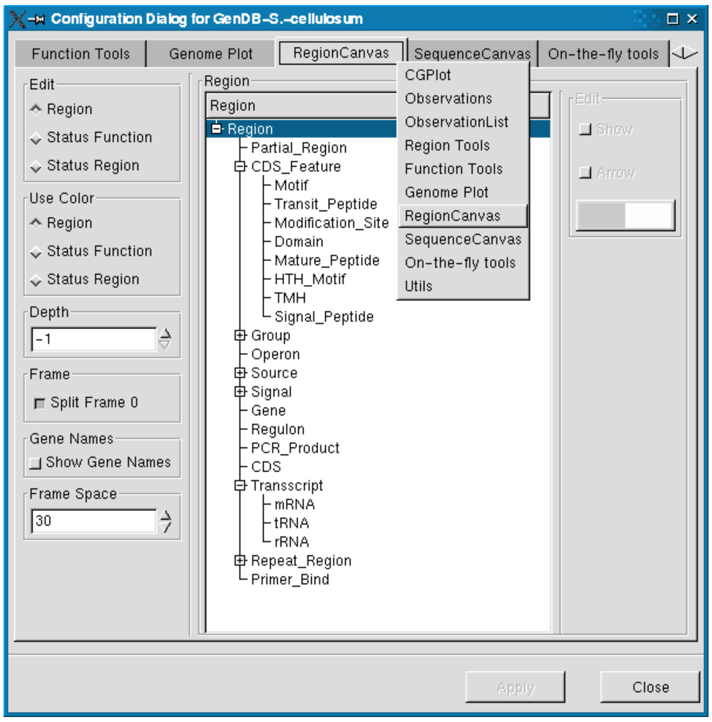
8.3.5. ConfigurationInterface

In most graphical user applications, the user can change a number of settings which are also stored after closing the GUI. A reasonable approach for larger applications is to group these configurations into sections that correspond to specific parts of the frontend so that the user can easily understand which features will be affected upon changes in the configuration. Furthermore, each plug-in component should be able to register its own configuration section and integrate it into a common configuration dialog.

The *ConfigurationInterface* provides a general framework for implementing configuration frontends. It can be used to define GUI widgets for editing configurable attributes. Widgets for configurable attributes are observed so that changes are registered and returned. Changes are also propagated to all *StatusWidgets* that are registered for a *ConfigurationInterface* so that the affected elements in a GUI can be updated according to the new settings. The API for the class *ConfigurationInterface* is described in appendix A.5.9.

8.3.6. ConfigurationDialog

The *ConfigurationDialog* is a simple dialog window for managing *ConfigurationInterfaces*. Each *ConfiguratioInterface* is packed into a separate page of a *Gtk::Notebook* as illustrated in figure 8.11.



Depth: -1
Split Frame 0
Show Gene Names
Frame Space: 30

Figure 8.11.: This screenshot of the GenDB *ConfigurationDialog* shows the *ConfigurationInterface* that can be used to change a number of settings for the visualization of all kinds of regions. Other *ConfigurationInterfaces* can be accessed by clicking on the notebook pages or via a popup menu that contains all available configuration sections. All *ConfigurationInterfaces* share the same buttons for accepting or discarding new settings.

When the user modifies the settings of a configurable attribute, the *ConfigurationDialog* is informed about these changes in the current *ConfigurationInterface* (a single page or section of the notebook). After accepting a new configuration, the settings are stored and propagated to all registered *StatusWidgets* of a *ConfigurationInterface* where they can be applied to all affected GUI elements. All methods for the class *ConfigurationDialog* can be found in appendix A.5.10.

8.3.7. Communication interfaces

While the *General Project-Management System* can be used to control all data access and allows to define a meta project that consists of several (different) subprojects, there is still the need for some methods that connect distinct projects and allow integrated data access among different components (e.g. display a CDS in GenDB that corresponds to a spot selected in EMMA). This problem has been solved by creating so called "Communication Interfaces" that connect two components.

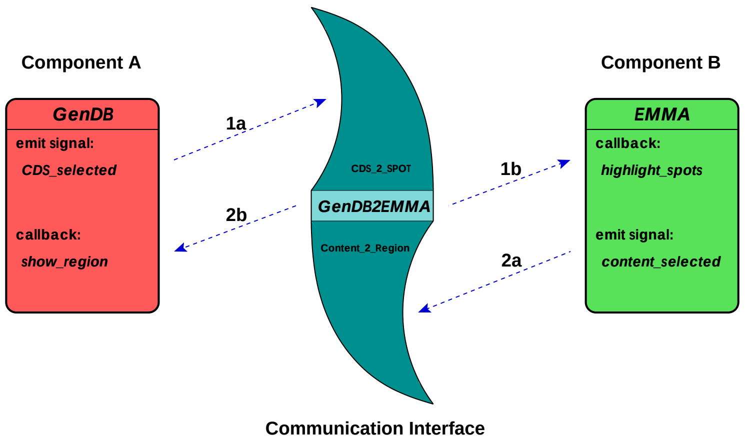


Figure 8.12.: Communication between different BRIDGE components. As an example, the "*GENDB2EMMA*" interface connects a signal ("*CDS_selected*", 1a) of the GenDB system with a callback ("*highlight_spots*", 1b) in EMMA: spots on a slide that correspond to annotated CDS regions will be highlighted upon selection of a region in GenDB. In the reverse direction, a region is displayed in GenDB when the user selects an entry of the sequence library in EMMA (2a and 2b).

As displayed in figure 8.12, such a module always connects two classes (e.g. GenDB and EMMA or two GenDB components) by receiving signals from a sender and redirecting a request to callbacks in the corresponding component. Technically, these *Communication Interfaces* are implemented as *ContextMenuInterfaces* that add some external menu items to object specific popup menus. These modules implement the method `get_menu` that returns menu definitions as *Gtk::ItemFactory* objects which are then added dynamically to the standard menus. This approach is also flexible enough for more complex operations that allow, for example, to open a window in the KEGG browser that displays some expression profile of the EMMA system. Furthermore, each module can add its own entries into the menus of the menu bar in the main application (here: the main BRIDGE frontend).

8.3.8. Putting it all together

The main BRIDGE application has been implemented as a framework that can load one or more of the specialized components dynamically. Figure 8.13 illustrates how the different BRIDGE-GUI modules are integrated into this framework.

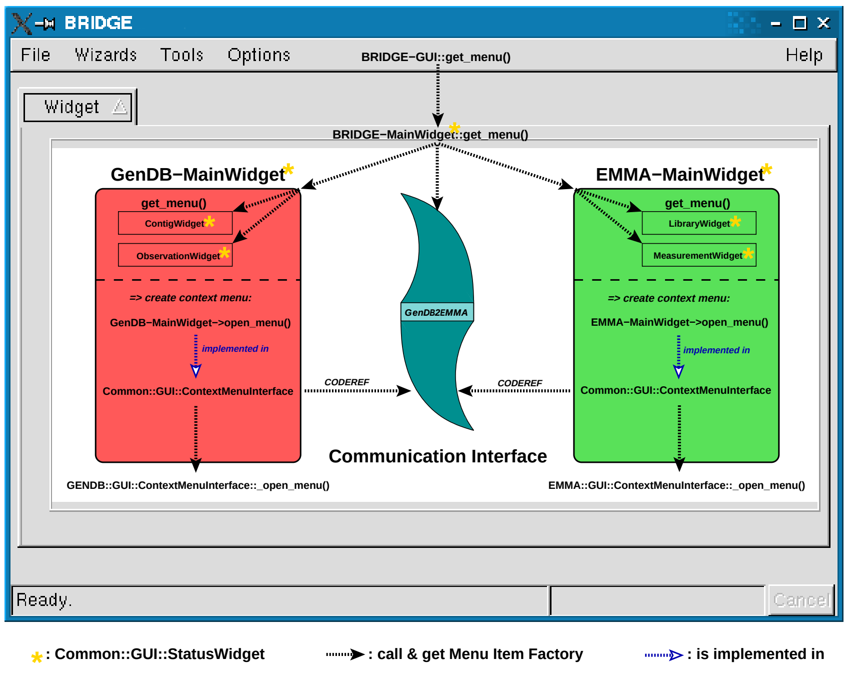


Figure 8.13.: The BRIDGE application basically contains a special BRIDGE *StatusWidget* that loads all other sub modules. The menu bar is constructed recursively by retrieving *Gtk::ItemFactories* from each loaded component. Dynamic context menu interfaces are constructed with entries from the *MainWidgets* and external methods added by the *CommunicationInterfaces*.

The functionality of the different modules and the different initialization steps can be described as follows:

- The main program initializes the application and after selecting a project, the required components are loaded dynamically. The system will also try to establish all requested database connections and register the corresponding *Application_Frames*.
- All loaded components (e.g. one or more *GenDB-MainWidgets* or *EMMA-MainWidgets*) are integrated into the *BRIDGE-MainWidget*.

- The main menu bar is constructed by calling the `get_menu` method. This method call is propagated recursively through all *StatusWidgets*. Thereby, each *StatusWidget* can add its own menus which have to be defined using the *Gtk::ItemFactory*.
- Depending on the loaded components, different *CommunicationInterfaces* are initialized and loaded as well and the static method `add_extern_menu_creator` is executed in order to register additional menus which are implemented in each loaded *CommunicationInterface*. This is done by adding a simple code reference which refers to a subroutine that constructs external menus.
- Whenever the user requests a context sensitive menu (e.g. by clicking on a specific object with the right mouse button), the corresponding *ContextMenu* is constructed. This is done by calling the *open_menu* method of a widget that has been derived from the super class *Common::GUI::ContextMenuInterface* (e.g. the *GenDB-MainWidget*).
- Since the `open_menu` method is only implemented in the *Common::GUI::ContextMenuInterface* module, this method call is propagated to this common module and only executed there.
- The `open_menu` method of the *Common::GUI::ContextMenuInterface* calls the method `_open_menu` implemented in the individual context menu interfaces (e.g. in the *GENDB::GUI::ContextMenuInterface*). Similar to the `get_menu` calls, these methods return a *Gtk::ItemFactory*.
- Finally, the *Common::GUI::ContextMenuInterface* executes all external code references that were added when the *CommunicationInterfaces* were loaded. These code references refer to subroutines implemented in individual *CommunicationInterfaces* that return an addtional *Gtk::ItemFactory* depending on the type of the current object.

8.3.9. InterfaceCreator

In addition to the core modules for the BRIDGE GUI, the *InterfaceCreator* provides a simplified API for rapid prototyping of graphical user interfaces. It can be used to quickly implement simple applications and it ensures a consistent creation of homogeneous dialogs that share the same look & feel. Besides standard widgets like buttons, text entries, and lists, it features more complex interfaces such as a file, font, or color selection dialog. By simply defining a hash of widget elements, the complete user interface can be constructed and visualized without knowing the details of widget arrangements in Gtk. User input values and results can be obtained by simply calling the method `get_result`. The following example shows the hash definition for some simple widgets:

```
use Gtk;
use GUI::InterfaceCreator;

init Gtk;

my $description = [
                   {
                       type       => 'string',
                       name       => 'Password:',
                       default    => 'mypass',
                       input_type => 0,
                       max        => 10,
                       editable   => 1
                   },
                   {
                       type       => 'float',
                       name       => 'Float:',
                       default    => 1.2345,
                       min        => -1.1111,
                       max        => 1.9999,
                       digits     => 4
                   },
                   {
                       type       => 'file',
                       name       => 'File:',
                       default    => $ENV{HOME}."/file.txt"
                   },
                   {
                       type       => 'separator',
                       name       => 'separator'
                   },
                   {
                       type       => 'text',
                       name       => 'Text:',
                       default    => "A multi-line\ntext entry!",
                       font       => "-bitstream-courier-*-r-*-*-*-*-*-*-*-*-*-*",
                       width      => 400,
                       height     => 100
                   }
                   ];

my $InterfaceCreator = new GUI::InterfaceCreator;
my $widget = $InterfaceCreator->make_interface($description);

my $window = new Gtk::Window('toplevel');
$window->add($widget);

$window->show_all;

Gtk->main_iteration while ($window->visible);

my $result = $InterfaceCreator->get_result;

...

Gtk->exit(0);
```

Executing this small script creates a window containing the widgets described above as shown in figure 8.14.

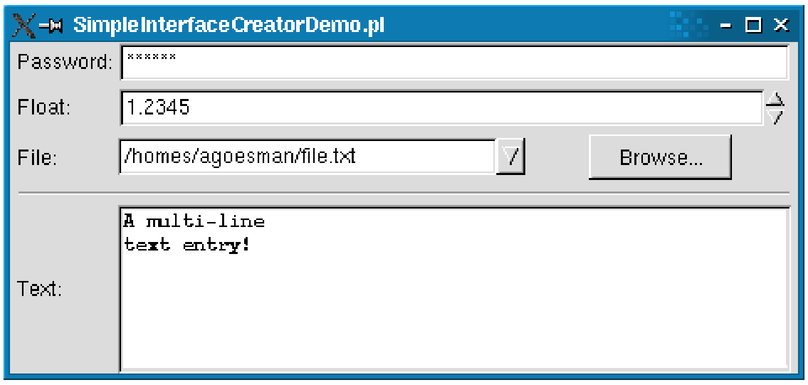


Figure 8.14.: This screenshot shows a simple dialog that was created using the *InterfaceCreator*.

After closing such a dialog, all current values of each widget are stored in a hash that is returned by calling the `get_result` method (the label or the name of the widget has to be used as the key). The complete API of the *InterfaceCreator* can be found in appendix A.5.11.

CHAPTER 9

Applications

In this chapter selected topics and some successful applications of the developed tools will be described. The first section illustrates a simple script that uses the BRIDGE platform for finding gene clusters. During the last three years, the GenDB system has been employed as the primary resource for the functional analysis and annotation in a number of genome projects. Therefore, a special automatic annotator was developed that allows reliable and reproducible high-quality function assignments. Recently, the projection of microarray data onto metabolic pathways and gene ontologies has shown some promising results that are shown in the last sections of this chapter.

9.1. Finding gene clusters

In order to illustrate the simplicity and usefulness of the developed BRIDGE platform, the sample algorithm described in chapter 1 as a pseudocode example (see algorithm 1) was implemented as a small Perl script (here slightly simplified version without looking at 10 best homologous sequences). The code shows only one simple solution for finding clusters of co-regulated genes by looking at the expression ratios obtained from a microarray experiment. In this case, genes are considered to be co-regulated whenever their gene product is an enzyme that is involved in a given pathway. Furthermore, genes are only clustered if they are located on the same strand in a given maximal distance. With some basic knowledge and experience using the APIs of the GenDB, EMMA, and GOPArc systems, the implementation of this script can be done within a few minutes.

```
#!/usr/bin/env perl

use BridgeFunc;

use GPMS::Application_Frame::EMMA;
use GPMS::Application_Frame::GENDB;
use GPMS::Application_Frame::GPMS;


use go::Pathway;
use go::Enzyme;

use IO::Handle;
use Getopt::Std;
#
#  this is necessary if the script is started via rsh(1)
#  otherwise you won’t see any output until the first <RETURN>
#
STDOUT->autoflush(1);


our ($opt_u, $opt_p, $opt_e, $opt_x, $opt_m);

getopts(’u:p:e:x:m:’);

my $user = $opt_u;
my $password = $opt_p;
my $emma_project_name = $opt_e;
my $experiment_name = $opt_x;
my $pathway_name = $opt_m;

my $maxGeneDist = 10000;
my $cluster_ctr = 1;

# initialize a connection to the local Project Management System
my $gpms_appframe = GPMS::Application_Frame::GPMS->new($user, $password);

die "Unable to contact GPMS!" unless (ref $gpms_appframe);

# initialize an Application_Frame for the current project
my $emmaAppFrame = GPMS::Application_Frame::EMMA->new($user,
                                                      $password,
                                                      $gpms_appframe->gpms_master);

# try to initialize a project for the given name
$emmaAppFrame->project($emma_project_name);

# initialize the BRIDGE layer
my $bridgefunc = BridgeFunc->new($gpms_appframe, ’cebitec.uni-bielefeld.de’);

# register the Application_Frames for the local namespace
$bridgefunc->register_AppFrame(’cebitec.uni-bielefeld.de’, $emmaAppFrame);
$bridgefunc->register_AppFrame_Type(’GPMS::Application_Frame::GENDB’,
                                    ’GENDB::DB’);
```

```
my $emmaMaster = $emmaAppFrame->application_master();

print "Initializing experiment...\n";
my $experiment = $emmaMaster->Experiment->init_identifier($experiment_name);
if (!ref $experiment) {
    print STDERR "Error: Could not initialize experiment for given experiment name!\n";
    exit 0;
}


print "Initializing pathway...\n";
my $pathway = go::Pathway->init_name($pathway_name);
if (!ref $pathway) {
    print STDERR "Error: Could not initialize pathway for given pathway name!\n";
    exit 0;
}


# fetch all Quantitations for the given experiment
print "Fetching quantitations...\n";
my $quantitations = $experiment->fetchall_Quantitations();


print "Please wait while searching for regulated enzymes...\n";
my @genes = ();
foreach my $q (@$quantitations) {
    my $ratio = ($q->ch2i - $q->ch2bg) / ($q->ch1i - $q->ch1bg);
    if ($ratio > 2) {
        push(@spots, $q->spot);

        my $content = $q->spot->Well->content;
        if (ref $content && $content->isa("EMMA::DB::Content::Sequence")) {
            my $region = $content->sequence->GenDB_Region;
            if (ref $region) {
                my $annotation = $region->latest_annotation_function();
                # check if we have a latest annotation and a CDS
                if (ref $annotation && $region->isa("GENDB::DB::Region::CDS")) {
                    if ($annotation->EC_Number() ne "" ) {
                        if(go::Enzyme->check_pathway_for_ec_number($annotation->EC_Number(),
                                                                   $pathway_name)) {
                            push(@genes, $region);
                        }
                    }
                }
            }
        }
    }
}


print "Checking for clusters in $pathway_name...\n\n";
my $currentPos = 0;
my $currentStrand = 0;
my @ClusterGenes;
foreach my $g (sort {$a->start <=> $b->start} @genes) {
    next if $currentPos == $g->start();
    if ($g->strand() eq $currentStrand) {
        if ($g->start - $currentPos < $maxGeneDist) {
            push(@ClusterGenes, $g);
        }
        else {
            &print_genes(\@ClusterGenes);
            @ClusterGenes = ($g);
        }
    }
    else {
        &print_genes(\@ClusterGenes);
        $currentStrand = $g->strand();
        @ClusterGenes = ($g);
    }
    $currentPos = $g->start;
}


##########################################################################
### subroutine for pretty printing of identified clusters in a pathway ###
##########################################################################
sub print_genes {
    my ($genes_ref) = @_;

    my @genes = @$genes_ref;
    if ($#genes >= 1) {
        my $separator = " -> ";
        $separator = " <- " if $genes[0]->strand() eq "-";
        my $cgs = join($separator, map($_->name.
                                       " ".
                                       $_->latest_annotation_function->name.
                                       " (".
                                       $_->latest_annotation_function->EC_Number.
                                       ")",
                                       @genes));

        print "Cluster $cluster_ctr:\n$cgs\n\n";
        $cluster_ctr++;
    }
}
```

Running this script for a given experiment and pathway produces a simple list of predicted gene clusters. For example, this algorithm was applied for a microarray experiment performed for *C. glutamicum*. A search for gene clusters in the phenylalanine, tyrosine, and tryptophan biosynthesis produced the following output:

```
Cluster 1:
cg0503 aroD (4.2.1.10) -> cg0504 aroE (1.1.1.25)

Cluster 2:
cg1129 aroF (4.1.2.15) -> cg1134 pabAB (4.1.3.-)

Cluster 3:
cg1574 pheS (6.1.1.20) -> cg1575 pheT (6.1.1.20)
```

```
Cluster 4:
cg1827 aroB (4.6.1.3) <- cg1828 aroK (2.7.1.71) <- cg1829 aroC (4.6.1.4)
   <- cg1835 aroE3 (1.1.1.25)
```

Figure 9.1 depicts a linear plot of GenDB-2 for the largest predicted cluster.

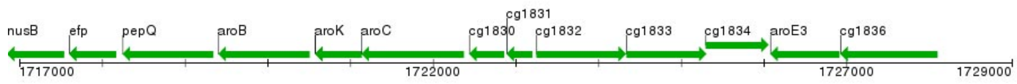


Figure 9.1.: A cluster of 4 genes involved in the phenylalanine, tyrosine, and tryptophan biosynthesis was found to be significantly up-regulated in the sample experiment.

9.2. Annotation of *Mycoplasma mycoides subsp. mycoides SC*

The genome of *Mycoplasma mycoides subsp. mycoides SC* (MmymySC) has been sequenced by the group of Joakim Westberg at the Swedish Royal Institute of Technology in Stockholm to facilitate studies regarding the organism's cell function and the disease it causes. MmymySC[1] is the etiological agent of contagious bovine pleuropneumonia (CBPP), a highly contagious respiratory disease in cattle and buffalo. MmymySC has a circular genome of 1,213,174 bp in size and a very low *GC content* of 27%. Similar to other mycoplasma, the *genetic code 4* (start codons TTA, TTG, CTG for leucine, ATG for methionine, ATT, ATC, ATA for isoleucine, and GTG for valine and TAG and TAA as stop codons) is applied instead of the standard *genetic code 11* used for most prokaryotes. The annotation phase of the genome project was intensively accompanied by the Bioinformatics group at the Center for Genome Research in Bielefeld (installation of GenDB on a laptop for J. Westberg, computation of facts, update of contigs and recomputation of facts, implementation of several special purpose scripts, web frontend for published genome, etc.). The annotation with GenDB revealed that the genome of MmymySC contains a large number of long repetitive sequences (IS elements). The complete sequence of the assembled genome contains 1,060 putative genes (as predicted by Glimmer). For the final annotation and publication of this genome, the BRIDGE system was used to categorize all genes according to their functional classification.

[1] *http://www.biotech.kth.se/molbio/key_achievements/mycoplasma.html*

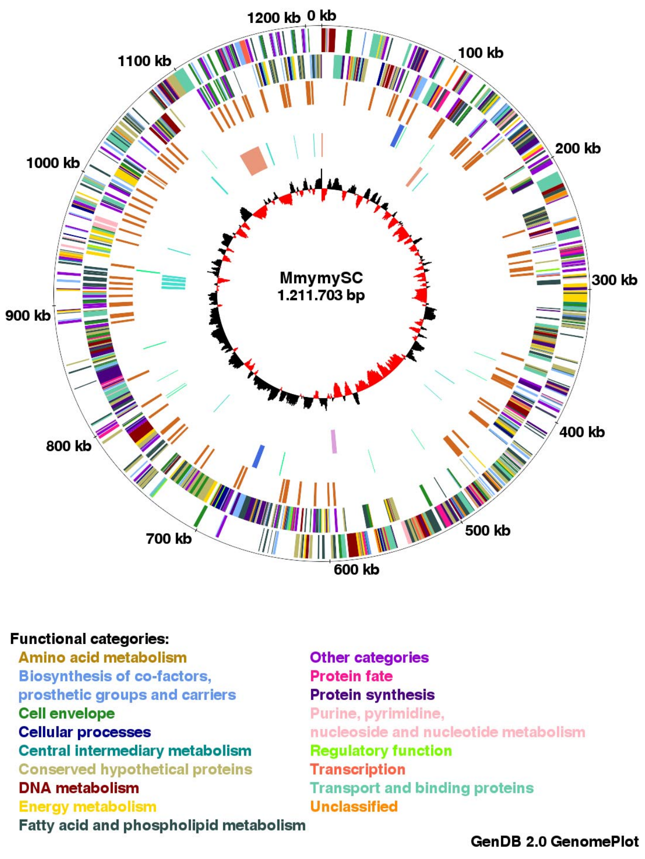


Figure 9.2.: Circular genome plot of *Mycoplasma mycoides subsp. mycoides SC* created with GenDB-2.0. Outer concentric circle: genome positions in bases, where position one is the first base of the *dnaA* gene. Second and third concentric circle: the predicted genes on the leading and lagging strand. Fourth concentric circle: IS-elements. Fifth concentric circle: tRNA and rRNA genes. Sixth concentric circle: the capsule biosynthesis clusters, the hydrogen peroxide biosynthesis cluster, and the genes encoding variable surface proteins. Innermost concentric circle: the *GC skew* (G-C / G+C) plot.

A circular genome plot of the genome – created with GenDB-2.0 as displayed in figure 9.2 – shows all genes characterized into their functional categories. Therefore, a first prototype of the BRIDGE system was implemented for connecting a GenDB and GOPArc module. The original manual annotation created with GenDB-1 provided the functional classification which was then imported into a GenDB-2 project. Different colors were assigned to each category (here: Monica Riley categories) using the GOPArc module, afterwards the plot was created using the newly integrated circular plot feature of GenDB-2.0. Finally, the genome was published in Genome Research [WPH+04] and submitted to EMBL/GenBank/DDBJ under the accession number BX293980.

9.3. Annotation of *Bdellovibrio bacteriovorus*

In another cooperation with the Max-Planck-Institute for Developmental Biology in Tuebingen, the genome of the predatory bacteria *Bdellovibrio bacteriovorus HD100* was annotated with GenDB-2.0 in order to analyze its life cycle. The analysis of the *Bdellovibrio bacteriovorus* genome revealed a size of 3,782,950 bp which is fairly large with respect to the cell dimensions (0.2 to 0.5 μm wide and 0.5 to 2.5 μm long). 3584 proteins were predicted for the complete sequence with an average GC content of 50.7%. As a predatory bacteria, *Bdellovibrio bacteriovorus* attaches specifically to certain other bacteria in order to invade them and consume the host cell from the inside. As illustrated in figure 9.3, *Bdellovibrio* can grow and develop in the periplasm of its prey utilizing the amino acids and other nutrients of the host cell for its own life cycle.

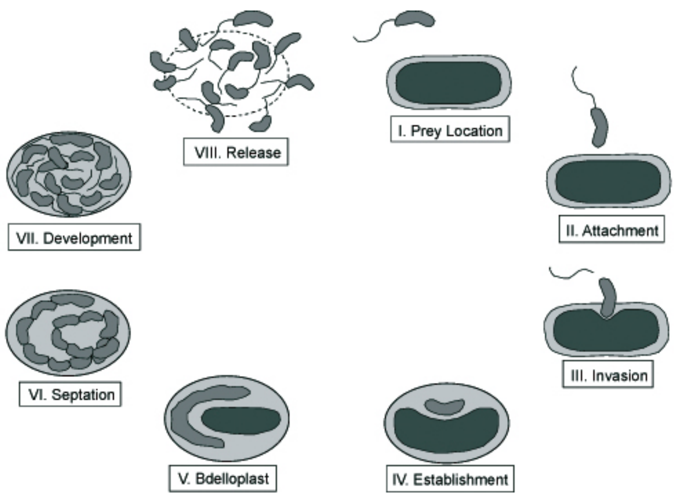


Figure 9.3.: *Bdellovibrio bacteriovorus* has a quite fascinating life cycle: Once it has collided with a prey cell and verified its suitability for invasion *Bdellovibrio* starts to enter the host. After navigating into the periplasmic space between the outer and inner membrane of the prey cell *Bdellovibrio* begins to consume the host cell from the inside by degrading all kinds of biopolymers. When all resources of the prey are exhausted, the bacteria septates and finally, an odd number of progeny cells is released by dissolving the remaining prey cell. (Adapted from [RJR+04])

All predicted coding sequences were analyzed automatically with GenDB-2.0 and more than 2 million observations were computed and stored in the project database. Afterwards, an initial automatic annotation was created based on different tool results (BLAST vs. EMBL, SwissProt and KEGG, HMM searches vs. the TIGRFAM and Pfam databases, InterPro, etc.). Functional categories were derived by blasting each CDS against the COG database and by evaluating the results with the GOPArc module (see figure 9.4). Finally, the automatically assigned functions were verified by a manual annotation.

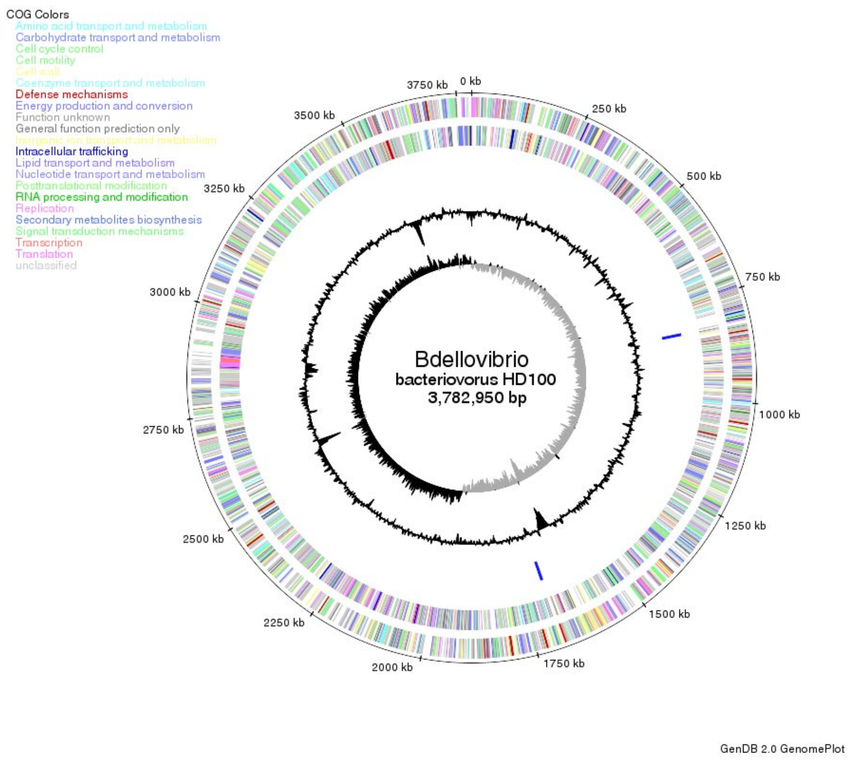

COG Colors
Amino acid transport and metabolism
Carbohydrate transport and metabolism
Cell cycle control
Cell motility
Cell wall
Coenzyme transport and metabolism
Defense mechanisms
Energy production and conversion
Function unknown
General function prediction only
Inorganic ion transport and metabolism
Intracellular trafficking
Lipid transport and metabolism
Nucleotide transport and metabolism
Posttranslational modification
RNA processing and modification
Replication
Secondary metabolites biosynthesis
Signal transduction mechanisms
Transcription
Translation
unclassified
0 kb
250 kb
500 kb
750 kb
1000 kb
1250 kb
1500 kb
1750 kb
2000 kb
2250 kb
2500 kb
2750 kb
3000 kb
3250 kb
3500 kb
3750 kb
Bdellovibrio
bacteriovorus HD100
3,782,950 bp
GenDB 2.0 GenomePlot

Figure 9.4.: The circular genome plot of *Bdellovibrio bacteriovorus HD100* was created using GenDB-2.0 and GOPArc via BRIDGE. Outer concentric circle: genome positions in bases, where position one is the first base of the *dnaA* gene. Second and third concentric circle: the predicted genes on the leading and lagging strand colored by COG category. Fourth concentric circle: rRNA genes. Fifth concentric circle: the GC content. Innermost circle: the *GC skew* (G-C / G+C) plot.

As described in more detail in the corresponding Science publication [RJR+04], the automatic and manual annnotation revealed that *Bdellovibrio* lacks the biosynthesis pathways for some essential amino acids. Instead, it utilizes the chemical compounds of its prey which is indicated by a large number and broad range of transport systems. Furthermore, a huge contingent of lytic enzymes (numbering over 200 genes) was found which is essential for invading the host cell, degrading biopolymers, and for finally dissolving the prey cell. Future anti-microbial strategies aim at using *Bdellovibrio* as a “living antibiotic” since it is not capable of infecting eukaryotic cells.

9.4. Analysis of 5 microbial genomes

On June 1st 2001, the BMB+F funded network for *Genome Research on Bacteria Relevant for Agriculture, Environment and Biotechnology*[2] settled at Bielefeld University started its work with the main goal to develop this new research field and to contribute important results to biotechnology. Since one major goal of the network’s research was to establish the nucleotide sequences of six bacterial genomes (37 Megabases in total, see table 9.1), the GenDB system was chosen as the platform for the annotation and all further downstream analysis of these genomes.

Bacterium	Genome size (Mb)
Azoarcus sp.	$\sim$ 4.6
Clavibacter michiganensis subsp. michiganensis	$\sim$ 3.5
Xanthomonas campestris pv. campestris	$\sim$ 5.5
Xanthomonas campestris pv. vesicatoria	$\sim$ 5.5
Alcanivorax borkumensis	$\sim$ 3.2
Sorangium cellulosum	$\sim$ 12.2
Streptomyces cosmids	$\sim$ 2.1
Total	$\sim$ **35**

Table 9.1.: Genome projects of the Bielefeld network for *Genome Research on Bacteria Relevant for Agriculture, Environment and Biotechnology*. Altogether, the network is working on the assembly and annotation of more than 35 million basepairs, i.e. approximately 35,000 genes.

The network comprises the areas “Agriculture”, “Environment”, and “Biotechnology”. For the area “Agriculture”, the endophyte *Azoarcus* sp. is analyzed as a nitrogen fixing bacteria and compared to *Sinorhizobium meliloti* and *Bradyrhizobium japonicum* that are both capable of fixing nitrogen in symbiotic root nodules. Understanding and exploiting this potential by comparative genomics is the key objective in order to reduce nitrogen fertilization of crops such as rice or soybeans by biological nitrogen fixation. Furthermore, the

[2] *http://www.GenoMik.Uni-Bielefeld.DE/*

plant-pathogenic bacteria *Clavibacter michiganensis* subsp. michiganensis, *Xanthomonas campestris* pv. campestris, and *Xanthomonas campestris* pv. vesicatoria are in the center of the network's interest since these organisms are responsible for worldwide multi-billion dollar crop yield losses each year. The information gathered in these genome projects is supposed to contribute to the design of environmentally-friendly agrochemicals for controlling these pests.

Within the area "Environment", *Alcanivorax borkumensis* has been sequenced since this organism has a special feature in that it uses crude mineral oil as its sole source for carbon and energy. There is hope that the elucidation of its metabolic potential will make a major contribution towards the design of strains capable for cleaning-up oil contaminated sites.

In the area "Biotechnology", the myxobacterium *Sorangium cellulosum* is analyzed because of its capability to produce low-molecular weight compounds such as chivosazoles and etnangiens with remarkable biological activities (secondary metabolites). In addition to the functional analyis of more than 10,000 expected genes, the identification of new drug candidates with anticancer, antibacterial, fungicidal, or immune-modulating effects is a major goal in this project. Furthermore, the DNA sequence of cosmids that carry biosynthetic gene clusters of *Streptomyces* is analyzed in order to identify new antibiotic synthesis pathways.

While the annotation of the *A. borkumensis* genome is already finished, *Azoarcus* and *X. campestris* pv. vesicatoria are currently being annotated. At the time of this writing, the sequences of *C. michiganensis* and *S. cellulosum* are polished in order to obtain a high quality sequence of the genome.

Since the manual annotation is a very time-consuming work for the analysis of a genome, a specialized meta annotator (*Metanor*) was implemented that automatically assigns a gene function based on a sophisticated combination of different tool results:

- *BLAST2p vs. KEGG*
  Since the KEGG database represents a resource of all annotated genomes that are stored in a unified and consistent way, the functional descriptions of the genes contained therein can be used to identify a gene name, gene product, and an EC number for enzymes. Therefore, each CDS is blasted against all genes of the organisms contained in KEGG on the amino acid level. *Metanor* can be configured to use the *n* best hits vs. the KEGG database in order to find the most frequently assigned gene name, gene product, and EC number.

- *BLAST2p vs. nearest neighbor*
  When the genome under investigation is closely related to another genome that was already annotated, *Metanor* can be configured to use the annotation of the latter for the function assignment. Therefore, all predicted genes are blasted on the amino acid level vs. all genes of the nearest neighbor genome. The gene name, gene product, and the description of the annotated homologous CDS are used, if the level of the best

observation is better than a specified threshold, e.g. better than level 3. Previously assigned gene names or gene products derived from KEGG are overwritten.

- *PSI-BLAST SwissProt*
  The SwissProt database represents a manually curated high-quality repository of comprehensively annotated amino acid sequences. Thus, the information of this database can be used for accurate and highly specific function assignments. Therefore, PSI-BLAST is used to identify the most specific homologous entry in SwissProt. Again, this hit is only used if it is better than a specified threshold; the description of the hit is then added to the description of the annotation. *Metanor* also tries to extract the detailed functional description that is often available for SwissProt entries. Based on the level of the best SwissProt hit, a confidence level (1 – 6) is assigned ranging from "High confidence in function and specificity" via "Specificity unclear", "Function unclear", "Family membership", and "Conserved hypothetical protein" down to "Hypothetical protein".

- *InterPro*
  In the third step of the automatic annotation, *Metanor* extracts a unique list of GO numbers from the best InterPro observation. These GO numbers and their corresponding descriptions are added to the annotation.

- *PSI-BLAST COG*
  Similar to the SwissProt hit, the best observation computed by a PSI-BLAST run vs. the COG database is used for the annotation if the level is above the specified threshold. This hit is used to assign a functional classification (a COG number, a COG category, and a COG category ID) to each CDS. If no hit was found above the given threshold, the default COG category (COG0000) for an unclassfied protein is used.

- *TIGRFAM*
  The HMM TIGRFAM database is used to find specific domains or motifs that characterize a CDS. The description of the best hit is added to the description of the annotation. If none of the other tools used so far produced a significant observation above the specified threshold, this tool can be used to identify at least some motif that probably contains hints about the function of a CDS. If the previously assigned confidence level was worse than level 4 ("Family membership"), a significant hit vs. the TIGRFAM database is used to assign the latter level, otherwise the previous level is kept.

In general, all observations used by *Metanor* for the automatic annotation are added to a list of observations stored with the annotation as supporting evidence. By looking at this list, a manual annotator or scientist can always understand how the automatic annotation was derived. Before writing a new annotation, the maximal level of all observations is checked.

If a minimal required level was specified for the current *Metanor* run, the extracted information is only stored if the maximal level exceeds the minimal level. Otherwise, only a default annotation is created with most fields remaining empty and the CDS is described as a “hypothetical protein predicted by Glimmer/Critica”. For reasons of convenience, *Metanor* can be configured to delete or keep old *Metanor* annotations and it can be selected whether the latest annotation function should be set. The latter feature is especially useful, if manual annotations should be kept as the current annotation that is presented to the user and e.g. exported to an EMBL file.

As an example, *Metanor* was able to annotate more than 90% percent of all *A. borkumensis* genes in a way that the human annotators could simply confirm the automatic result or just needed to modify only a few details. In particular, such automatic annotation strategies will be very useful for the annotation of the largest bacterial genome known to date, *Sorangium cellulosum*, which is supposed to have about 10,000 genes.

9.5. Postgenome analysis

In addition to the genome projects described above, the Bielefeld Center for Genome Research is also focused on postgenome analyses, i.e. transcriptomics and proteomics. The following sections illustrate two examples for successful applications of the BRIDGE platform in this area of research.

9.5.1. Genome comparison of *Corynebacterium glutamicum* and *Streptomyces coelicolor*

The manual annotation of *C. glutamicum* was started in 2001 using the GenDB-1 system. Thus, a special script was implemented for migrating a GenDB-1 project to GenDB-2.0. The final annotation of the genome was recently published in [KBB$^+$03b] and submitted to EMBL/GenBank/DDBJ.

For the methionine biosynthesis and other metabolic pathways, the BRIDGE system was used as a tool for genome comparison. For example, two GenDB projects were created (for *C. glutamicum* [THM$^+$02] and *S. coelicolor* [BCCT$^+$02]) and all annotated enzymes were mapped automatically via their corresponding EC numbers onto the KEGG metabolic pathways in the GOPArc system.

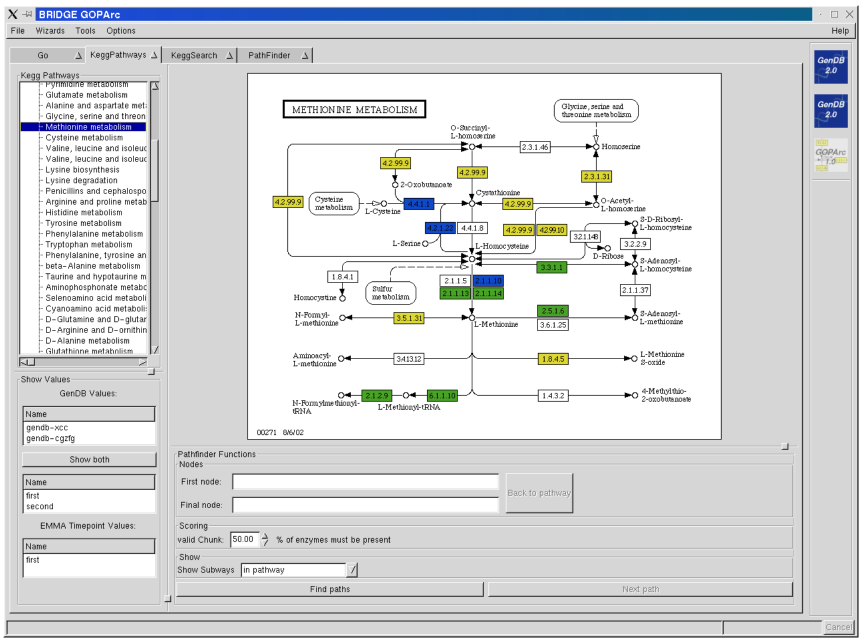
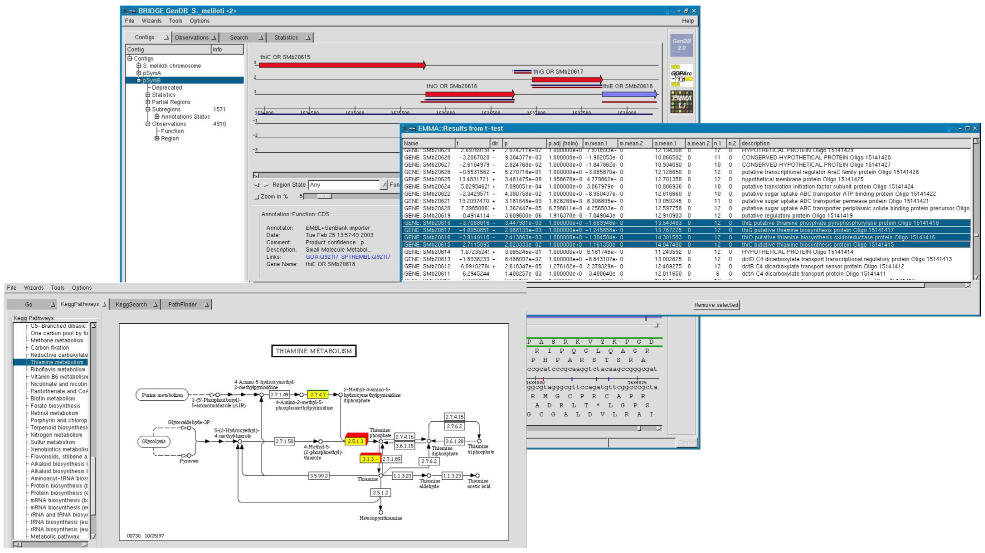


Figure 9.5.: The methionine biosynthesis pathway as derived from the annotations stored in the GenDB system. EC numbers shown in yellow have been found for *C. glutamicum*, enzymes show in blue have been annotated for *S. coelicolor* and EC numbers displayed in green were found in both genomes.

In figure 9.5 one can see immediately that (starting from the top) the first steps of the L-methionine biosynthesis for *S. coelicolor* differ completely from those of the closely related organism *C. glutamicum* [RPK03]. Experimental results have already shown that *S. coelicolor* is prototrophic for L-methionine, thus leading to the conclusion that *S. coelicolor* may produce L-homocysteine from L-cysteine and L-homoserine. The integration of different specialized components (here two GenDB modules and the GOPArc browser) into a common interface showed its usability for a comparative analysis of metabolic pathways in two related organisms.

9.5.2. Expression analysis of *Sinorhizobium meliloti*

As a second example, a GenDB, EMMA, and GOPArc project were integrated for the expression analysis of *S. meliloti* [GFL^{+}01]. Significantly up or down regulated genes that were identified using the t-test statistics wizard in EMMA were mapped onto the annotated genes and the KEGG metabolic pathways.



Name	t	dir	p	p.adj (holm)	m.mean.1	m.mean.2	a.mean.1	a.mean.2	n.1	n.2	description
GENE: SMb20629	2.6976919(	+	2.074211e-02	1.000000e+0	7.970593e-	0	12.194308	0	12	0	HYPOTHETICAL PROTEIN Oligo 15141429
GENE: SMb20628	-3.2067028	-	9.384377e-03	1.000000e+0	-1.902053e-	0	10.866582	0	11	0	CONSERVED HYPOTHETICAL PROTEIN Oligo 15141428
GENE: SMb20627	-2.6104979	-	2.824768e-02	1.000000e+0	-1.847862e-	0	10.934090	0	10	0	CONSERVED HYPOTHETICAL PROTEIN Oligo 15141427
GENE: SMb20626	-0.6531562	-	5.270716e-01	1.000000e+0	-3.085870e-	0	12.126650	0	12	0	putative transcriptional regulator AraC family protein Oligo 15141426
GENE: SMb20625	13.4831721	+	3.481475e-08	1.958678e-0	4.779862e-	0	12.701350	0	12	0	hypothetical membrane protein Oligo 15141425
GENE: SMb20624	5.0295462(	+	7.098051e-04	1.000000e+0	3.067979e-	0	10.806936	0	10	0	putative translation initiation factor subunit protein Oligo 15141424
GENE: SMb20622	-2.3429571	-	4.380758e-02	1.000000e+0	-8.950437e-	0	12.616660	0	10	0	putative sugar uptake ABC transporter ATP binding protein Oligo 15141422
GENE: SMb20621	19.2097470	+	3.181648e-09	1.826266e-0	8.306895e-	0	13.059245	0	11	0	putative sugar uptake ABC transporter permease protein Oligo 15141421
GENE: SMb20620	7.3985006(	+	1.362447e-05	6.798611e-0	4.256503e-	0	12.597758	0	12	0	putative sugar uptake ABC transporter periplasmic solute binding protein precursor Oligo
GENE: SMb20619	-8.4914114	-	3.689600e-06	1.916378e-0	-7.849843e-	0	12.910983	0	12	0	putative regulatory protein Oligo 15141419
GENE: SMb20618	-3.7088616	-	3.447981e-03	1.000000e+0	-1.689946e-	0	13.543483	0	12	0	thiE putative thiamine phosphate pyrophosphorylase protein Oligo 15141418
GENE: SMb20617	-4.0050851	-	2.068139e-03	1.000000e+0	-1.245858e-	0	13.767225	0	12	0	thiG putative thiamine biosynthesis protein Oligo 15141417
GENE: SMb20616	-3.9149110	-	2.413863e-03	1.000000e+0	-1.304504e-	0	14.301583	0	12	0	thiO putative thiamine biosynthesis oxidoreductase protein Oligo 15141416
GENE: SMb20615	-2.7115895	-	2.023333e-02	1.000000e+0	-1.161350e-	0	14.847400	0	12	0	thiC putative thiamine biosynthesis protein Oligo 15141415
GENE: SMb20614	1.0723524(	+	3.065245e-01	1.000000e+0	6.161748e-	0	11.243592	0	12	0	HYPOTHETICAL PROTEIN Oligo 15141414
GENE: SMb20613	-1.8936233	-	8.486097e-02	1.000000e+0	-6.843107e-	0	13.002625	0	12	0	dctD C4 dicarboxylate transport transcriptional regulatory protein Oligo 15141413
GENE: SMb20612	6.8910270(	+	2.618347e-05	1.276182e-0	2.379329e-	0	12.469275	0	12	0	dctB C4 dicarboxylate transport sensor protein Oligo 15141412
GENE: SMb20611	-6.2945244	-	1.488257e-03	1.000000e+0	-3.408640e-	0	12.011650	0	6	0	dctA C4 dicarboxylate transport protein Oligo 15141411

BRIDGE GenDB_S_meliloti <2>
File Wizards Tools Options Help
Contigs Observations Search Statistics
thiC OR SMb20615
thiG OR SMb20617
thiO OR SMb20616
thiE OR SMb20618
EMMA::Results from t-test
Remove selected
KeggPathways KeggSearch PathFinder
THIAMINE METABOLISM

Figure 9.6.: Expression analysis of *Sinorhizobium meliloti* on the level of metabolic pathways. Yellow rectangles mark the annotated enzymes of the thiamin biosynthesis pathway and green or red boxes highlight positive and negative expression ratios (normalized M values) respectively that were calculated with the EMMA module (see small screen-shot of t-test result list). Red CDS regions displayed in the GenDB system above the list indicate down regulated genes. The annotated genes of the thiamin biosynthesis pathway are marked on the selected megaplasmid pSymB and gene *thiE* (annotated with EC number 2.5.1.3) is therefore highlighted in blue.

As displayed in figure 9.6, the three highlighted genes with annotated EC numbers of the thiamin biosynthesis pathway are spread over the megaplasmid pSymB [FWW$^+$01]. The last one of these genes (*SMb20618*, *thiE*) is a member of a cluster of four genes (*thiC*, *thiO*, *thiG*, and *thiE*) that have been annotated as putative thiamin biosynthesis proteins. The corresponding microarray analysis of gene expression under phosphate limitation in the wild type strain has shown that all four clustered genes are significantly down regulated between 1.2 and 1.6 fold (mean of normalized logarithmic M values) thus meeting the expectations [KB03]. In this example, the integrated approach supported by the BRIDGE system has simplified the evaluation of expression data and facilitated the analysis of regulatory networks.

9.5.3. Integrated microarray analysis

After implementing the software, a testing and evaluation phase was set up. More than 20 test slides were hybridized and analyzed with EMMA. In doing this we found that the system is easy to use. The design concept also holds as EMMA has proven to be easy to extend by

R-packages. For example, variance stabilization was published during the testing phase and could be immediately integrated into the system.

EMMA was tested and is running stable under three different UNIX/LINUX variants (Solaris 8/9, SuSE Linux 7.2/8.1, and FreeBSD 4.5). Because of the multi-platform capabilities of Perl, EMMA should be easily portable to other operating systems.

The MicroLIMS system provides comfortable and reliable upload facilities for experimental setups and protocols. Its ability to provide a centralized resource for laboratory protocols has proven to be superior to decentralized storage using word processors or paper based forms.

The successful testing of the platform did encourage us to apply the platform in three international projects.

Within the European Union project MEDICAGO[3], comprehensive *Medicago truncatula* Mt6k root interaction transcriptome (Mt6k-RIT) microarrays representing approximately 5,700 genes were hybridized against probes from symbiotic root interactions and evaluated using EMMA [Küs03]. That way, more than 300 genes significantly upregulated in mature root nodules and more than 100 genes significantly upregulated in endomycorrhiza were identified. These sets of genes contain numerous nodule-specific and mycorrhiza-upregulated genes that are well-known from the literature [WPK03].

In another project, Sm6k microarrays containing approximately 6,200 unique open reading frames from *Sinorhizobium meliloti* where produced [RTK$^+$03]. Sample arrays were hybridized with cDNA-probes from cells grown under microaerobic conditions versus aerobic conditions. Differentially expressed genes identified with EMMA were mapped onto replicons and functional categories. This way, a majority of genes overexpressed under microaerobic conditions were found to be located on the pSymA plasmid which is known to contain numerous genes specific to nitrogen and oxygen metabolism [BFJ$^+$01]. Also, a large proportion of regulated genes were assigned to functional class I (Becker, A., personal communication) which is specific for small molecule metabolism like nitrogen metabolism and electron transport [GFL$^+$01].

In the *Corynebacterium glutamicum* project [KBB$^+$03b][4] which is conducted by the Center for Genome Research two types of microarrays with different layouts were made: the CG05kPCR microarray carries approximately 500 unique open reading frames with 72 replicates each and the Cg4kPCR whole genome microarray covering approximately 93% of the genome with four replicates per gene. Within more than 40 experiments, EMMA was used to store and analyze datasets resulting from hybridizations [HBB$^+$03].

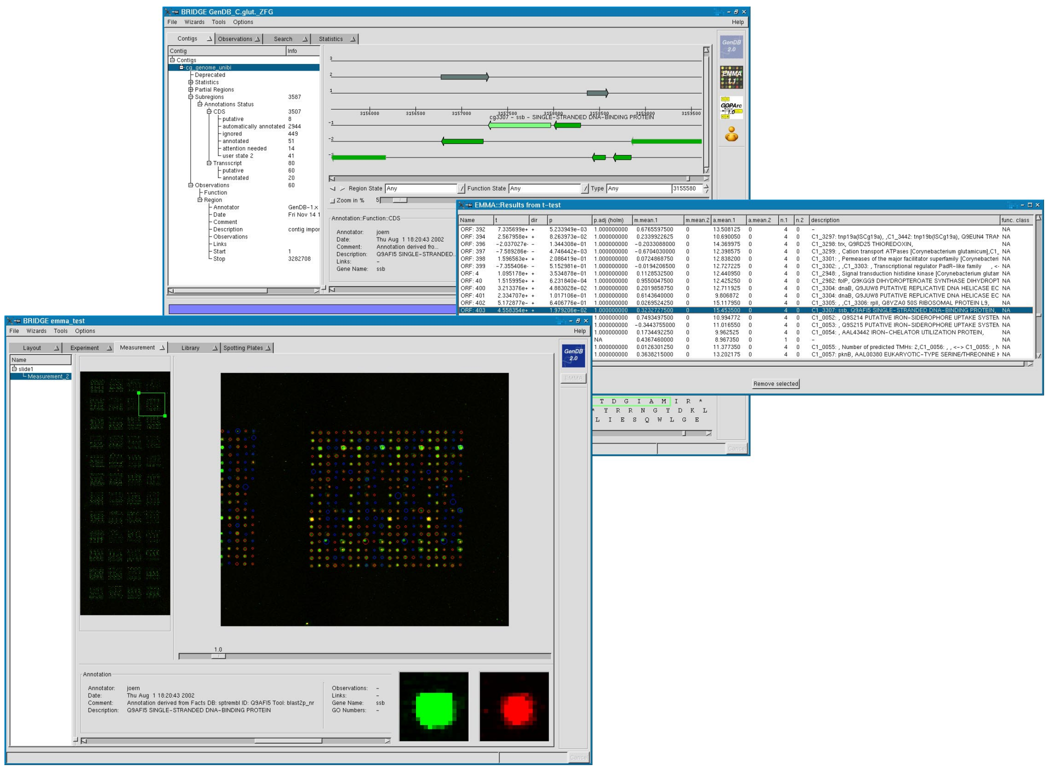
Altogether, in these projects hundreds of microarrays made from prokaryotes and eukaryotes were hybridized and analyzed with EMMA. Six different microarray layouts were imported into EMMA from robotic spotter files. These microarrays comprise small test slides as well as large scale microarrays with up to 8,000 genes and 24,000 spots. The microarray images were analyzed using AIM and ImaGene and the raw data was imported into EMMA.

[3] *http://medicago.toulouse.inra.fr/*

[4] *http://www.Genetik.Uni-Bielefeld.DE/Genetik/coryne/coryne.eng.html*

Before applying normalization, the data from each microarray was inspected by using scatterplots and a normalization preview. After applying normalization, M vs. A scatterplots of the data were generated and lists of candidate genes for differential expression were obtained by applying the t-test. Filtering was applied by removing spots with low intensity values from the result. Afterwards, the results were compared with already existing annotations stored in EMMA and in GenDB.

The EMMA system can run as a stand-alone application, but its effectiveness can be increased by the integration with other systems, e.g. GenDB and ProDB [WRB+03] as displayed in figure 9.7. The current level of integration with other software is accomplished by using the BRIDGE system.



Contig	Info
Contigs	
cg_genome_unibi	
Deprecated	
Statistics	
Partial Regions	
Subregions	3587
Annotations Status	
CDS	3507
putative	8
automatically annotated	2944
ignored	449
annotated	51
attention needed	14
user state 2	41
Transcript	80
putative	60
annotated	20
Observations	60
Function	
Region	
Annotator	GenDB-1.x
Date	Fri Nov 14 1
Comment	
Description	contig impor
Observations	
Links	
Start	1
Stop	3282708

Annotation::Function::CDS

Annotator:	joern
Date:	Thu Aug 1 18:20:43 2002
Comment:	Annotation derived fro...
Description:	Q9AFI5 SINGLE-STRANDED...
Links:	-
Gene Name:	ssb

EMMA::Results from t-test

Name	t	dir	p	p.adj (holm)	m.mean.1	m.mean.2	a.mean.1	a.mean.2	n.1	n.2	description	func. class
ORF: 392	7.335699e+	+	5.233949e-03	1.000000000	0.6765597500	0	13.508125	0	4	0	-	NA
ORF: 394	2.567958e+	+	8.263973e-02	1.000000000	0.2339922625	0	10.690050	0	4	0	C1_3297: tnp19a(ISCg19a), ,C1_3442: tnp19b(ISCg19a), Q9EUN4 TRAN	NA
ORF: 396	-2.037027e-	-	1.344308e-01	1.000000000	-0.2033088000	0	14.369975	0	4	0	C1_3298: trx, Q9RD25 THIOREDOXIN,	NA
ORF: 397	-7.589286e-	-	4.746442e-03	1.000000000	-0.6704030000	0	12.398575	0	4	0	C1_3299: , Cation transport ATPases [Corynebacterium glutamicum],C1_	NA
ORF: 398	1.596563e+	+	2.086419e-01	1.000000000	0.0724868750	0	12.838200	0	4	0	C1_3301: , Permeases of the major facilitator superfamily [Corynebacteri	NA
ORF: 399	-7.355406e-	-	5.152981e-01	1.000000000	-0.0194206500	0	12.727225	0	4	0	C1_3302: , ,C1_3303: , Transcriptional regulator PadR-like family , <-	NA
ORF: 4	1.095178e+	+	3.534878e-01	1.000000000	0.1128532500	0	12.440950	0	4	0	C1_2948: , Signal transduction histidine kinase [Corynebacterium glutam	NA
ORF: 40	1.515995e+	+	6.231840e-04	1.000000000	0.9550047500	0	12.425250	0	4	0	C1_2982: folP, Q9KGG9 DIHYDROPTEROATE SYNTHASE DIHYDROPT	NA
ORF: 400	3.213376e+	+	4.883028e-02	1.000000000	0.2019858750	0	12.711925	0	4	0	C1_3304: dnaB, Q9JUW8 PUTATIVE REPLICATIVE DNA HELICASE EC	NA
ORF: 401	2.334707e+	+	1.017106e-01	1.000000000	0.6143640000	0	9.806872	0	4	0	C1_3304: dnaB, Q9JUW8 PUTATIVE REPLICATIVE DNA HELICASE EC	NA
ORF: 402	5.172877e-	+	6.406776e-01	1.000000000	0.0269524250	0	15.117950	0	4	0	C1_3305: , ,C1_3306: rplI, Q8YZA0 50S RIBOSOMAL PROTEIN L9,	NA
ORF: 403	4.558354e+	+	1.979206e-02	1.000000000	0.3232727500	0	15.453500	0	4	0	C1_3307: ssb, Q9AFI5 SINGLE-STRANDED DNA-BINDING PROTEIN,	NA
				1.000000000	0.7493497500	0	10.994772	0	4	0	C1_0052: , Q9S214 PUTATIVE IRON-SIDEROPHORE UPTAKE SYSTEM	NA
				1.000000000	-0.3443755000	0	11.016550	0	4	0	C1_0053: , Q9S215 PUTATIVE IRON-SIDEROPHORE UPTAKE SYSTEM	NA
				1.000000000	0.1734492250	0	9.962525	0	4	0	C1_0054: , AAL43442 IRON-CHELATOR UTILIZATION PROTEIN,	NA
				NA	0.4367460000	0	8.967350	0	1	0	-	NA
				1.000000000	0.0126301250	0	11.377350	0	4	0	C1_0055: , Number of predicted TMHs: 2,C1_0056: , , <-> C1_0055: , N	NA
				1.000000000	0.3638215000	0	13.202175	0	4	0	C1_0057: pknB, AAL00380 EUKARYOTIC-TYPE SERINE/THREONINE K	NA

Annotation

Annotator:	joern	Observations:	-
Date:	Thu Aug 1 18:20:43 2002	Links:	-
Comment:	Annotation derived from Facts DB: sptrembl ID: Q9AFI5 Tool: blast2p_nr	Gene Name:	ssb
Description:	Q9AFI5 SINGLE-STRANDED DNA-BINDING PROTEIN	GO Numbers:	-

Figure 9.7.: Integration of EMMA and GenDB via BRIDGE: a spot on a slide has been selected in EMMA while GenDB provides additional annotation information about the content of the spot (in this case a CDS). GenDB displays the CDS position in its contig view.

The content of a spot (e.g. oligonucleotide, PCR product, EST) or the result of a statistical test or cluster analysis may be linked directly to a region stored in the GenDB genome

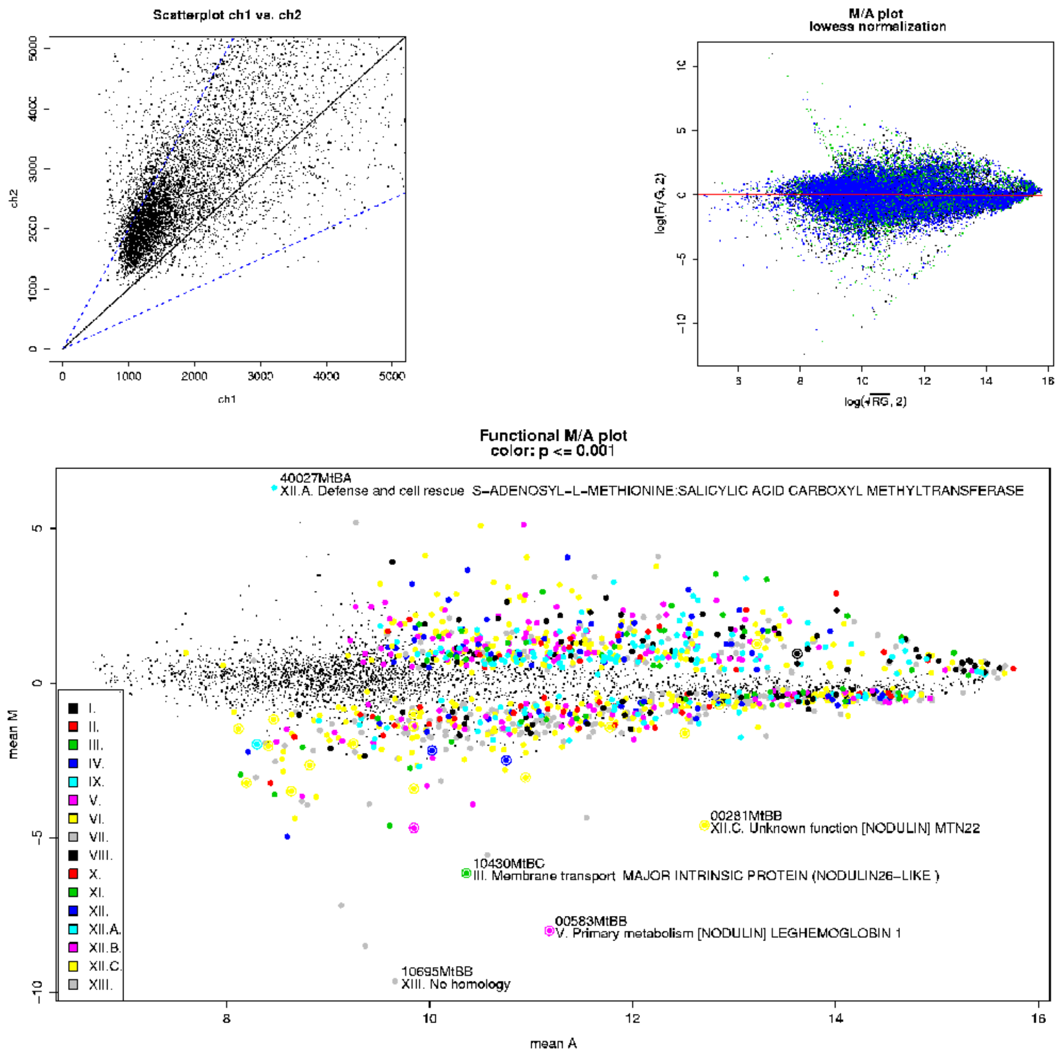


Figure 9.8.: Three scatterplots of experimental data created with EMMA's R-plotting device, each representing a distinct stage of analysis. In this experiment, three Mt6k-RIT arrays [Küs03] were hybridized against labeled probes from mature root nodules and uninfected roots (total RNA and hybridization data were provided by Pascal Gamas, INRA-CNRS Toulouse, France and Helge Küster, Center for Genome Research, Bielefeld University, Germany). The top-left plot contains a scatterplot of raw intensities for each spot and the top-right graphic shows an M vs. A plot of all slides after lowess normalization. The third plot shows an M vs. A plot for each gene. Differentially expressed genes identified by the t-test are mapped on Monica Riley categories. Annotation information can be added to points of interest interactively.

annotation system that provides extensive information about its annotated function. Selecting a spot will then show the corresponding region in the contig view or display a report with detailed information in the GenDB frontend.

Integrating EMMA with the GOPArc system allows further analysis of microarray data based on functional categories (e.g. COG, Monica Riley) or metabolic pathways. Scatterplots of microarray data can be generated according to the functional classification of the corresponding genes (see figure 9.8).

CHAPTER 10

Summary

This chapter intends to summarize the most important aspects of this work. A graphical overview of the timeline of this work is presented that shows the different steps for the development of the BRIDGE system.

10.1. Summary of this work

In this PhD thesis, the BRIDGE system was developed as a framework for the integration of specialized components for separate scopes. Due to the modular architecture and system design, five modules were implemented:

- GenDB-2.0:
  Based on GenDB-1, a more complex data model that now contains a number of subclasses for arbitrary genomic regions such as CDS, RBS, tRNAs, Operons, etc. was designed. In addition to that, a more flexible tool concept was realized. Individual bioinformatics tools are now implemented as separate classes that can also have a special class for the observations they generate. Furthermore, all tools are integrated into a grid framework for a quite comfortable scheduling of jobs using the Sun Grid Engine. Last but not least, the new version of GenDB provides a well designed web frontend that can be used for a distributed annotation of genomes.

- EMMA-1.1:
  Since there was no open source transcriptomics platform that fulfilled our requirements, the EMMA system was developed. The current version provides the most important features for the analysis of microarray data.

- GOPArc-1.0:
  The GOPArc system was implemented as a prototype for a module that supports the analysis and visualization of metabolic pathways based on the ideas developed in the PathFinder system. GOPArc also provides tools for the analysis of functional categories in genomic data.

- GPMS:
  The *General Project-Management System* is one of the key components that is required for the integration of heterogeneous data from different data sources and projects. It is essential for the administration of users and allows the realization of accurate and individual access policies.

- BRIDGE:
  The original BRIDGE system basically consists of two core components that were developed for the integration of heterogeneous data into a common framework. The *BridgeFunc* layer provides the functionality for accessing and initializing external or remote objects which can be especially useful for programmers working with different data sources. For a comfortable usability, a framework of GUI modules was implemented that facilitates the integration of specialized components into an interactive and highly customizable user frontend.

A well structured and extensible platform for systems biology was developed by incorporating a set of full featured specialized components. The implemented graphical user frontends already support a number of features that directly link different types of data (e.g. map expression data onto metabolic pathways). In addition to the graphical user frontends, the BRIDGE system can be used to implement individual algorithms for analyzing the data in an easy and intuitive way. The BRIDGE system allows to answer questions like "*Show me all genes of pathway A that are 2-fold up-regulated in experiment B, have an unusual GC content and ...*" by using the higher level programming environment provided by the O2DBI server classes and the *BridgeFunc* layer. Abstract pseudo code descriptions for special tasks can be translated almost directly into executable and human readable programs.

Finally, the usefulness and utility of this approach was shown for various sample applications.

To conclude this chapter, the most important milestones of this work are illustrated in figure 10.1.

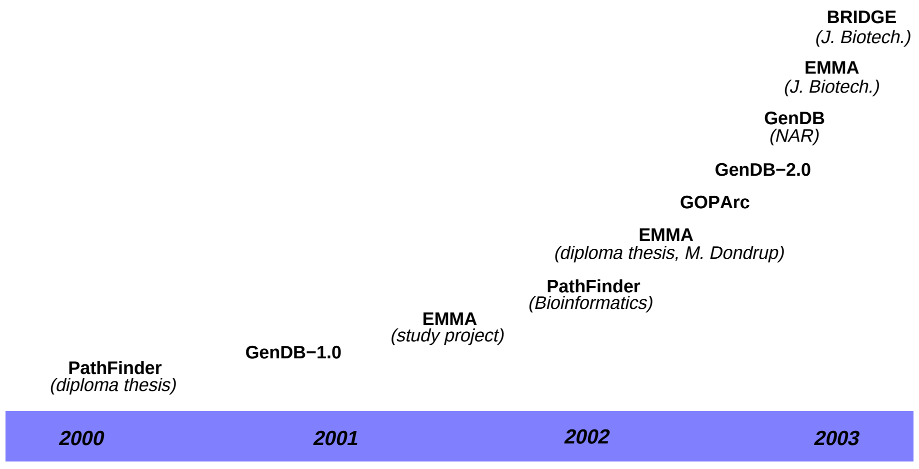


Figure 10.1.: Starting in spring of 2000 with my diploma thesis about the PathFinder system, the BRIDGE system was developed in several steps. Finally, GenDB, EMMA, and the BRIDGE system were accepted for publication.

Starting after my diploma thesis, most of the initial work was directed towards establishing a first stable version of the GenDB genome annotation system that incorporated the original PathFinder software. The development of the EMMA system was initiated by a study project managed by myself and Michael Dondrup continued this project by evaluating and implementing a variety of methods for the analysis of microarray data in his diploma thesis that was also conducted on my personal advice. The publication of the PathFinder software was followed by the development of a GOPArc prototype that enhanced the PathFinder system in several ways. At the same time a completely redesigned version of GenDB was implemented and finally published in NAR. Recently, the EMMA system and the BRIDGE architecture were accepted for publication in a Special Issue of the Journal of Biotechnology.

CHAPTER 11

Discussion

In the following chapter, the results of this PhD thesis are analyzed critically and some aspects are discussed that might be improved by further work on the system. Finally, a short outlook is presented, illustrating some ideas for an ongoing development towards a platform for systems biology.

11.1. Results

Although many questions can be answered with the BRIDGE system, there are still some challenging open questions and problems waiting for solutions. Beyond the apparent request to share and unify these concepts of data integration, there is still an enduring need for widely accepted standard data formats. In my opinion, there is hope that the idea of providing open source software as a common resource helps to concentrate efforts and allows other researchers to integrate their own ideas, thus preventing the reinvention of the wheel in many areas. Only the design of the GenDB software currently restricts the use of the BRIDGE system to prokaryotes but the next generation of GenDB is already designed for supporting the analysis of eukaryotic organisms. The architecture of the BRIDGE platform currently features the inclusion of additional components or interfaces as **a)** an O2DBI application that directly integrates into the BRIDGE layer, **b)** a commonly used web service and **c)** a separate graphical user interface compliant to our specialized components. In particular, further extension and integration of web services will be a major task for future developments.

11.2. Outlook

The usefulness and applicability of the currently implemented BRIDGE platform was illustrated in chapter 9. Nevertheless, there are still some things left that could be done to improve the system and to enhance it. Future extensions and further development of this software could be directed towards incorporating additional specialized components for detailed genome comparison with various navigation metaphors and for the analysis of regulatory mechanisms that allow users to create a comprehensive repository of enriched genome annotations. This includes the extension of the GenDB system for the analysis of eukaryotic organisms, a re-implementation of the GOPArc prototype, a component for the analysis of proteome data, novel modules for comprehensive genome comparison, and also for the prediction or analysis of operon structures. Furthermore, it might be worthwhile to employ other more enhanced toolkits like Java Beans or the Qt toolkit instead of Perl-Gtk for all further development.

Another issue that will remain important for all further analysis is the necessity to ensure a certain quality of the obtained raw data. For instance, the quality of a contig sequence matters quite a lot for all further downstream sequence analysis and genome research: as displayed in figure 11.1, a polished DNA sequence significantly improves the gene prediction and reduces the number of false positives.

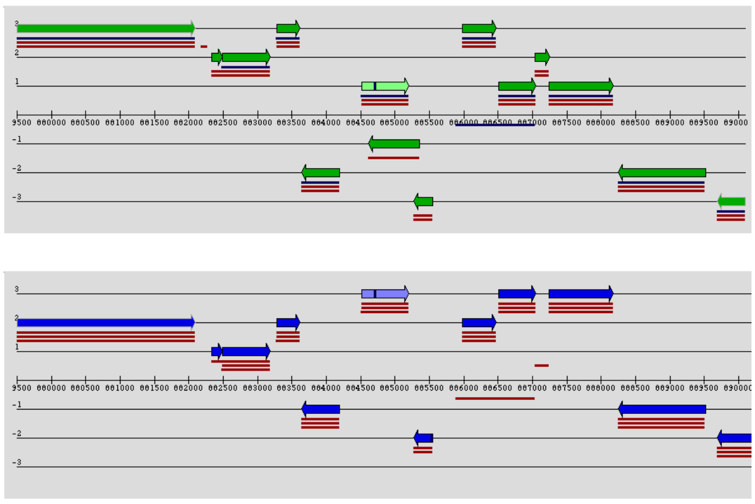
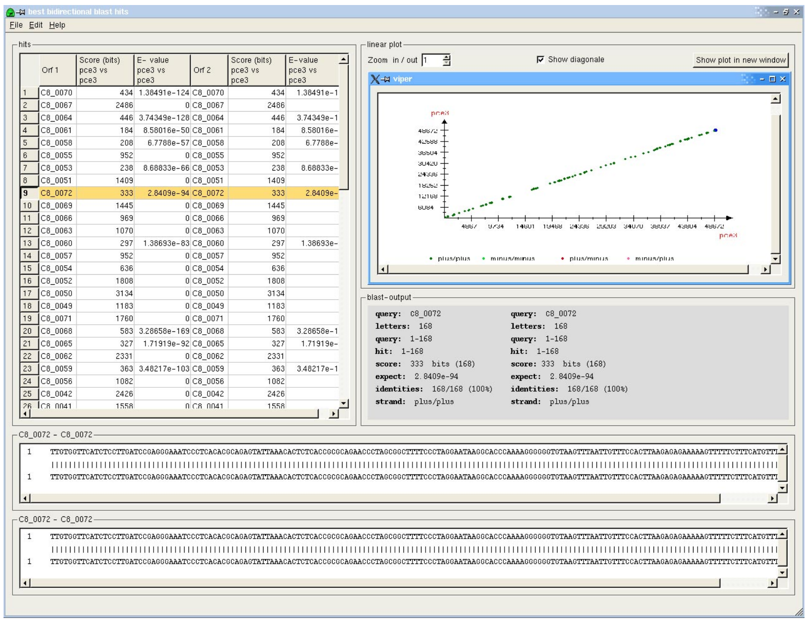


Figure 11.1.: Sequence quality matters a lot for all further downstream analysis. The upper part shows a screenshot of the GenDB main window displaying the results of a first gene prediction after finishing the shotgun sequencing phase. The lower part displays the same region after polishing the contig, the contradictory CDS prediction in the middle was resolved automatically by recomputing the gene prediction and another small CDS was no longer predicted as a real gene.

While several new standards and approaches for the integration of heterogeneous data are currently under development, it is clear that widely used stable data exchange mechanisms should be incorporated into the BRIDGE system once they are established. Additionally, the versatility of this software could be enhanced by implementing a generic query interface that provides interactive exploration of complex heterogeneous data structures.

Recently, an ongoing study project (VIPER) was initiated which aims at providing different views for genome comparison. A viewer for bi-directional best BLAST hits (see figure 11.2), an MGA frontend for multiple genome alignments, and a module for comparison based on domains and motifs (Pfam) are currently under development using Qt for the development of all graphical user interfaces.



	Orf 1	Score (bits) pce3 vs pce3	E- value pce3 vs pce3	Orf 2	Score (bits) pce3 vs pce3	E-value pce3 vs pce3
1	C8_0070	434	1.38491e-124	C8_0070	434	1.38491e-1
2	C8_0067	2486	0	C8_0067	2486	
3	C8_0064	446	3.74349e-128	C8_0064	446	3.74349e-1
4	C8_0061	184	8.58016e-50	C8_0061	184	8.58016e-
5	C8_0058	208	6.7788e-57	C8_0058	208	6.7788e-
6	C8_0055	952	0	C8_0055	952	
7	C8_0053	238	8.68833e-66	C8_0053	238	8.68833e-
8	C8_0051	1409	0	C8_0051	1409	
9	C8_0072	333	2.8409e-94	C8_0072	333	2.8409e-
10	C8_0069	1445	0	C8_0069	1445	
11	C8_0066	969	0	C8_0066	969	
12	C8_0063	1070	0	C8_0063	1070	
13	C8_0060	297	1.38693e-83	C8_0060	297	1.38693e-
14	C8_0057	952	0	C8_0057	952	
15	C8_0054	636	0	C8_0054	636	
16	C8_0052	1808	0	C8_0052	1808	
17	C8_0050	3134	0	C8_0050	3134	
18	C8_0049	1183	0	C8_0049	1183	
19	C8_0071	1760	0	C8_0071	1760	
20	C8_0068	583	3.28658e-169	C8_0068	583	3.28658e-1
21	C8_0065	327	1.71919e-92	C8_0065	327	1.71919e-
22	C8_0062	2331	0	C8_0062	2331	
23	C8_0059	363	3.48217e-103	C8_0059	363	3.48217e-1
24	C8_0056	1082	0	C8_0056	1082	
25	C8_0042	2426	0	C8_0042	2426	
26	C8_0041	1558	0	C8_0041	1558	

```
query:  C8_0072                    query:  C8_0072
letters:  168                      letters:  168
query:  1-168                      query:  1-168
hit:  1-168                        hit:  1-168
score:  333  bits (168)            score: 333  bits (168)
expect:  2.8409e-94                expect:  2.8409e-94
identities:  168/168 (100%)        identities:  168/168 (100%)
strand:  plus/plus                 strand:  plus/plus
```

C8_0072 - C8_0072

```
1    TTGTGGTTCATCTCCTTGATCCGAGGGAAATCCCTCACACGCAGAGTATTAAACACTCTCACCGCGCAGAACCCTAGCGGCTTTTCCCTAGGAATAAGGCACCCAAAAGGGGGGTGTAAGTTTAATTGTTTCCACTTAAGAGAGAAAAAGTTTTTCTTTCATGTTT
     ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
1    TTGTGGTTCATCTCCTTGATCCGAGGGAAATCCCTCACACGCAGAGTATTAAACACTCTCACCGCGCAGAACCCTAGCGGCTTTTCCCTAGGAATAAGGCACCCAAAAGGGGGGTGTAAGTTTAATTGTTTCCACTTAAGAGAGAAAAAGTTTTTCTTTCATGTTT
```

C8_0072 - C8_0072

```
1    TTGTGGTTCATCTCCTTGATCCGAGGGAAATCCCTCACACGCAGAGTATTAAACACTCTCACCGCGCAGAACCCTAGCGGCTTTTCCCTAGGAATAAGGCACCCAAAAGGGGGGTGTAAGTTTAATTGTTTCCACTTAAGAGAGAAAAAGTTTTTCTTTCATGTTT
     ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
1    TTGTGGTTCATCTCCTTGATCCGAGGGAAATCCCTCACACGCAGAGTATTAAACACTCTCACCGCGCAGAACCCTAGCGGCTTTTCCCTAGGAATAAGGCACCCAAAAGGGGGGTGTAAGTTTAATTGTTTCCACTTAAGAGAGAAAAAGTTTTTCTTTCATGTTT
```

Figure 11.2.: Visualization of bi-directional BLAST hits using the Qt frontend of the VIPER project.

Last but not least, a vision for the ongoing development of this platform in the future could be directed towards a *Genomic Desktop Environment - GDE*, that incorporates most of the currently available bioinformatics tools and applications (e.g. EMBOSS) as an integrative modular platform for the comprehensive analysis and simulation of complex biological systems.

APPENDIX A

Selected topics of the source code

The following sections contain more details about the most important topics that were implemented in this work. Instead of pure source code examples the APIs of some modules are listed in order to increase the readability of the some modules presented here. All Perl modules contain some inline documentation in POD (Plain Old Documentation) format that can be extracted automatically and converted into different other formats (e.g. HTML, $\LaTeX$, or man pages).

A.1. Role and right definitions for GenDB-2.0

This section describes the *Roles* and *Rights* as they were defined for the genome annotation system GenDB-2.0 which extensively uses different roles for a sophisticated access control.

```
###################################
### ROLES defined for GenDB-2.0 ###
###################################

PROJECT_CLASS GENDB

# user with read only permissions and almost completely restricted access
ROLE Guest
        RIGHT basic_access
```

```
# user who is allowed to write annotations and recompute the observations
# for a single region
ROLE Annotator
        RIGHT basic_access
        RIGHT annotate
        RIGHT export_region_data
        RIGHT recompute

# (external) user who is allowed to do most of the necessary tasks for
# maintaining a project (e.g. import/export/edit/delete contig sequences,
# add tools and submit all jobs for their computation)
# this role should be used if several persons have to edit the sequence,
# e.g. to correct frameshifts
ROLE Maintainer
        RIGHT basic_access
        RIGHT recompute
        RIGHT submit_jobs
        RIGHT contig_import_export
        RIGHT edit_sequence
        RIGHT add_tools
        RIGHT export_region_data
        RIGHT delete_contig
        RIGHT annotate
        RIGHT region_prediction

# user who is responsible for the database and for the solution of bugs and problems
# can do almost everything and also MODIFY THE DATABASE (e.g. alter table)
ROLE Developer
        RIGHT contig_import_export
        RIGHT region_prediction
        RIGHT submit_jobs
        RIGHT recompute
        RIGHT edit_sequence
        RIGHT add_tools
        RIGHT export_region_data
        RIGHT delete_contig
        RIGHT configure_project
        RIGHT basic_access
        RIGHT annotate
        RIGHT modify_db

# user who is responsible for the project and who can do everything except
# modifying the database (e.g. configure the project)
# has to add Maintainers, Annotators and Guests but cannot grant all rights
# that are needed by Developers
ROLE Chief
        RIGHT annotate
        RIGHT add_user
        RIGHT contig_import_export
        RIGHT region_prediction
        RIGHT submit_jobs
        RIGHT recompute
        RIGHT edit_sequence
        RIGHT add_tools
        RIGHT export_region_data
        RIGHT delete_contig
        RIGHT configure_project
        RIGHT basic_access
```

```
####################################
### RIGHTS defined for GenDB-2.0 ###
####################################

PROJECT_CLASS GENDB

RIGHT basic_access
        DS_TYPE GENDB
                DB select
        DS_TYPE GPMSDB
                DB select
                TABLE sessions delete update insert
                TABLE sessions_not_permanent delete update insert
                TABLE sessions_permanent delete update insert
                TABLE Member_User_Project_Configs update delete insert
                TABLE Member_User_Project_Configs_hash_value update delete insert
                TABLE ProjectManagement_counters update

RIGHT annotate
        DS_TYPE GENDB
                DB insert update

RIGHT export_region_data

RIGHT recompute
        DS_TYPE GENDB
                DB delete update insert

RIGHT submit_jobs
        DS_TYPE GENDB
                DB insert update delete

RIGHT contig_import_export
        DS_TYPE GENDB
                DB insert update delete

# may only be granted to user if user has the right to annotate
RIGHT edit_sequence
        DS_TYPE GENDB
                DB update insert

RIGHT add_tools
        DS_TYPE GENDB
                DB insert update

RIGHT delete_contig
        DS_TYPE GENDB
                DB delete

RIGHT region_prediction
        DS_TYPE GENDB
                DB insert update delete

RIGHT configure_project
        DS_TYPE GENDB
                DB insert update delete
```

```
RIGHT modify_db
        DS_TYPE GENDB
                DB insert update delete alter index create drop references

RIGHT add_user
        DS_TYPE GENDB
                DB grant insert update delete
        DS_TYPE GPMSDB
                DB grant insert update delete
```

A.2. API of the ApplicationFrame

The following description of a general *Application_Frame* was directly included from the documentation of the Perl module.

```
Description
     An Application_Frame provides a general easy-to-use framework
     for the General Project Management System (GPMS). It should
     be used for all O2DBI connections based on the GPMS (see
     documentation of O2DBI-II by B. Linke for further
     details about master objects etc.).

Concepts
     An Application_Frame acts as a container for all GPMS data
     that are required by an application for connecting to an
     O2DBI database. It provides access to:

     o a ProjectManagement master object
     o an application master object
     o a ProjectManagement::Project object
     o a ProjectManagement::User object
     o a ProjectManagement::Member object
     o the users password
     o user specific configurations for a Project

     For using the Application_Frame a subclass has to be created
     that contains an application specific _init_application
     method, that provides the application master (see the
     documentation of _init_application for more details). The
     Application_Frame was initially introduced for some
     bioinformatics applications developed at the Center for Genome
     Research, Bielefeld University. But it may be used as well
     with every other GPMS based application.

Methods
     o new($login, $passwd [,$gpms_master] [,$errh])
          Constructor, used for creating a new
          GPMS::Application_Frame object that provides the O2DBI-II
          master objects and all GPMS data relevant for the
          application. $errh is an optional error handler method,
          which is executed if an error occurs. As an argument
          $errh receives a string containing the error message.
          When $errh is specified it will be executed everytime
          a method of the object is called and an error occurs.
          The optional argument $gpms_master is a
          ProjectManagement master object. It may be used when
          several Application_Frame objects are supposed to share
          the same GPMS master.

     o errh([$errh])
          Method used to get/set the error handler method. $errh
          is a reference on a subroutine that is executed if an
          error occurs. The only argument that is returned is the
          error message string.

     o login
          This method returns the login name of an user that was
          specified in the constructor method.
```

```
o passwd
    The users password is required for establishing the
    connection to a database. Therefore the password is
    provided by this method.

o gpms_master([$master])
    This method can be used to get/set the O2DBI-II master
    object for the GPMS.

o application_master([$master])
    Get/set the O2DBI-II master object specific for the
    application or hash of O2DBI-II master objects if a
    project uses more than one database.

o user
    Use this method to get the ProjectManagement::User
    object.

o project([$project])
    This method can be used to get/set the project.
    $project can be a ProjectManagement::Project object or
    simply the name of a project.

o get_available_projects($type)
    This method returns an array reference of all usable
    projects for the specified user. The type can be used
    to specify a subclass of ProjectManagement::Project
    (e.g. ProjectManagement::Project::GENDB).

o user_name
    Get the full name of the current user.

o user_email
    Get the email address of the current user.

o error
    Default error method that can be used to get/set an
    error message if an error occured.

o _init_application($project)
    This method has to be overloaded when a new subclass is
    created. The derived method can implement project
    specific initializations. It should always return an
    application master or a reference on a hash of
    application masters if a project uses more than one
    database. Since the Application_Frame is only
    applicable for projects using the second generation of
    the O2DBI tool, other applications that use the old
    version by J. Clausen have to return 1 for successful
    connections or 0 if an error occured. The $project
    argument specifies a ProjectManagement::Project for
    which the Application_Frame should be initialized.

o project_name
    Get the name of the project the Application_Frame
    was created for.
```

```
     o project_description
        Get the description of the currently used project.

     o member([$member])
        Use this method to get/set the current member.

     o right($right_name)
        This method returns the value for a given project right
        for the current user. Use this method to check the
        individual permissions (rights).

     o rights
        This method returns a reference on a hash of all
        project rights defined for the current user.

     o projectDB_by_datasource_type_name($datasource_type_name)
        Use this method to retrieve the database of a project
        for a specified datasource_type.

     o project_dbs
        This method returns a reference on an array of all
        databases in the current project.

     o project_datasources
        Use this method to retrieve a reference on an array of
        all available datasources for the current project.

     o user_project_config([$config])
        Use this method to get/set the complete project
        configurations for the current member of a project.
        $config is a hash of hash containing the configuration
        parameters and values for different configuration
        sections.

     o destroy
        Delete the Application_Frame object and clean up
        everything.

Deriving a sub-class of an Application_Frame
       For using the Application_Frame a subclass has to be cre-
       ated that contains an application specific "_init_applica-
       tion" method. This method has to provide the application
       master (see the documentation for _init_application for
       more details). The Application_Frame was initially intro-
       duced for some bioinformatics applications developed at
       the Center for Genome Research, Bielefeld University, but
       it may be used as well with every other application that
       uses the GPMS.

       You can easily adopt the Application_Frame for your own
       systems by deriving a sub-class and overloading a number
       of methods:

       o _init_application($project, $options)
           This method initializes the application specific
           database modules. The default implementation is geared
           towards an O2DBI-II generated database interface with
           a MySQL backend. The derived method can implement pro-
           ject specific initializations. It should always return
           an application master or a reference on a hash of
           application masters if a project uses more than one
           database. Since the Application_Frame was basically
           developed for projects using the second generation of
           the O2DBI tool, other applications have to return 1
           for successful connections or 0 if an error occured
           (e.g. projects using the old version of O2DBI by J.
           Clausen). "_init_application" is executed when the
           method project() is executed for setting a project.
           The $project argument specifies a ProjectManage-
           ment::Project for which the Application_Frame should
           be initialized. The argument $options may be any
           options.

       o master_class()
           This method returns the name of the O2DBI-II master
           class that should handle an application. The name is
           used in the default implementation of "_init_applica-
           tion" to create the backend and master objects.

       o db_type()
           The GPMS datasource type that can be handled by the
           Application_Frame.

       Both methods, "master_class()" and "db_type()" are used by
       the BridgeFunc layer to resolve external references. The
       derived classes also have to care for loading all neces-
       sary modules, e.g. the O2DBI-II generated backend and mas-
       ter modules.


       Example: The GenDB-2.0 Application_Frame:


       package GPMS::Application_Frame::GENDB;

       use strict; use GENDB::DB; use GENDB::DB_MySQL;
       use base('GPMS::Application_Frame');

       1;

       sub master_class {
           return 'GENDB::DB';
       }

       sub db_type {
           return 'GENDB';
       }
```

A.3. A sample script and project initialization

The sample script below illustrates the initialization of a GenDB project. All contig objects in the project database are retrieved and their names are printed.

```
#!/usr/bin/env perl

# simple GenDB demo script that reads all contigs and writes their names

# $Id: quellcode.tex,v 1.23 2004/03/03 12:36:22 agoesman Exp $

use strict;
use Carp;
use Getopt::Std;
use Term::ReadKey;
use IO::Handle;
use GPMS::Application_Frame::GENDB;

#
#  this is necessary if the script is started via rsh(1)
#  otherwise you won't see any output until the first <RETURN>
#
STDOUT->autoflush(1);

sub usage {
    print "gendb_demo - get all contig sequences and write their names\n";
    print "usage: gendb_demo -p <project>\n\n";
}

# global variables
our($opt_p);

getopts('p:');

# start sanity checks
if (!$opt_p) {
    usage;
    print "ERROR: Can't initialize GenDB: No project name given!\n";
    exit 1;
};

# get the login name of the current user
my $user = defined( $ENV{'LOGNAME'} ) ? $ENV{'LOGNAME'} : (getpwuid( $> ))[0];

print "Enter your database password: ";
ReadMode('noecho');
my $password = ReadLine(0);
chomp $password;
print "\n";
ReadMode('normal');

# try to initialize GenDB project
# initialize an Application_Frame for the current project
my $gendbAppFrame = GPMS::Application_Frame::GENDB->new($user, $password);

# check if the initialization succeeded
die "Unable to initialize ApplicationFrame for GenDB project!" unless (ref $gendbAppFrame);
```

```
# try to initialize a project for the given name
$gendbAppFrame->project($opt_p);

# check a basic privilege
exit unless $gendbAppFrame->right("basic_access");

# get a global O2DBI-2 master object
my $master = $gendbAppFrame->application_master();

# fetchall contigs and print their names
print "Contigs in GenDB project $opt_p:\n\n";
my $contigs = $master->Region->Source->Contig->fetchall();
foreach my $contig (@$contigs) {
    print $contig->name."\n";
}

print "\nDone.\n\n";
```

A.4. The GenDB-2.0 tool and job concept

This section describes all components that are essential for understanding the GenDB-2.0 tool and job concept in more detail.

A.4.1. A sample tool

In GenDB-2.0 each tool is integrated as a separate subclass (e.g. *Tool::Function::HTH*) in order to facilitate an easy and individual implementation. The following example describes some details of a tool implementation:

```
Name
       GENDB::DB_Server::Tool::Function::HTH - integration of
       helix-turn-helix

Description
       This module implements the server side extensions to class
       GENDB::DB::Tool::Function::HTH.  It can be used to compute
       helix-turn-helix motifs in coding sequences (CDS). HTH can
       be run as a scheduled tool or on the fly with user defined
       settings. Observations can be recomputed on demand, the
       tool result is returned as plain text.

Additional methods
       o bool can_run_queued()
           Use this method to check if Jobs can be created for
           running HTH so that it can be scheduled (e.g. via
           Codine) and run queued.
             RETURNS: true, if HTH can run queued,
                      false otherwise

       o bool can_recompute_observation()
           Use this method to check if HTH observations can be
           recomputed on demand.
             RETURNS: true if observations can be recomputed,
                      false otherwise

       o bool can_run_immediately()
           Use this method to check if HTH can be run on the fly
           with user defined settings.
             RETURNS: true if HTH can run immediately,
                      false otherwise

       o SCALAR run(OBJECT)
           Overloaded method for running HTH. Depending on the
           type of the parameter this method will perform differ-
           ent actions:
               GENDB::DB::Region:      for a given region (e.g. a
                                       CDS) HTH will be run on the
                                       fly and return the tool re-
                                       sult
               GENDB::DB::Observation: the tool result will be re-
                                       recomputed for the given
```

```
                               observation and returned as
                               a string
            GENDB::DB::Job:         in this case, HTH is com-
                               puted, the result will be
                               parsed, and observations are
                               stored in the database,
                               RETURNS: true on success,
                                        false otherwise
          RETURNS: see above for return values on success,
                   returns false for errors, for severe failures
                   this method dies (-> use eval)

       o STRING command_line()
           Create the command line with parameters from the tool
           configuration.
             RETURNS: the command line for HTH

       o bool auto_annotate(OBJECT)
           Run a simple HTH auto annotator for all created obser-
           vations of a tool/region combination.  For HTH we cre-
           ate a new Region::CDS_Feature::HTH first, using the
           previously created Observation::Region::Feature.
           Afterwards we add an Annotation::Region and an Annota-
           tion::Function for this region.
             OBJECT:  the GENDB::DB::Region that should be anno-
                      tated, in this case only CDS
             RETURNS: true if the region was annotated successfully,
                      false otherwise
```

A.4.2. Computing tools – `runtool.pl`

This script can not only be used for the computation of batch jobs but also for debugging newly implemented tools and for recomputing single jobs, e.g. all tools for a specific region.

```
#!/usr/bin/env perl

use vars qw($opt_p $opt_j $opt_d $opt_v $opt_a);
use strict;

use Carp qw(croak);
use Data::Dumper;
use Getopt::Std;
use Term::ReadKey;
use IO::Handle;
use GENDB::Common::GlobalConfig;
use GPMS::Application_Frame::GENDB;

#
#  this is necessary if the script is started via rsh(1)
#  otherwise you won’t see any output until the first <RETURN>
#
STDOUT->autoflush(1);
```

```
sub usage {
    print "runtool - executes single jobs for GenDB 2\n";
    print "usage: runtool -p <project> -j <job id>\n"
        ."    -d debug mode (necessary if you try to run a job WITHOUT SGE scheduler)\n"
        ."    -v verbose tools\n"
        ."    -a start auto annotator if available\n\n";
}


getopts('p:j:dva');

if (!$opt_p) {
    usage;
    print STDERR "Error: Missing project name!\n\n";
    exit 1;
}

if (!$opt_j) {
    usage;
    print STDERR "Error: Missing job id (project: $opt_p)!\n\n";
    exit 1;
}


# if we are trying to run this script in the debug mode instead of the standard gendb user
# we have to enter a database password...
my $usr = $GENDB_DEFAULT_USER;
my $password = $GENDB_DEFAULT_PWD;
if ($opt_d) {
    $usr = defined( $ENV{'LOGNAME'} ) ? $ENV{'LOGNAME'} : (getpwuid( $> ))[0];

    print "Enter your database password: ";
    ReadMode('noecho');
    my $password = ReadLine(0);
    chomp $password;
    print "\n";
    ReadMode('normal');
};


# try to init GenDB project
# initialize an Application_Frame for the current project
my $gendbAppFrame = GPMS::Application_Frame::GENDB->new($user, $password);

# check if the initialization succeeded
die "Unable to initialize ApplicationFrame for GenDB project!" unless (ref $gendbAppFrame);

# try to initialize a project for the given name
$gendbAppFrame->project($opt_p);

# check if the user has basic access to the current project
# and the privilege to write observations/annotate
exit unless $gendbAppFrame->right("annotate");

# get a global O2DBI-2 master object
my $master = $gendbAppFrame->application_master();
```

```
# try to init job for given job id
my $job = $master->Job->init_id($opt_j);
if (!ref $job) {
    usage;
    print STDERR "Error: No job with id $opt_j!\n";
    exit 1;
};

# reset job state to submitted if we recompute a single job without using the JobSubmitter
if ($opt_d) {
    $job->submitted($opt_p);
};

# get tool for job
my $tool = $job->tool;

# print some debug output
$tool->verbose(1) if $opt_v;

# try to run the job
my $finished;
eval {
    $job->running($opt_p);
    $finished = $tool->run($job, $opt_p);
};

# update job status
if ($@) {
    print STDERR "Error: Running job $opt_j for project $opt_p failed: $@\n";
    $job->failed($opt_p);
}
else {
    if(!$finished) {
        # this is required for pipeline tools only
        $job->pending($opt_p);
    }
    else {
        # run auto annotator if tool has reference on annotator
        if (ref $tool->auto_annotator && $opt_a) {
            print STDERR "Starting auto annotator...\n";
            eval {
                $tool->auto_annotate($job->region, $opt_p);
                # we’re done
                $job->finished($opt_p);
            };

            # check eval result
            if ($@) {
                print STDERR "Error: Running job $opt_j for project $opt_p failed: $@\n";
                $job->failed($opt_p);
            }
        }
        else {
            # we’re done
            $job->finished($opt_p);
        }
    }
}
```

A.4.3. The GENDB::Job class definition

The following section describes the API of the class *GENDB::DB::Job*.

```
Name
       GENDB::DB_Server::Job - a class for scheduling the compu-
       tation of bioinformatics tools

Description
       This module implements the server side extensions for the
       class GENDB::DB::Job.  A Job can have 6 different state
       values represented by integer values indicating the status
       of a Job: PENDING (1), SUBMITTED (2), RUNNING (3), CAN-
       CELLED (4), FINISHED (5), or FAILED (6). Initially after
       their creation Jobs are PENDING. When a Job was registered
       successfully in the scheduler the status is SUBMITTED. All
       other stati should be self explaining.

Additional methods
       o <Job> create_job(<Region>, <Tool>, integer)
           Additional constructor method to the standard "create"
           method. It creates a new Job object, sets the
           date_ordered attribute to the current time and initial
           default status of the Job is set to PENDING. An
           optional integer value can be used to set an initial
           priority for the Job.
             <Region>: an arbitrary region for which the Job
                       should be created
             <Tool>:   the required Tool for the Job
             integer:  a numeric value for setting a defaul pri-
                       ority of the new Job
             RETURNS:  the newly created Job object, -1 if the
                       creation failed

       o integer get_current_state()
           Get the current status of a Job directly from the
           database.
             RETURNS: an integer for the current status.

       o bool submit(string, bool)
           This method submits a Job to the scheduler and sets
           the job status to submitted. A single Job is only sub-
           mitted if 1) the status is PENDING, 2) all mandatory
           Jobs were finished successfully (state = FINISHED) and
           3) all optional Jobs are done (state = FINiSHED, CAN-
           CELLED, or <FAILED>).
             string:  name of the current project
             bool:    option for activating automatic annotators
             RETURNS: true if the Job was submitted successfully,
                      false otherwise

       o bool submitted(string)
           Set the status of a Job to SUBMITTED.
             string:  name of the current project
             RETURNS: true if the new status could be set,
                      false otherwise
```

```
o bool pending(string)
    Set the status of a Job to PENDING.
      string:  name of the current project
      RETURNS: true if the new status could be set,
               false otherwise

o bool running(string)
    Set the status of a Job to RUNNING. This can only be
    done successfully if the previous status was SUBMITTED.
      string:  name of the current project
      RETURNS: true if the new status could be set,
               false otherwise

o bool finished(string)
    Set the status of a Job to FINISHED. This can only be
    done successfully if the previous status was RUNNING.
      string:  name of the current project
      RETURNS: true if the new status could be set,
               false otherwise

o bool failed(string)
    Set the status of a Job to FAILED. This can only be
    done successfully if the previous status was RUNNING.
      string:  name of the current project
      RETURNS: true if the new status could be set,
               false otherwise

o bool cancel(string)
    Set the status of a Job to CANCELLED. This can only be
    done successfully if the previous status was SUBMITTED
    or RUNNING.
      string:  name of the current project
      RETURNS: true if the new status could be set,
               false otherwise

o bool is_pending()
    This method can be used to query whether a Job is cur-
    rently PENDING.
      RETURNS: true if the state is PENDING,
               false otherwise

o bool is_submitted()
    This method can be used to query whether a Job is cur-
    rently SUBMITTED.
      RETURNS: true if the state is SUBMITTED,
               false otherwise

o bool is_running()
    This method can be used to query whether a Job is cur-
    rently RUNNING.
      RETURNS: true if the state is RUNNING,
               false otherwise

o bool is_cancelled()
    This method can be used to query whether a Job is cur-
    rently CANCELLED.
      RETURNS: true if the state is CANCELLED,
               false otherwise
o bool is_finished()
    This method can be used to query whether a Job is cur-
    rently FINISHED.
      RETURNS: true if the state is FINISHED,
               false otherwise

o bool is_failed()
    This method can be used to query whether a Job is cur-
    rently FAILED.
      RETURNS: true if the state is FAILED,
               false otherwise

o integer job_number_by_state(integer)
    This method returns the number of all current Jobs
    with a given status.
      integer: the number of the status
      RETURNS: the number of all Jobs with the given status

o ARRAY get_job_statistic()
    This method can be used to retrieve a list of Job
    statistics. It returns a list with the number of cur-
    rent Jobs for each status.
      RETURNS: a list of Job numbers for each status
```

A.4.4. The *JobSubmitter wizard*

This section describes the API of the *JobSubmitter Wizard* that provides the basic functionality for submitting a batch of *Jobs* to the scheduler.

```
Name
       GENDB::Wizard::JobSubmitter - a wizard for submitting a
       batch of jobs

Synopsis
           use GENDB::Wizard::JobSubmitter;

           # init wizard and register progress callback
           my $wizard = GENDB::Wizard::JobSubmitter->new;
           $wizard->register_callback('progress', \&my_progress_indicator);
           $wizard->register_callback('finish', \&my_notifier);

           # create jobs to predict regions
           $wizard->regions(\@regions, \@tools);

           # create jobs to predict functions
           $wizard->functions(\@region, \@tools);

           # submits created jobs
           $wizard->submit;

Description
       This Wizard implements a generic mechanism to create Job
       objects.
```

```
Methods
       o GENDB::Wizard::JobSubmitter->new ([priority,
       restart_failed, restart_submitted, restart_finished]);
           Creates a new JobSubmitter-Wizard. The optional prior-
           ity parameter can be use can be used to set job prior-
           ities (e.g. to increase the priority for recomputing
           jobs).  restart_failed, restart_submitted and
           restart_finished may be used to restart failed, sub-
           mitted or even already finished jobs (in case of wrong
           tool configuration or other errors that occured).

       o $wizard->register_callback(name, callback sub)
           Registers a callback to be invoked in certain situa-
           tions.

           o progress
               Indicates the update of a progress widget. Invokes
               the callback with two parameters, the number of
               actions finished and the overall number of actions
               to perform.

           o finish
               Called when all actions have been performed.

       o $wizards->regions (\@regions, \@tools)
           Creates jobs to predict regions. \@regions and \@tools
           are optional arrays to restrict the wizard to certain
           regions and tools. If no regions are specified, the
           wizard will submit jobs for all Region::Source and
           <Region::Partial_Region> objects. If no tools are
           specified, all tools applicable to Region::Source and
           Region::Partial_Region are used.

       o $wizard->functions (\@regions, \@tools)
           Creates jobs to predict functions. The parameters
           \@regions and \@tools can be used as described in the
           regions() method to restrict the submitted jobs to
           certain regions and tools.

       o $wizard->submit(options, email_address)
           Submits the previously created jobs into the job
           queue. Use the options string for aditional parame-
           ters. The whole string is directly passed to the
           Scheduler.  SGE sends an email to the given address
           after all jobs have been finished if you set a valid
           email adress with the -m option.
```

A.4.5. Submitting jobs – `submit_job.pl`

This script can be used to create and submit a batch of *Jobs* to the scheduler. It provides a simple command line interface for using the *JobSubmitter Wizard*:

```
submit_job - submit a batch of jobs for a GenDB-2 project

usage: submit_job -p <project> [-c <region name>] [-t <tool name>]
                  -R|-F [-S] [-r|-s|-f] [-o] [-m user[@host]]

       -p name of GenDB project as registered in project management
       -c name of a region (e.g. a contig) in the selected project database
       -t name of a single tool stored in the GenDB database
       -R submit tools for the prediction of regions
       -F submit tools for the prediction of functions
       -S submit tools for all subregions of a given region
       -r restart all failed jobs
       -s restart all submitted jobs
       -f restart all finished jobs
          if none of -r|-s|-f is given, only submit
          jobs for new tool/region combinations
       -a activate automatic annotators
       -o additional options for SGE (e.g. \'-l arch=solaris\')
       -m email address to send a message after jobs have been finished
       -l use given SysLog facility to collect all job output
          (stdout and stderr), e.g. 'local4.info'

       PLEASE NOTE: Use explicit quoting for parameters that contain whitespaces etc.:
        e.g. 'local4.info'
```

A.4.6. The Sun Grid Engine API (Codine.pm)

The module Scheduler::Codine provides a Perl API for the Sun Grid Engine (former Codine) and implements the following class methods:

```
NAME
       Scheduler::Codine - A perl API for the Sun Grid Engine
       (Codine).

SYNOPSIS
       Submit a single job:

           use Scheduler::Codine;
           my $job = Scheduler::Codine->new();
           $job->command( "/some/path/tool $args > outfile" );
           $job->submit();

       Submit an array of jobs:

           use Scheduler::Codine;
           Scheduler::Codine->freeze();
           foreach my $cmd( @jobs ){
               my $job = Scheduler::Codine->new();
```

```
                $job->command( $cmd );
                $job->submit();
            }
            Scheduler::Codine->thaw();

DESCRIPTION
       This module provides a Perl API for the Sun Grid Engine
       (SGE). Check out http://gridengine.sunsource.net/ for pro-
       ject details of SGE. Since there does not seem to be a (do-
       cumented) API for this queueing system yet, this module was
       written in order to provide an easy-to-use Perl interface
       for submitting jobs.


       Class methods:


       o new()
           Constructor method - returns a new Codine job object.

       o freeze()
           Freezes the scheduler - collects submitted jobs in an
           array.

       o thaw()
           Thaws the scheduler - submits job array(s) to Codine.

       o options()
           Gets or sets Codine specific options (e.g. ’-l
           arch=solaris’).  See the qsub(1) manpage for a com-
           plete list of available options.

       o email()
           Gets or sets email address. Codine sends an email to
           the specified address when all submitted jobs are fin-
           ished.

       o syslog_facility()
           Gets or sets syslog facility. Codine directs both the
           output channel and the stderr channel to the syslog
           facility given here.

       Object methods:


       o command()
           Gets or sets the command. The full path to the exe-
           cutable has to be provided since Codine will run it in
           a very limited environment.

       o directory()
           Gets or sets the working directory. This defaults to
           the current working directory.

       o output()
           Gets or sets the pathname of the job’s standard output
           channel. This defaults to /dev/null. Note: this will
           not work if you set syslog_facility().
```

```
o stderr()
    Gets or sets the pathname of the job’s standard error
    channel. This defaults to /dev/null. Note: this will
    not work if you set syslog_facility().

o queue()
    Gets or sets the queue. Since Codine defines queues
    per execution host we decided to provide 3 (host inde-
    pendent) queues: -1, 0, 1.  -1 corresponds to a low
    priority queue, 0 to a queue with medium priority and
    1 to a high priority queue. The default queue is 0.
    The specified queue is combined with the given prior-
    ity() value (see below) to an SGE specific priority
    value.

o priority()
    Gets or sets the priority. This value must be between
    -64 and 63 and defaults to 0.

o submit()
    Submits job to Codine. Depending on the state of the
    scheduler (frozen after using the freeze() method or
    thawn otherwise) the submitted job will be either col-
    lected in a job array or directly submitted to SGE as
    a single job.
```

A.5. BRIDGE modules

The following sections describe the most important basic modules implemented for the BRIDGE platform.

A.5.1. BridgeFunc

```
Name
       BridgeFunc - an interface for integrating heterogeneous
       data sources

Synopsis
        use BridgeFunc;

        # construct a GPMS::ApplicationFrame::GPMS object
        my $bridgefunc = BridgeFunc->new($gpms_frame);

        # or write alternatively
        my $bridgefunc = BridgeFunc->new($gpms_frame, 'my_local_namespace');

        # register the Application_Frames

        # for the main Application_Frame:
        $bridgefunc->register_AppFrame('my_local_namespace',$my_appframe);

        # and for further Application_Frames that should be used, e.g.:
        $bridgefunc->register_AppFrame_Type('GPMS::Application_Frame::GENDB',
                                            'GenDB');

        # initialize an internal object that refers to an external object
        my $internal_object = MyClass->init_id(123);

        # access an external object
        my $external_object = $internal_object->ext_ref();
        if (ref $external_object) {
            # do some stuff with the initialized external object
        }
        else {
            # do some error handling
            # either the lookup failed or the external reference was not set
        }

Description
       This BridgeFunc module provides some special functions
       implementing the BRIDGE layer on top of the O2DBI server
       classes for the integration of heterogeneous data sources.
       It is responsible for storing and managing different
       Application_Frame objects that are associated with a spe-
       cific namespace. In addition to this, the methods provided
       by this module can be used to obtain URIs for referencing
       objects and for retrieving such external objects.
```

```
Application management

Internally all applications are registered and managed
using their corresponding Application_Frame objects.
Thereby, an application is associated with a namespace
that uniquely identifies their data source.

Object management

The main purpose of the BridgeFunc layer is the handling
of inter-application references. This includes the identi-
fication and initialization of objects and the automatic
handling of Application_Frames for accessing such objects.
Although most of the work is done transparently, the meth-
ods used are also available for explicitly requesting
objects and URIs (see below).

URIs

URIs (Unified Resource Identifiers) are used in the BRIDGE
system for referencing objects across different data
sources (e.g. different databases).  These objects may be
part of other projects, projects of different application
types or even projects running on another server. The for-
mat (syntax) of these special URIs is defined as follows:

"o2xr://<namespace>/<projectname>/<datasourcetype>?uid"

o o2xr
    This is a simple name for the BRIDGE naming scheme
    (derived from O2DBI eXternal Reference).

o namespace
    The namespace denotes an unique identifier for an
    installation or instance of the General Project-Man-
    agement System (GPMS) that is used for managing all
    kinds of projects.  Internally, all external GPMS sys-
    tems have to be registered in the local GPMS database
    as a special kind of project entry so that remote con-
    nections can be established. When an URI is resolved,
    the local GPMS database is queried for the information
    that is required for accessing other (remote) GPMS
    systems. As an example, a namespace could be the name
    of an institute such as CeBiTec.Uni-Bielefeld.DE.

o projectname
    This is an identifier for the project in the scope of
    the namespace. The same project name may be used in
    different namespaces, but the combination of namespace
    and project has to be unique.

o datasourcetype
    Projects may be composed by combining different kinds
    of data sources (e.g. different O2DBI dataschemes for
    different databases) but each project can only have
    one data source of each different data source type.
    Since objects are local to data sources, their unique
    ids are local, too.
```

```
       o uid
           The unique id of an object.

       Namespace, projectname and datasourcetype identify the
       database that has to be used. The unique id thus refers to
       a single object in that database.

       Accessing local and remote GPMS

       Since internal objects can refer to external objects
       stored on a remote systems, the BridgeFunc layer has to be
       able to resolve remote server names. Instead of using
       classical (global) resolving service like DNS, a local
       instance of the GPMS is used to resolve remote GPMS sys-
       tems. This local GPMS is managed as a specialisation of
       the project class in the implementation of the GPMS, which
       allows the full range of possible configurations (differ-
       ent access ways, redundant systems etc.). Thus the Bridge-
       Func constructor expects the local GPMS and its namespace
       as parameters. This GPMS instance will then be used to
       lookup all namespaces that do not match the local names-
       pace identifier given as an optional parameter. Conse-
       quently a BridgeFunc application has to know its local
       namespace and the O2DBI master for the local GPMS system
       has to be constructed before the BridgeFun layer can be
       used.

       Using BridgeFunc

       Most of the functions of the BridgeFunc layer were imple-
       mented as static methods.  Nevertheless, the BridgeFunc
       module can be used in an object oriented way which also
       provides a proper cleanup of the environment (especially
       important when using mod_perl).

       After creating a BridgeFunc object external reference
       attributes can be used transparently and in the same man-
       ner as objects from other registered Application_Frames.
       If the lookup of an object fails, the getter method
       returns the URI itself. Thus the return value should
       always be checked (see Synopsis for further details).

Methods
       o new(<GPMS Application_Frame>, [string])
           Default constructor method for creating a new Bridge-
           Func object.
               GPMS ApplicationFrame: this Applica-
           tion_Frame::GPMS object is used for accessing the
           "local" GPMS
               string:               optional name of the local
           namespace; if no name is give, the name of the
                                     project associated to the
           GPMS Application_Frame is used

       o bool register_AppFrame(string, <Application_Frame>)
           This method explicitly registers a new Applica-
           tion_Frame object.
```

```
          string:           name of the namespace
          ApplicationFrame: the Application_Frame object
          RETURNS:          true if the namespace was regis-
        tered correctly, false otherwise, e.g. if the names-
        pace was already registered

o void register_AppFrame_Type(string, string)
        The kind of an Application_Frame used for a specific
        data source depends on the data source type that is
        required since the BridgeFunc layer does not have a
        priori knowledge about the corresponding application
        that has to be instantiated internally (using the
        "get_Object" method described below). Thus it has to
        maintain a mapping of data source types to its corre-
        sponding Application_Frame classes.
          string: name of an Application_Frame module
          string: the name of the data source type

o void remove_AppFrame(string, string)
        Removes the Application_Frame that has been registered
        for the given namespace and the given project.
          string: name of the namespace
          string: name of the project that corresponds to an
        Application_Frame

o <Application_Frame> get_AppFrame(namespace, project
name)
        This method returns the Application_Frame that has
        been registered for a given namespace and a given pro-
        ject.
          string:  the name of a namespace
          string:  the name of the project
          RETURNS: an Application_Frame object for the given
        namespace and project, undef otherwise,
            e.g. if the namespace or project was not regis-
        tered before

o  get_namespace_project(Application_Frame / O2DBI II mas-
ter)
        This method returns an array containing the namespace
        and project name of a Application_Frame or O2DBI II
        master object. If the Application_Frame or master
        object is not registered with BridgeFunc, undef is
        returned.

o string get_URI(<OBJECT>)
        This method generates the URI for a given object. The
        class of the object has to be marked as a referable
        class in the O2DBI dataschema.
          OBJECT:  any O2DBI object that can be referenced
        externally
          RETURNS: the complete URI for the given object if it
        can be referenced externally, an empty string other-
        wise

o OBJECT get_Object(string)
        This method tries to resolve a given URI and returns
        the object referenced by the URI.  Internally, this
```

```
           method creates a new Application_Frame object if this
           is necessary for accessing an external database.
             string:  the URI that refers to an external object
             RETURNS: the requested object if it was initialized
           successfully, undef otherwise
```

A.5.2. BridgeFunc::Projects

```
Name
       BridgeFunc::Projects - simple class for handling projects
       that are managed by the GPMS

Description
       Simple module for handling applications using the BRIDGE
       functionality. All projects that belong to a specific
       namespace can be managed by this module.

Methods
       o new(<GPMS>)
           Default contructor method for creating a new Projects
           object
             <GPMS>: reference on GPMS::Application_Frame

       o GPMS::Application_Frame gpms()
           Get the current GPMS::Application_Frame that corre-
           sponds to this object.
             RETURNS: GPMS::Application_Frame

       o void add_project(string, <Project>)
           Add a new project to this project handler.
             string:  name of the project
             Project: the project itself

       o void remove_project(string)
           Remove the project with the given name.
             string: the name of the project that should be
           removed

       o <Project> get_project(string)
           Get the project that corresponds to a given name.
             string: the name of the project

       o string query_project([Application_Frame|O2DBI_Master])
           Get the project name for a given Application_Frame or
           an O2DBI_Master.

       o void destroy()
           Explicit destructor method for objects of this class.
           Remove all projects and unset the GPMS::Applica-
           tion_Frame
```

A.5.3. BridgeFunc::Namespaces

```
Name
       BridgeFunc::Namespaces - a simple module for managing
       namespaces used in BRIDGE applications

Description
       Helper module to handle namespaces

Methods
       o new(<GPMS>, <Namespace>)
           Default contructor method for creating a new Names-
           paces object
             <GPMS>:       reference on GPMS::Application_Frame
             <Namespace> : the name of the namespace

       o <GPMS> local_gpms()
           Method for retrieving the ApplicationFrame::GPMS of
           the local installation.
             RETURNS: the local ApplicationFrame::GPMS

       o bool query_namespace(string)
           Method for checking if a namespace is defined or not.
             string:  the name of a namespace
             RETURNS: true if namespace is registered, false oth-
           erwise

       o ARRAYREF projects(string)
           Get all projects for a given namespace.
             string:   the name of the namespace
             ARRAYREF: list of all projects

       o ARRAYREF query_project(string, string)
           Check if a project has been registered for a names-
           pace.
             string:  the name of the namespace
             string:  the name of the project
             RETURNS: the project if it is available, undef oth-
           erwise

       o ARRAY query_namespace_project([Applica-
       tion_Frame|O2DBI_Master])
           This method returns the namespace and the project name
           for a given Application_Frame or an O2DBI_Master.

       o void add_project(string, <Application_Frame>)
           Add a project to an existing namespace.
             string:            the name of the namespace
             Application_Frame: the corresponding Applica-
           tion_Frame of the project

       o void remove_project(string, string)
           Remove the Application_Frame of a project from a
           namespace.
             string: the name of the namespace
             string: the name of the project that should be
           removed
       o void set_namespace(string, <GPMS>)
           Set a new namespace handler.
             string: the name of the namespace
             GPMS:   Application_Frame::GPMS that should be set
           for the given namespace

       o <GPMS> get_GPMS(string)
           Get the Application_Frame::GPMS for a given namespace.
           This method will also try to construct a new Applica-
           tion_Frame::GPMS if none exists.
             string:  the name of the namespace
             RETURNS: the Application_Frame::GPMS if available,
           undef otherwise
```

A.5.4. BridgeFunc::AppFrames

```
Name
       BridgeFunc::AppFrames

Description
       This class can be used for managing Application_Frames in
       the BridgeFunc layer.  This module keeps track of the
       associations between Application_Frames and their corre-
       sponding data sources (as they are used by the GPMS).  It
       also processes the O2DBI master modules and overwrites the
       attribute handlers for classes that contain external ref-
       erences.

Methods
       o new()
           Default contructor method for creating a new AppFrames
           object

       o string <Application_Frame> get_appframe_class(string)
           Get a class of ApplicationFrames.
             string:  the name of a data source type
             RETURNS: the class name of an Application_Frame

       o void add_ds_type(string, <Application_Frame>, bool)
           Add a new data source type.
             string:            name of the data source type.
             Application_Frame: an Application_Frame object
             bool:              flag to indicate whether external
           references should be resolved or not

       o string get_ds_type(<Application_Frame>)
           Get the data source type for a given Applica-
           tion_Frame.
             ApplicationFrame: an Application_Frame object
             RETURNS:          the name of the data source type
           that corresponds to a given Application_Frame
```

A.5.5. StatusWidget

```
Name
       GUI::StatusWidget - abstract base class for widgets used
       in the BRIDGE framework

Description
       A StatusWidget provides a general framework for implement-
       ing graphical user interfaces with Gtk, e.g. as special-
       ized components of the BRIDGE system. It is an abstract
       widget with status message signals.

Concepts
       Basically, a StatusWidget is an extended container which
       provides some common functionality that is frequently
       needed in many GUI applications.

       o Widget Hierarchy:
            Gtk::Object
            |
            +-Gtk::Widget
            | |
            . +-Gtk::Container
            .   |
            .   +-Gtk::Box
                  |
                  +-Gtk::VBox
                    |
                    +-GUI::StatusWidget

       Since a StatusWidget is a subclass of a Gtk::VBox, derived
       widgets of this class can be nested and packed into each
       other. Signals that are emitted by a nested StatusWidget
       will be passed back through all parent StatusWidgets until
       they are received by a toplevel widget which handles the
       emitted signal. As an example the signal message could be
       connected to a status bar of the main window in order to
       display messages.

Signals
       o message (GUI::StatusWidget, string)
           Emitted when new message is sent
            GUI::StatusWidget -> the StatusWidget object
            string            -> the message

       o init_progress (GUI::StatusWidget, int)
           Emitted when new progress starts
            GUI::StatusWidget -> the StatusWidget object
            int               -> total number of computations to
           be done

       o update_progress (GUI::StatusWidget, int)
           Emitted every computation loop
            GUI::StatusWidget -> the StatusWidget object
            int               -> current computation number

       o end_progress (GUI::StatusWidget)
```

```
           Emmited when computations are finnished
            GUI::StatusWidget -> the StatusWidget object

       o change_cursor (GUI::StatusWidget, int)
           Emmited when cursor has to be changed
            GUI::StatusWidget -> the StatusWidget object
            int               -> number of the cursor

Methods
       o void init (CLASS)
           Method to register a StatusWidget subclass as a Gtk
           class if this has not been done already. This step is
           required to access some Gtk internal functions and
           make a new widget work. This method has to be called
           directly in the constructor method of a subclass (in
           most cases in the implementation of the method ’new’).

       void destroy()
           Extended Gtk destroy method for destroying also all
           subwidgets.

       o void register_subwidget(GUI::StatusWidget, GUI::Sta-
       tusWidget)
           This method has to be called to register a StatusWid-
           get as a sub-widget of a parent StatusWidget. Only
           calling this method will enable the functionality of a
           StatusWidget described above.

       o get_subwidgets
           This method returns a list of all sub-widgets.

       o void clear_subwidgets()
           Clear the list of all sub-widgets.

       o void remove_subwidget($widget)
           Remove a given widget and all its sub-widgets from the
           list of all sub-widgets.

       o void print_subwidgets()
           Debug method to print out all sub-widgets recursively.

       o ARRAYREF get_subclasses()
           This methods returns a list of all class names of the
           sub-widgets.

       o void add_config_classes(HASHREF, ARRAY)
           This method can be used to register sub-classes of the
           ConfigurationInterface.
             HASHREF: contains additional data used in the Con-
           figurationInterface, e.g.
               a backend for storing the configuration (see also
           UserConfig)
             ARRAY: list of class names

       Configuration Concept:

       In order to create a configuration frontend for an imple-
       mented StatusWidget the following steps are required:
```

```
    1) Implement a sub-class of the GUI::ConfigurationIn-
    terface that contains the GUI elements for editing
    attributes. For further details see the documentation
    of the GUI::ConfigurationInterface.
    2) Invoke this method in the constructor of the Sta-
    tusWidget with the class name(s) of the corresponding
    ConfigurationInterface(s). Thereby, the StatusWidget
    is registered as a widget that has to be notified upon
    changes in this/these ConfigurationInterface(s). At
    the same time the ConfigurationInterface(s) is/are
    added to the list of config classes which are shown in
    the ConfigDialog for the current application.
    3) Implement the "apply" method in the StatusWidget
    which is called upon changes of the configuration.
    This method is called with a hash argument containing
    all changes of the configuration. Thus the "apply"
    method has to update for example some elements of the
    GUI.
o void get_config_classes(string)
    Return a list of all registered classes of Configura-
    tionInterface.
      string: optional name of a group of configuration
    classes

o void add_tooltips(Gtk::Widget, string, string)
    Add a tooltip for any widget.
      Gtk::Widget: a widget for which the tooltip should
    be set
      string: the text itself shown in the tooltip or if a
    second string argument is given
        this can be the name of an application (see Hel-
    pRepository)
      string: key of a section for an application docu-
    mented in the HelpRepository

o void toggle_tooltips (GUI::StatusWidget, bool)
    Toggle the visualization of all tooltips on or off
    recursively for all sub-widgets.

o void stop_progress()
    Set the interrupt signal in order to stop a time con-
    suming operation.

o bool _stop_progress()
    Check the interrupt signal if a time consuming opera-
    tion should be stopped.

o LIST get_menu()
    This method has to be implemented in the sub-classes
    of a StatusWidget in order to define individual menu
    entries. This method has to be called in the imple-
    mented ’get_menu’ method itself in order to get the
    menus of the subwidgets.

o void set_attribute(string, SCALAR)
    Set an internal attribute for an a StatusWidget.
      string: name of a key for the attribute
```

```
             SCALAR: any kind of value

       o SCALAR get_attribute(string)
           Get an internal attribute of a StatusWidget.
             string: name of an attribute key
             RETURNS: scalar value of requested attribute

       o SCALAR get_attributes(string)
           Get an internal attribute of a StatusWidget.  This
           method is only provided for convenience.
             string: name of an attribute key
             RETURNS: scalar value of requested attribute

       o void copy_attributes(GUI::StatusWidget)
           Copy all attributes from this StatusWidget to another
           StatusWidget.
             GUI::StatusWidget: a StatusWidget that receives the
           attributes
```

A.5.6. MenuCreator

```
Name
       GUI::MenuCreator - a dynamic menubar

Description
       The MenuCreator provides a dynamic menu bar that can be
       modified on demand.

Concepts
       Standard Gtk::MenuBar widgets can’t be modified using
       Gtk::ItemFactory. Thus this class provides the requried
       functionality to change such menu bars dynamically.

       o Widget Hierarchy:
            Gtk::Object
            |
            +-Gtk::Widget
            | |
            . +-Gtk::Container
            .   |
            .   +-Gtk::Box
                  |
                  +-Gtk::HBox
                    |
                    +-GUI::MenuCreator

Methods
       o new
           Constructor method for creating a new MenuCreator.

       o void set_menu(Gtk::Window, ARRAYREF)
           This method re-builds the menu bar from the descrip-
           tion of a Gtk::ItemFactory.
             Gtk:Window: a window required for setting accelera-
           tors
             ARRAYREF: the description for the Gtk::ItemFactory
           In addtion to those items and options defined for the
           standard Gtk::ItemFactory, the following key words can
           be used in the description:
             default: initially activate check menu items
             disabled: disable single menu items
             get_widget: provide access to the widget represent-
           ing a menu item
```

A.5.7. ContextMenuInterface

```
Name
       GUI::ContextMenuInterface - a framework for building con-
       text sensitive menus

Description
       The ContextMenuInterface is a simple interface (not a
       class!) that provides some basic functionality for context
       sensitive menus.

Concepts
       Context sensitive menus can be opened and build depending
       on the context that they were openend for. Modules using
       this interface have to implement the method ’_open_menu’.

Methods
       o void open_menu(SCALAR, Gdk::Event)
           This method simply opens a menu and pops it up onto
           the screen.
             SCALAR: context that is propagated to the context
           menu (e.g. an object such as a CDS)
             Gdk::Event: this event is required for controling
           the menu behaviour

       o void add_extern_menu_creator(string, CODEREF)
           Add an external menu creator.
             string: name of the menu creator
             CODEREF: method for building the menu

       o void remove_extern_menu_creator(string)
           Remove an external menu creator described by the given
           name.
             string: name of the menu creator
```

A.5.8. PopoutBook

```
Name
       GUI::PopoutBook - Notebook with pages that can pop out

Synopsis
        use Gtk;
        use GUI::PopoutBook;

        init Gtk;
        ...
        my $widget = new GUI::PopoutBook;
        $widget->append_page(Tab-Text, Widget);
        ...
        main Gtk;

Description
       Set of pages with bookmarks, pages can pop out into a win-
       dow.

Widget Hierarchy
        Gtk::Object
        |
        +-Gtk::Widget
        | |
        . +-Gtk::Container
        .   |
        .   +-Gtk::Bin
              |
              +-Gtk::Notebook
                |
                +-GUI::PopoutBook

Signals
       o page_pop_out (int, Gtk::Widget)
           Emitted when the user or a function pops the page into
           a window.
             GUI::PopoutBook: the PopoutBook object
             int:             the page number
             Gtk::Widget:     widget that is popped out

       o page_pop_in (int, Gtk::Widget)
           Emitted when the user or a function kills a window and
           pops the widget back into a PopoutBook page.
             GUI::PopoutBook: the PopoutBook object
             int:             the page number
             Gtk::Widget:     Widget that is popped in

       o page_toggled (int, Gtk::Widget, int)
           Emitted when the user or a function toggles a page
           (either pop out or pop in).
             GUI::PopoutBook: the PopoutBook object
             int:             the page number
             Gtk::Widget:     widget that is popped in or out
             int:             1 = page was popped out/0 = page
           popped in

Methods
       o new()
           Default constructor method for creating a new Popout-
           Book widget.

       o int append_page(string, Gtk::Widget)
           Appends a new page
             string:      the text in the tab
             Gtk::Widget: the child widget for the new page
             RETURNS: the number of the position where the new
           page was added
```

```
o append_page_menu(string, Gtk::Widget, string)
    Appends a new page and inserts the corresponding menu
    item
      string:      the text in the tab
      Gtk::Widget: the child widget for the new page
      string:      text for the menu item

o insert_page(string, Gtk::Widget, int)
    Inserts a new page at the specified position.
      string:      the text in the tab
      Gtk::Widget: the pages child widget
      int:         the position to insert the page

o insert_page_menu(string, Gtk::Widget, string, int)
    Inserts a new page and the corresponding menu item
    into a PopoutBook
      string:      the text in the tab
      Gtk::Widget: the pages child widget
      string:      the text displayed in the menu item
      int:         the position to insert the page

o destroy()
    Remove all pages and destroy their parent PopoutBook.
    Overloaded Gtk method to ensure that all pages are
    destroyed.

o remove_page(int)
    Remove a given page from the PopoutBook.
      int: page number

o remove_pages()
    Remove all pages from the PopoutBook.

o get_pos_by_widget(Gtk::Widget)
    Get the pagenumber for a given widget.
      Gtk::Widget: a widget that has been inserted into
    the PopoutBook
      RETURNS:     the page number on which the given wid-
    get was inserted, -1 otherwise

o set_text(int, string)
    change the text label of a given tab
      int:    page number
      string: new text

o set_popout_mode(enum)
    Set the popout mode.
      enum: values are (’all’, ’one’)
            all means, all pages can pop out
            one means, one page is kept

o toggle_window(int, bool)
    Change the state of a page and switch between popped
    out and popped in
      int:  the page number to toggle
      bool: if true put the widget into a window,
            else put it back into a page
```

```
       o all_to_window()
           Pop all pages out into separate windows.

       o pop_all_back()
           Close all windows and put their content back into the
           notebook.

       o menu_popup(Gdk::Event)
           Show the menu at any place.
             Gdk::Event: a standard Gdk event

       o set_popout_active(int, bool)
           Set the tab activity for a single page and enable
           popout of one page.
             int: which page
             bool: enable/disable

       o set_popouts_active(bool)
           Set the tab activity for all pages and enable popup of
           all pages.
             bool: enable/disable

       o use_individual_config(bool)
           Each page that was popped out into a window can have
           its own size configuration (see GUI::DialogWindow).
             bool: enable/disable
```

A.5.9. ConfigurationInterface

```
Name
       GUI::ConfigurationInterface - A general interface for
       implementing configuration frontends.

Synopsis
         package Foo;
         @ISA = "GUI::ConfigurationInterface";

         sub config_box {
             # create config box
             return $config_box;
         }

         sub get_config_hash {
             # create config_hash
             return \%config_hash
         }

         sub register_userconfig {
             # register UserConfig
         }

         sub config_name {
             return $my_name;
         }
```

```
Description
       Interface class for all user configuration frontends. A
       ConfigurationInterface can be used to define GUI widgets
       for editing configurable attributes. Widgets for config-
       urable attributes are observed so that changes are regis-
       tered and returned. Changes are also propagated to all
       StatusWidgets that are registered for a ConfigurationIn-
       terface.

Widget Hierarchy
         Gtk::Object
         |
         +-Gtk::Widget
         | |
         . +-Gtk::Container
         .   |
         .   +-Gtk::Box
               |
               +-Gtk::VBox
                 |
                 +GUI::ConfigurationInterface

Signals
       o saved()
           This signal is emitted when changes in a Configura-
           tionInterface have been saved.  It is used in the Con-
           figurationDialog to switch the Apply button.

       o changed()
           This signal is emitted when some configuration entity
           was changed.  It is used in the ConfigurationDialog to
           switch the Apply button.

Methods
       The implementation of a ConfigurationInterface requires
       the definition of four abstract methods:

       o Gtk::Widget config_box()
           Abstract method which defines the basic GUI widget
           which contains all widgets for editing the config-
           urable attributes. It is packed into a Gtk::VBox of
           the ConfigurationInterface.  All editable widgets have
           to be observed for changes by using the "observate"
           method.
             RETURNS: Gtk::Widget, the GUI container widget

       o get_config_hash(HASHREF)
           Abstract method that has to be implemented for saving
           the changes. Normally it returns the (modified) hash
           of observed values. This method can be used as a fil-
           ter for mapping internally used attribute names onto
           global attributes.
               HASHREF: hash of observed values
               RETURNS: hashref, changed values

       o register_userconfig()
           Abstract method which registers the UserConfig.
```

```
o config_name ()
    Abstract method that defines and returns the name of
    the ConfigurationInterface.


The ConfigurationInterface also provides a number of meth-
ods for the interaction with its corresponding Configura-
tionDialog:

o new(attributes)
    Constructor method for creating a new ConfigurationIn-
    terface object.
      attributes: a list of attributes

o add_subwidget(GUI::StatusWidget, string)
    Add a GUI::StatusWidget object. All changes in this
    StatusWidget will be observed by the current Configu-
    rationInterface.
      GUI::StatusWidget: a new widget that will be further
    observed by the current ConfigurationInterface
      string: a name for this instance of the Configura-
    tionInterface

o clear_subwidgets()
    Use this method to clear all subwidgets of a Configu-
    rationInterface.

o apply(string)
    Apply all changes and send them to the subwidgets.
      string: name of configuration section

o observate(Gtk::Widget, string, [ string | CODE ])
    Observate a widget and propagate all changes to the
    ConfigurationInterface.
        Gtk::Widget: the widget to observate
        string:      name of a signal that is emitted when
    a widget was changed
        string:      the name of the changed attribute
        CODE:        run subroutine to get the name if a
    single widget (e.g. a tree) can
                     be used to modify several attributes

o changed()
    Check if any configurations were changed and therefore
    have to be saved.

o set_attribute(key, value)
    Set the value of an attribute.
      key: the name of the attribute
      value: the new value for the attribute

o get_attribute(key)
    Get the current value of an attribute.
      key: the name of the attribute for which the value
    should be retrieved

o get_attributes(key)
    Get the current value of an attribute.
```

```
             key: the name of the attribute for which the value
           should be retrieved

       o copy_attributes(widget)
           Copy the attributes of one widget to another one.
             widget: a Gtk widget that will obtain the attributes
```

A.5.10. ConfigurationDialog

```
Name
       GUI::ConfigurationDialog - A dialog to manage Configura-
       tionInterfaces.

Synopsis
         use Gtk;
         use GUI::ConfigurationDialog;

         init Gtk;

         ...
         my $dialog = new GUI::ConfigurationDialog;
         $dialog->set_config_classes(@class_names);
         ...

         main Gtk;

Description
       The ConfigurationDialog is a simple dialog window for man-
       aging ConfigurationInterfaces. Each ConfiguratioInterface
       is packed into a separate page of a Gtk::Notebook. When
       the user modifies the settings of a configurable attribute
       the ConfigurationDialog is informed about these changes in
       the current ConfigurationInterface (a single page or sec-
       tion of the notebook). After accepting a new configuration
       the settings are stored and propagated to all registered
       StatusWidgets of a ConfigurationInterface where they can
       be applied to all affected GUI elements

Widget Hierarchy
        Gtk::Object
        |
        +-Gtk::Widget
        | |
        . +-Gtk::Container
        .   |
        .   +-Gtk::Bin
              |
              +-Gtk::Window
                |
                +-GUI::DialogWindow
                  |
                  +-GUI::ConfigurationDialog

Signals
       So far no special signals were defined for this class.
```

```
Methods
       o new(attributes)
           Constructor method for creating a new ConfigurationDi-
           alog object.
             attributes: a list of attributes

       o show()
           Show this ConfigurationDialog and all sub widgets con-
           tained therein.

       o objects()
           Class method: return the number of current objects
               RETURN: number of current objects

       o set_config_classes(@strings)
           Set the configuration classes that should be used. All
           classes that are added have to be subclasses of
           GUI::ConfigurationInterface.
               @strings: list of classnames

       o apply_config()
           Apply all changes for the current configuration inter-
           face.

       o set_attribute(key, value)
           Set the value of an attribute.
             key: the name of the attribute
             value: the new value for the attribute

       o get_attribute(key)
           Get the current value of an attribute.
             key: the name of the attribute for which the value
           should be retrieved

       o copy_attributes(widget)
           Copy the attributes of one widget to another one.
             widget: a Gtk widget that will obtain the attributes

       o close()
           Check if something was changed and has to be saved.
           Close the dialog afterwards.
```

A.5.11. InterfaceCreator

```
Name
       GUI::InterfaceCreator - Create GUIs from simple hash
       descriptions.

Synopsis
           use Gtk;
           use GUI::InterfaceCreator;

           use Data::Dumper;

           init Gtk;
```

```
my $description = [

                  {
                      type       => ’label’,
                      name       => ’label’,
                      label      => ’A Label’,
                      default    => ’label’,
                      optional   => 1,
                      add_entry  => sub { return time(); },
                      add_name   => "change Label",
                  },

                  {
                      type       => ’string’,
                      name       => ’string’,
                      default    => ’string’,
                      input_type => 0,
                      max        => 10,
                      editable   => 1,
                  },

                  {
                      type       => ’int’,
                      name       => ’int’,
                      default    => 100,
                      max        => 100,
                      min        => -100,
                  },

                  {
                      type       => ’float’,
                      name       => ’float’,
                      default    => 1.3233,
                      min        => -1.1111,
                      max        => 1.3255,
                      digits     => 4,
                  },

                  {
                      type       => ’list’,
                      name       => ’list’,
                      list_vals  => [qw(item1 item2 item3 item4 item5)],
                      default    => ’item3’,
                  },

                  {
                      type       => ’file’,
                      name       => ’file’,
                      default    => $ENV{HOME}."/file.txt",
                  },

                  {
                      type       => ’color’,
                      name       => ’color’,
                      default    => ’green’,
                  },

                  {
```

```
        type       => 'font',
        name       => 'font',
        default    => "-bitstream-courier-*-r-*-*-*-*-*-*-*-*-*-*",
    },

    {
        type       => 'separator',
        name       => 'separator',
    },

    {
        type       => 'text',
        name       => 'text',
        default    => "A multiline\nText!",
        font       => "-bitstream-courier-*-r-*-*-*-*-*-*-*-*-*-*",
        width      => 400,
        height     => 100,
    },

    {
        type       => 'switchlist',
        name       => 'switchlist',
        default    => [["item4a", "item4b"]],
        source     => [
                        ["item1a","item1b"],
                        ["item2a","item2b"],
                        ["item3a","item3b"]
                      ],
        titles     => ["Source1", "Source2"],
        titles2    => ["Dest1", "Dest2"],
        hide       => [0],
        width      => 400,
        height     => 100,
    },

    {
        type       => 'array',
        name       => 'array',
        titles     => ["Column1", "HiddenColumn2", "Column3"],
        default    => [
                        ["item1a", "item1b", "item1c"],
                        ["item2a", "item2b", "item2c"],
                        ["item3a", "item3b", "item3c"],
                      ],
        hide       => [1],
        width      => 400,
        height     => 100,
        on_click   => sub { print Dumper @_; },
    },

    {
        type       => 'multiplelist',
        name       => 'multiplelist',
        titles     => ["Column1", "HiddenColumn2", "Column3"],
        list_vals  => [
                        ["item1a", "item1b", "item1c"],
                        ["item2a", "item2b", "item2c"],
                        ["item3a", "item3b", "item3c"],
                                           ],
                              default    => [["item2a", "item2b", "item2c"]],
                              hide       => [1],
                              width      => 400,
                              height     => 100,
                              editable   => 0,
                          },

                      ];

           my $imaker = new GUI::InterfaceCreator;
           my $widget = $imaker->make_interface($description);

           my $window = new Gtk::Window('toplevel');
           $window->add($widget);

           $window->show_all;

           Gtk->main_iteration while($window->visible);

           my $result = $imaker->get_result;
           print Dumper $result;

           Gtk->exit(0);

Description
       Create graphical user interfaces (GUIs) from simple tex-
       tual description.  The GUI description has to be provided
       as a list of hashes where each hash describes a single
       widget. The InterfaceCreator is especially useful for sim-
       ple user interfaces and for rapid prototyping of Gtk fron-
       tends.  The current implementation provides the most fre-
       quently used widgets and their most important properties
       which have to be specified in the hash keys.

       Available hashkeys (widgets) and their corresponding value
       types are:

       o separator - create a Gtk::Separator
                   name       (string REQ) - internally used name

       o label - create a Gtk::Label
                   name       (string REQ) - internally used name

                   label      (string OPT) - the label shown on the left side
                                             if not set, the name is used

                   default    (string OPT) - the value shown on the right side

                   optional   (bool   OPT) - create a Gtk::Checkbox

                   add_entry  (CODE   OPT) - create a Gtk::Button, if clicked
                                             CODE is called and the lable
                                             is changed to the return value

                   add_name   (string OPT) - only used with 'add_entry',
                                             set the Gtk::Button text
```

```
o string - create a Gtk::Entry
            name       (string REQ) - internally used name

            label      (string OPT) - the label shown on the left side
                                      if not set, the name is used

            default    (string OPT) - the default value shown
                                      in the Gtk::Entry

            optional   (bool   OPT) - create a Gtk::Checkbox

            add_entry  (CODE   OPT) - create a Gtk::Button, if clicked
                                      CODE is called and the lable
                                      is changed to the return value

            add_name   (string OPT) - only used with ’add_entry’,
                                      set the Gtk::Button text

            input_type (bool   OPT) - if set to 0, the text is invisible

            max        (int    OPT) - maximum text length

            editable   (bool   OPT) - set the Gtk::Entry editable

o int - create a Gtk::SpinButton
            name       (string REQ) - internally used name

            label      (string OPT) - the label shown on the left side
                                      if not set, the name is used

            default    (string OPT) - the default value shown
                                      in the Gtk::Entry

            optional   (bool   OPT) - create a Gtk::Checkbox

            add_entry  (CODE   OPT) - create a Gtk::Button, if clicked
                                      CODE is called and the lable
                                      is changed to the return value

            add_name   (string OPT) - only used with ’add_entry’,
                                      set the Gtk::Button text

            input_type (bool   OPT) - if set to 0, the text is invisible

            max        (int    OPT) - maximum value

            min        (int    OPT) - minimum value

            editable   (bool   OPT) - set the Gtk::SpinButton editable

o float - create a Gtk::SpinButton
            name       (string REQ) - internally used name

            label      (string OPT) - the label shown on the left side
                                      if not set, the name is used

            default    (string OPT) - the default value shown
                                      in the Gtk::Entry
```

```
                   optional   (bool   OPT) - create a Gtk::Checkbox

                   add_entry  (CODE   OPT) - create a Gtk::Button, if clicked
                                             CODE is called and the lable
                                             is changed to the return value

                   add_name   (string OPT) - only used with 'add_entry',
                                             set the Gtk::Button text

                   input_type (bool   OPT) - if set to 0, the text is invisible

                   max        (int    OPT) - maximum value

                   min        (int    OPT) - minimum value

                   editable   (bool   OPT) - set the Gtk::SpinButton editable

                   digits     (int    OPT) - number of digits to show

               ...

Widget Hierarchy
         Gtk::Object
         |
         +-GUI::InterfaceCreator
         |
         +-...

Methods
       o new()
         Constructor method for creating a new InterfaceCreator object.

       o make_interface(description)
         Create the graphical user interface for a given hash description.
           description: hash description for the GUI, see sec-
         tion 'Description' above
           RETURN: Gtk::Widget, container widget

       o set({keys => values})
         Change a certain widgets by specifying key/value pairs
         in a hash.
           keys: names of the widgets
           values: new values

       o wait()
         Wait for user input.

       o get_widget(string)
         Get a certain widget that is specified by its name.
             string: the name of the requested widget
             RETURN: the Gtk::Widget

       o get_result(name)
         Return value of widget name ???  if value is not
         defined return values of all widgets
             name: the widgets name
             RETURN: values
```

A.6. Description of "Common" modules

The CVS tree of the BRIDGE system contains some general purpose tools that can be used by all components. Since these modules provide simple commonly used functions they are installed in a separate directory. All currently available modules are listed below in alphabetical order:

- **AAUtils.pm**
  This module contains a collection of useful methods for handling amino acid sequences, e.g. for calculating the molecular weight of a given sequence.

- **AsynchronStarter.pm**
  The *AsynchronStarter* package can be used for starting concurrent processes. It provides a simple interface to the `fork` system call.

- **CSV.pm**
  The *CSV* module provides generic methods for parsing and writing general spreadsheet files like comma or tab delimited files.

- **DNAUtils.pm**
  This module contains some useful functions to handle DNA sequences, e.g. for translating a DNA sequence or for computing the reverse complement.

- **FastaUtils.pm**
  This package provides simple methods for reading and writing files in FASTA (Pearson) format.

- **FileStorage.pm**
  The *FileStorage* module can be used to manage a file repository, e.g. of large images. It handles all read and write accessess and implements a locking strategy.

- **FileUtils.pm**
  The package *FileUtils.pm* provides some comfortable functions for handling files and directories, e.g. set permissions upon creation, remove all data recursively.

- **HelpRepository.pm**
  This module can be used to manage an interactive online help repository. Help messages can be provided in a variety of formats (text, html, etc.) and at different levels of detail.

- **Overlap.pm**
  The *Overlap.pm* package has been implemented for computing overlapping regions and intergenic regions based on a generic input of position values.

- **Profiler.pm**
  This package provides some basic methods for the profiling of Perl programs. It can be used to compute time intervals required for the execution of specific parts in a running program.

- **QuerySRS.pm**
  The *QuerySRS.pm* module encapsulates accessess to an SRS server. It has been implemented for retrieving database entries via the SRS system and it can return the data in different formats.

- **StringUtils.pm**
  This module provides simple methods for handling strings, e.g. splitting a given string near to an approximately given position where a specified delimiter occurs.

- **UserConfig.pm**
  The package *UserConfig* has been implemented as an generic interface that can be used to access different configuration backends (e.g. .ini files or the project management system). Individual storage backends are thus implemented in special subclasses which are not listed here.

- **Wizard.pm**
  This module can be used as a basic framework for the implementation of special functional modules. A *Wizard* can provide special callbacks which are executed upon specific actions. As an example, the *JobSubmitterWizard* has been derived from this class.

APPENDIX B

Installation of the software

This chapter finally gives some additional information and hints for the installation of the BRIDGE system. The current version and updates of the software can be found on the web site of each project.

B.1. System requirements

Since one aim of all projects described above is to provide a platform for end users and developers, the system has very modest system requirements. A Unix system with Perl, an SQL database and BioPerl is necessary. For the full functionality some additional tools are needed that have to be installed on the system. If the user wants to schedule computations the Sun Grid Engine or another queuing system has to be installed as well. For a complete local installation of the GenDB system, the sequence databases used by the tools and some sequence retrieval mechanism are required. We currently use SRS and BioPerl for this purpose. Of the systems available today only SRS provides user friendly views on the sequence databases. The system can be installed on a single (e.g. Linux) server or can be spread out over multiple machines creating a client-server installation. Locally, test and development installations exist on single CPU Linux platforms like Suse Linux, while our production environment includes a client-server environment with a server for the frontend, a dedicated database server and a BioGrid to perform time consuming computations.

B.2. License

To provide a resource to the academic community the complete system (including source code) is distributed to non-commercial users under an open source license. Special commercial licenses are also available on request.

Bibliography

[AAB$^+$01] R. Apweiler, T. K. Attwood, A. Bairoch, A. Bateman, E. Birney, M. Biswas, P. Bucher, L. Cerutti, F. Corpet, M. D. R. Croning, R. Durbin, L. Falquet, W. Fleischmann, J. Gouzy, H. Hermjakob, N. Hulo, I. Jonassen, D. Kahn, A. Kanapin, Y. Karavidopoulou, R. Lopez, B. Marx, T. M. Mulder, N. J.and Oinn, M. Pagni, F. Servant, C. J. A. Sigrist, and E. M. Zdobnov. The InterPro database, an integrated documentation resource for protein families, domains and functional sites. *Nucleic Acids Res.*, 29(1):37–40, 2001.

[ABL$^+$99] M. A. Andrade, N. P. Brown, C. Leroy, S. Hoersch, A. de Daruvar, C. Reich, A. Franchini, J. Tamames, A. Valencia, C. Ouzounis, and C. Sander. Automated genome sequence analysis and annotation. *Bioinformatics*, 15(5):391–412, 1999.

[AMS$^+$97] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, 25:3389–3402, 1997.

[Bai00] A. Bairoch. The ENZYME database in 2000. *Nucleic Acids Res.*, 28(1):304–305, January 2000.

[BB90] Dodd I. B. and Egan J. B. Improved detection of helix-turn-helix DNA-binding motifs in protein sequences. *Nucleic Acids Res.*, 18(17):5019–5026, September 1990.

[BBC$^+$02] A. Bateman, E. Birney, L. Cerruti, R. Durbin, L. Etwiller, S. R. Eddy,

S. Griffiths-Jones, K. L. Howe, M. Marshall, and E. L. L. Sonnhammer. The Pfam protein families database. *Nucleic Acids Res.*, 30(1):276–280, 2002. The Pfam contribution to the annual NAR database issue.

[BCCT+02] S. D. Bentley, K. F. Chater, A.-M. Cerdeno-Tarraga, G. L. Challis, N. R. Thomson, K. D. James, D. E. Harris, M. A. Quail, H. Kieser, D. Harper, A. Bateman, S. Brown, G. Chandra, C. W. Chen, M. Collins, A. Cronin, A. Fraser, A. Goble, J. Hidalgo, T. Hornsby, S. Howarth, C.-H. Huang, T. Kieser, L. Larke, L. Murphy, K. Oliver, S. O'Neil, E. Rabbinowitsch, M.-A. Rajandream, K. Rutherford, S. Rutter, K. Seeger, D. Saunders, S. Sharp, R. Squares, S. Squares, K. Taylor, T. Warren, A. Wietzorrek, J. Woodward, B. G. Barrell, J. Parkhill, and D. A. Hopwood. Complete genome sequence of the model actinomycete *Streptomyces coelicolor* A3(2). *Nature*, 417(6885):141–147, 2002.

[BFJ+01] M. J. Barnett, R. F. Fisher, T. Jones, C. Komp, A. P. Abola, F. Barloy-Hubler, L. Bowser, D. Capela, F. Galibert, J. Gouzy, et al. Nucleotide sequence and predicted functions of the entire *Sinorhizobium meliloti* pSymA megaplasmid. *Proc. Natl. Acad. Sci. USA*, 98(17):9883–9888, 2001.

[BH02] P. Baldi and G. W. Hatfield. *DNA microarrays and gene expression: from experiment to data analysis and modeling*. Cambridge University Press, 2002.

[BHQ+01] A. Brazma, P. Hingamp, J. Quackenbush, G. Sherlock, P. Spellman, C. Stoeckert, J. Aach, W. Ansorge, C. A. Ball, H. C. Causton, T. Gaasterland, P. Glenisson, F. C. P. Holstege, I. F. Kim, V. Markowitz, J. C. Matese, H. Parkinson, A. Robinson, U. Sarkans, S. Schulze-Kremer, J. Stewart, R. Taylor, J. Vilo, and M. Vingron. Minimum information about a microarray experiment (MIAME) toward standards for microarray data. *Nature Genetics*, 29(4):365–371, 2001.

[BKML+02] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B. A. Rapp, and D. L. Wheeler. GenBank. *Nucleic Acids Res.*, 30(1):17–20, January 2002.

[BO99] H. Badger and G. J. Olsen. CRITICA: coding region identification tool invoking comparative analysis. *Mol. Biol. Evol.*, 16:512–524, 1999.

[Bow99] D. L. Bowtell. Options available from start to finish for obtaining expression data by microarray. *Nature genetics*, 21(1 Suppl):25–31, January 1999.

[BPB+97] F. R. Blattner, G. Plunkett, C. A. Bloch, N. T. Perna, V. Burland, M. Riley, J. Collado-Vides, J. D. Glasner, C. K. Rode, G. F. Mayhew, J. Gregor, N. W. Davis, H. A. Kirkpatrick, M. A. Goeden, D. J. Rose, B. Mau, and

Y. Shao. The complete genome sequence of Escherichia coli K-12. *Science*, 277(5331):1453–1462, 1997.

[BPSMM00] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML). *http://www.w3.org/TR/REC-xml*, 2000.

[BTM$^+$02] H. Bannai, Y. Tamada, O. Maruyama, K. Nakai, and S. Miyano. Extensive feature detection of N-terminal protein sorting signals. *Bioinformatics*, 18(2):298–305, 2002.

[Bun02] Bundesministerium für Bildung und Forschung (BMBF). Systeme des Lebens - Systembiologie. Referat Öffentlichkeitsarbeit, 53170 Bonn, September 2002.

[CAM$^+$99] V. G. Cheung, M. Aguilar, F. Massimi, A. Kucherlapati, and R. Childs. Making and reading microarrays. *Nature Genetics*, 21(1 Suppl.):15–19, January 1999.

[Cla02] Jörn Clausen. O2DBI. Technical report, Bielefeld University, 2002.

[DBC$^+$99] D. J. Duggan, M. Bittner, Y. Chen, P. Meltzer, and J. M. Trent. Expression profiling using cDNA microarrays. *Nature genetics*, 21(1 Suppl):10–14, January 1999.

[DGB$^+$03] M. Dondrup, A. Goesmann, D. Bartels, J. Kalinowski, L. Krause, B. Linke, O. Rupp, A. Sczyrba, A. Pühler, and F. Meyer. EMMA: A platform for consistent storage and efficient analysis of array-based data. *Journal of Biotechnology*, 106(2-3):135–146, December 2003.

[DHK$^+$99] A. L. Delcher, D. Harmon, S. Kasif, O. White, and S. L. Salzberg. Improved microbial gene identification with GLIMMER. *Nucleic Acids Res.*, 27:4636–4641, 1999.

[DIB97] J. L. DeRisi, V. R. Iyer, and P. O. Brown. Exploring gene expression on a genomic scale. *Science*, 278:680–686, 1997.

[DJ01] B. Dysvik and I. Jonassen. J-Express: exploring gene expression data using Java. *Bioinformatics*, 17:369–370, 2001.

[DJD$^+$01] R. D. Dowell, R. M. Jokerst, A. Day, S. R. Eddy, and L. Stein. The Distributed Annotation System. *BMC Bioinformatics*, 2(7), 2001.

[DJSH$^+$03] G. Dennis Jr., B. T. Sherman, D. A. Hosack, J. Y. W. Gao, H. C. Lane, and R. A. Lempicki. DAVID: Database for Annotation, Visualization, and Integrated Discovery. *Genome Biology*, 4, 2003.

[DYCS00] S. Dudoit, Y. H. Yang, M. J. Callow, and T. P. Speed. Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. Technical report, Department of Biochemistry, Stanford University, August 2000.

[DYCS02] S. Dudoit, Y. H. Yang, M. J. Callow, and T. P. Speed. Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. *Statistica Sinica*, 12(1):111–139, 2002.

[EA93] T. Etzold and P. Argos. SRS an indexing and retrieval tool for flat file data libraries. *Cabios*, 9:49–57, 1993.

[Eck95] W. W. Eckerson. Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. *Open Information Systems*, 10(1), 1995.

[Edd98] S. R. Eddy. Profile hidden Markov models. *Bioinformatics*, 14:755–763, 1998.

[EHKW03] L. B. M. Ellis, B. K. Hou, W. Kang, and L. P. Wackett. The university of minnesota biocatalysis/biodegradation database: Post-genomic datamining. *Nucleic Acids Res.*, 31(1):262–265, 2003.

[ESBB98] M. Eisen, P. Spellman, D. Botstein, and P. Brown. Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. USA*, 95:14863–14868, December 1998.

[FAH$^+$01] D. Frishman, K. Albermann, J. Hani, K. Heumann, A. Metanomski, A. Zollner, and H. W. Mewes. Functional and structural genomics using PEDANT. *Bioinformatics*, 17(1):44–57, 2001.

[FEN$^+$02] C. M. Fraser, J. A. Eisen, K. E. Nelson, I. T. Paulsen, and S. L. Salzberg. The value of complete microbial genome sequencing (You get what you pay for). *J. Bacteriol.*, 184:6403–6405, 2002.

[FES00] C. M. Fraser, J. A. Eisen, and S. L. Salzberg. Microbial genome sequencing. *Nature*, 406:799–803, 2000.

[FF97] C. M. Fraser and R. D. Fleischmann. Strategies for whole microbial genome sequencing and analysis. *Electrophoresis*, 18:1207–1216, 1997.

[FJT$^+$03] M. E. Frazier, M. J. Johnson, D. G. Thomassen, C. E. Oliver, and A. Patrinos. Realizing the Potential of the Genome Revolution: The Genomes to Life Program. *Science*, 300(290):290–293, April 2003.

[FMMG98] D. Frishman, A. Mironov, H. Mewes, and M. Gelfand. Combining diverse evidence for gene recognition in completely sequenced bacterial genomes. *Nucleic Acids Res.*, 26(12):2941–2947, 1998.

[FPB$^+$02] L. Falquet, M. Pagni, P. Bucher, N. Hulo, C. J Sigrist, K. Hofmann, and A. Bairoch. The PROSITE database, its status in 2002. *Nucleic Acids Res.*, 30:235–238, 2002.

[FPR$^+$02] M. Forster, A. Pick, M. Raitner, F. Schreiber, and F. J. Brandenburg. The System Architecture of BioPath. *In Silico Biology*, 2(0037), 2002.

[FRH$^+$93] S. P. Fodor, R. P. Rava, X. C. Huang, A. C. Pease, C. P. Holmes, and C. L. Adams. Multiplexed biochemical assays with biological chips. *Nature*, 364:555–556, 1993.

[FWW$^+$01] T. M. Finan, S. Weidner, K. Wong, J. Buhrmester, P. Chain, F. J. Vorhölter, I. Hernandez-Lucas, A. Becker, A. Cowie, J. Gouzy, B. Golding, and A. Pühler. The complete sequence of the 1,683-kb psymb megaplasmid from the n2-fixing endosymbiont *Sinorhizobium meliloti*. *Proc. Natl. Acad. Sci. USA*, 98:9889–9894, 2001.

[GFL$^+$01] F. Galibert, T. M. Finan, S. R. Long, A. Pühler, P. Abola, F. Ampe, F. Barloy-Hubler, M. J. Barnett, A. Becker, P. Boistard, G. Bothe, M. Boutry, L. Bowser, J. Buhrmester, E. Cadieu, D. Capela, P. Chain, A. Cowie, R. W. Davis, S. Dreano, N. A. Federspiel, R. F. Fisher, S. Gloux, T. Godrie, A. Goffeau, B. Golding, J. Gouzy, M. Gurjal, I. Hernandez-Lucas, A. Hong, L. Huizar, R. W. Hyman, T. Jones, D. Kahn, M. L. Kahn, S. Kalman, D. H. Keating, E. Kiss, C. Komp, V. Lelaure, D. Masuy, C. Palm, M. C. Peck, T. M. Pohl, D. Portetelle, B. Purnelle, U. Ramsperger, R. Surzycki, P. Thebault, M. Vandenbol, F. J. Vorhölter, S. Weidner, D. H. Wells, K. Wong, K. C. Yeh, and J. Batut. The composite genome of the legume symbiont *Sinorhizobium meliloti*. *Science*, 29:668–72, 2001.

[GH02] P. R. Graves and T. A. J. Haystead. Molecular Biologist's Guide to Proteomics. *Microbiology and Molecular Biology Reviews*, 66(1):39–63, 2002.

[GHM$^+$02] A. Goesmann, M. Haubrock, F. Meyer, J. Kalinowski, and R. Giegerich. PathFinder: reconstruction and dynamic visualization of metabolic pathways. *Bioinformatics*, 18(1):124–129, 2002.

[GHW96] U. E. Gibson, C. A. Heid, and P. M. Williams. A novel method for real time quantitative RT-PCR. *Genome Research*, 6(10):995–1001, 1996.

[GLR$^+$03] A. Goesmann, B. Linke, O. Rupp, L. Krause, D. Bartels, M. Dondrup, A. C. McHardy, A. Wilke, A. Pühler, and F. Meyer. Building a BRIDGE for the integration of heterogeneous data from functional genomics into a platform for systems biology. *Journal of Biotechnology,*, 106(2-3):157–167, December 2003.

[GOZ03] F.-B. Guo, H.-Y. Ou, and C.-T. Zhang. ZCURVE: a new system for recognizing protein-coding genes in bacterial and archaeal genomes. *Nucleic Acids Res.*, 31:1780–1789, 2003.

[Gre01] E. D. Green. Strategies for the systematic sequencing of complex genomes. *Nat. Rev. Genet.*, 2(8):573–583, August 2001.

[GS96] T. Gaasterland and C. W. Sensen. MAGPIE: automated genome interpretation. *Trends Genet*, 12(2):76–8, 1996.

[HBB$^+$03] A. T. Hüser, A. Becker, I. Brune, M. Dondrup, J. Kalinowski, J. Plassmeier, A. Pühler, I. Wiegräbe, and A. Tauch. Development of a *Corynebacterium glutamicum* DNA microarray and validation by genome-wide expression profiling during growth with propionate as carbon source. *Journal of Biotechnology*, submitted, 2003.

[HGPH00] J. G. Henikoff, E. A. Greene, S. Pietrokovski, and S. Henikoff. Increased coverage of protein families with the blocks database servers. *Nucleic Acids Res.*, 28:228–230, 2000.

[HHP99] S. Henikoff, J. G. Henikoff, and S. Pietrokovski. Blocks+: A non-redundant database of protein alignment blocks dervied from multiple compilations. *Bioinformatics*, 15(6):471–479, 1999.

[HSLW96] C. A. Heid, J. Stevens, K. J. Livak, and P. M. Williams. Real time quantitative PCR. *Genome Research*, 6(10):986–994, 1996.

[HvHS$^+$02] W. Huber, A. von Heydebreck, H. Sültmann, A. Poustka, and M. Vingron. Variance stabilization applied to microarray data calibration and to the quantification of differential expression. *Bioinformatics*, 18(Suppl 1):96–104, 2002.

[IG96] R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.

[IGH01] T. Ideker, T. Galitski, and L. Hood. A new approach to decoding life: Systems Biology. *Annual Review of Genomics and Human Genetics*, 2:343–372, 2001.

[Jol86] I.T. Joliffe. *Principal component analysis*. Springer, 1986.

[Kar98] P. D. Karp. Metabolic databases. *Trends In Biochemical Sciences*, 23(3):114–116, March 1998.

[Kat03] F. Katagiri. Attacking Complex Problems with the Power of Systems Biology. *Plant Physiol.*, 132(2):417–419, June 2003.

[KB03] L. Krol and A. Becker. Phosphate regulon of *Sinorhizobium meliloti*. personal communication, 2003.

[KBB$^+$03a] O. Kaiser, D. Bartels, T. Bekel, A. Goesmann, S. Kespohl, A. Pühler, and F. Meyer. Whole genome shotgun sequencing guided by bioinformatics pipelines – an optimized approach for an established technique. *Journal of Biotechnology*, 106(2-3):121–133, December 2003.

[KBB$^+$03b] J. Kalinowski, B. Bathe, D. Bartels, N. Bischoff, M. Bott, A. Burkovski, N. Dusch, L. Eggeling, B. J. Eikmanns, L. Gaigalat, A. Goesmann, M. Hartmann, K. Huthmacher, R. Krämer, B. Linke, A. C. McHardy, F. Meyer, B. Möckel, W. Pfefferle, A. Pühler, D. A. Rey, C. Rückert, O. Rupp, R. Sahm, V. F. Wendisch, I. Wiegräbe, and A. Tauch. The complete *Corynebacterium glutamicum* ATCC 13032 genome sequence and its impact on the production of l-aspartate-derived amino acids and vitamins. *Journal of Biotechnology*, 104(1-3):5–25, 2003. in press.

[KG00] M. Kanehisa and S. Goto. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Res.*, 28(1):27–30, January 2000.

[Kin81] S. Kindel. title unknown. *Technology*, 1:62, 1981.

[KKB03] I. S. Kohane, A. T. Kho, and A. J. Butte. *Microarrays for an Integrative Genomics*. The MIT Press, 2003.

[KKSed] M. Katzer, F. Kummert, and G. Sagerer. Robust automatic microarray image analysis. In *Proceedings of the International Conference on Bioinformatics*, Bangkok, 2002 (accepted).

[Koh97] Teuvo Kohonen. *Self-organizing maps*. Springer, 1997.

[KR90] R. Kaufman and P. J. Rousseeuw. *Finding Groups in Data. An Introduction to Cluster Analysis*. Wiley Series in Probability and Mathematical Statistics. Wiley, 1990.

[KR97] J. Koolman and K.-H. Röhm. *Taschenaltlas der Biochemie*. Georg Thieme Verlag, Stuttgart; New York, 2. edition, 1997.

[KRS$^+$00] P. D. Karp, M. Riley, M. Saier, I. T. Paulsen, S. M. Paley, and A. Pellegrini-Toole. The EcoCyc and MetaCyc databases. *Nucleic Acids Res.*, 28(1):56–59, January 2000.

[KSK02] J. Köhler and S. Schulze-Kremer. The Semantic Metadatabase (SEMEDA): Ontology based integration of federated molecular biological data sources. *In Silico Biology*, 2(21), 2002.

[Küs03] H. Küster. Using DNA arrays for expression profiling and identification of candidate genes. *Grain Legumes*, 38(23), 2003.

[LE97] T. M. Lowe and S. R. Eddy. tRNAscan-SE: a program for improved detection of transfer RNA genes in genomic sequence. *Nucleic Acids Res.*, 25(5):955–964, March 1997.

[Leh85] A. L. Lehninger. *Grundkurs Biochemie*. Walter de Gruyter, 2. edition, 1985. ISBN 3-11-0102210-8.

[Lin02] B. Linke. O2DBI II – ein Persistenzlayer für Perl-Objekte. Master's thesis, Bielefeld University, 2002.

[Mac67] J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297. University of California Press, 1967.

[Mey01] F. Meyer. *GenDB – A second generation genome annotation system*. PhD thesis, Bielefeld University, 2001.

[MGEa] MGED. Minimum information about a microarray experiment – MIAME. *http://www.mged.org/Annotations-wg/MGEDAnnotNov2000/ mgedannotnov2000.html.*

[MGEb] MGED. MicroArray and Gene Expression – MAGE. *http://www.mged.org /Workgroups/MAGE/mage.html.*

[MGM$^+$03] F. Meyer, A. Goesmann, A. McHardy, D. Bartels, T. Bekel, J. Clausen, J. Kalinowski, B. Linke, O. Rupp, R. Giegerich, and A. Pühler. GenDB – an open source genome annotation system for prokaryote genomes. *Nucleic Acids Res.*, 2003.

[Mic92] G. Michal, editor. *Biochemical Pathways*. Böhringer Mannheim, Germany, 3. edition, 1992.

[Mic99] G. Michal. *Biochemical pathways (Biochemie-Atlas)*. Spektrum Akademischer Verlag GmbH, Heidelberg; Berlin, 1999. ISBN 3-86025-239-9.

[MKM] P. Mattis, S. Kimball, and J. MacDonald. The Gimp Toolkit. *http://www.gtk.org/*.

[MKPM04] A. C. McHardy, J. Kalinowski, A. P"uhler, and F. Meyer. Comparing expression level-dependent features in codon usage with protein abundance: An analysis of predictive proteomics. *Proteomics*, 4(1):46–58, 2004.

[Mur85] F. Murtagh. Multidimensional clustering algorithms. *COMPSTAT Lectures*, 4, 1985.

[NEBH97] H. Nielsen, J. Engelbrecht, S. Brunak, and G. Heijne. Identification of prokaryotic and eukaryotic signal peptides and prediction of their cleavage sites. *Protein Engineering*, 10:1–6, 1997.

[Nis97] T. Nishizuka, editor. *Cell Funcions and Metabolic Maps*. Biochemical Society of Japan, 1997.

[OB02] Z. N. Oltvai and A.-L. Barabási. Life’s Complexity Pyramid. *Science*, 298, October 2002.

[OLP$^+$02] R. Overbeek, N. Larsen, G. D. Pusch, M. D’Souza, E. Selkov, N. Kyrpides, M. Fonstein, N. Maltsev, and E. Selkov. WIT: integrated system for high-throughput genome sequence analysis and metabolic reconstruction. *Nucleic Acids Res.*, 28:123–125, 2002.

[OLW$^+$03] R. Overbeek, N. Larsen, T. Walunas, M. D’Souza, G. Pusch, E. Selkov Jr., K. Liolios, V. Joukov, D. Kaznadzey, I. Anderson, A. Bhattacharyya, H. Burd, W. Gardner, P. Hanke, V. Kapatral, N. Mikhailova, O. Vasieva, A. Osterman, V. Vonstein, M. Fonstein, N. Ivanova, and N. Kyrpides. The ERGO genome analysis and discovery system. *Nucleic Acids Res.*, 31(1):164–171, January 2003.

[Pea90] W. R. Pearson. Rapid and Sensitive Sequence Comparison with FASTP and FASTA. *Methods in Enzymology*, 183:63–98, 1990.

[Per] Perl – Practical Extraction and Report Language. *http://www.perl.org/*.

[PL88] W. R. Pearson and D. J. Lipman. Improved Tools for Biological Sequence Comparison. *PNAS*, 85:2444–2448, 1988.

[Ril93] M. Riley. Functions of gene products of *Escherichia coli*. *Microbiol. Rev.*, 57(4):862–952, 1993.

[Ril98] M. Riley. Genes and proteins of *Escherichia coli* k-12 (genprotec). *nar*, 26:54–55, 1998.

[RJR$^+$04] S. Rendulic, P. Jagtap, A. Rosinus, M. Eppinger, C. Baar, C. Lanz, H. Keller, C. Lambert, K.J. Evans, A. Goesmann, F. Meyer, R.E. Sockett, and S.C. Schuster. A Predator Unmasked: The Lifecycle of *Bdellovibrio bacteriovorus* from a Genomic Perspective. *Science*, 303(5658):689–692, 2004.

[RKO$^+$03] A. Rowe, D. Kalaitzopoulos, M. Osmond, M. Ghanem, and Y. Guo. The discovery net system for high throughput bioinformatics. *Bioinformatics*, 19(1):225–231, 2003.

[RPC$^+$00] K. M. Rutherford, J. Parkhill, J. Crook, T. Horsnell, P. Rice, M.-A. Rajandream, and B. Barrell. Artemis: sequence visualisation and annotation. *Bioinformatics*, 16(10):994–945, 2000.

[RPK03] C. Rückert, A. Pühler, and J. Kalinowski. Genome-wide analysis of the L-methionine biosynthetic pathway in *Corynebacterium glutamicum*. *Journal of Biotechnology,*, 104(1-3):213–228, 2003.

[RTK$^+$03] S. Rüberg, Z. X. Tian, E. Krol, B. Linke, F. Meyer, Y. Wang, A. Pühler, and A. Becker. Construction and validation of a *Sinorhizobium meliloti* whole genome DNA microarray: genome-wide profiling of osmoadaptive gene expression. *Journal of Biotechnology*, 2003. submitted.

[San95] G. Sander. Vcg – visualization of compiler graphs. Technical report, Universität des Saarlandes, FB 14 Informatik, 1995.

[Sch95] G. Schussel. Client/server past, present, and future. online, 1995.

[SFT$^+$01] A. Siepel, A. Farmer, A. Tolopko, M. Zhuang, P. Mendes, W. Beavis, and B. Sobral. ISYS: a decentralized, component-based approach to the integration of heterogeneous bioinformatics resources. *Bioinformatics*, 17(1):83–94, 2001.

[SGMS98] E. Selkov, Y. Grechkin, N. Mikhailova, and E. Selkov. MPW: the Metabolic Pathways Database. *Nucleic Acids Res.*, 26(1):43–45, January 1998.

[SGS$^+$88] R. K. Saiki, D. H. Gelfand, S. Stoffel, S. J. Scharf, R. Higuchi, G. T. Horn, K. B. Mullis, and H. A. Erlich. Primer-directed enzymatic amplification of DNA with a thermostable DNA polymerase. *Science*, 239(4839):487–491, 1988.

[SNC77] F. Sanger, S. Nicklen, and A. R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proc. Natl. Acad. Sci. USA*, 74:5463–5467, 1977.

[SRG03] R. D. Stevens, A. J. Robinson, and C. A. Goble. $^{\text{my}}$Grid: personalised bioinformatics on the information grid. *Bioinformatics*, 19(1):302–304, 2003.

[SSDB95] M. Schena, D. Shalon, R. Davis, and P. Brown. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, 270:467–470, 1995.

[SSK+86] L. M. Smith, J. Z. Sanders, R. J. Kaiser, P. Hughes, C. Dodd, C. R. Connell, C. Heiner, S. B. Kent, and L. E. Hood. Fluorescence detection in automated DNA sequence analysis. *Nature*, 321(6071):674–679, June 1986.

[SSS95] D. Schomburg, D. Salzmann, and D. Stephan. *Enzyme Handbook, Classes 1-6*. Springer, 1990-1995.

[SSZ+98] P. T. Spellman, G. Sherlock, M. Q. Zhang, V. R. Iyer, K. Anders, M. B. Eisen, P. O. Brown, D. Botstein, and B. Futcher. Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. *Molecular Biology of the Cell*, 9:3273–3297, 1998.

[Str91] L. Stryer. *Biochemie*. Spektrum Akademischer Verlag GmbH, Heidelberg; Berlin; New York, 1991. ISBN 3-86025-005-1.

[STVC+02] L. H. Saal, C. Troein, J. Vallon-Christersson, S. Gruvberger, A. Borg, and C. Peterson. BioArray Software Environment (BASE): a platform for comprehensive management and analysis of microarray data. *Genome Biology*, 3(8), 2002.

[SvHK98] E. L. L. Sonnhammer, G. von Heijne, and A. Krogh. A hidden markov model for predicting transmembrane helices in protein sequences. In J. Glasgow, T. Littlejohn, R. Major, F. Lathrop, D. Sankoff, and C. Sensen, editors, *Proceedings of the Sixth International Conference on Intelligent Systems for Molecular Biology*, pages 175–182, Menlo Park, CA, 1998. AAAI Press.

[The00] The Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.

[THM+02] A. Tauch, I. Homann, S. Mormann, S. Rüberg, A. Billault, B. Bathe, S. Brand, O. Brockmann-Gretza, C. Rückert, N. Schischka, C. Wrenger, J. Hoheisel, B. Möckel, K. Huthmacher, W. Pfefferle, A. Pühler, and J. Kalinowski. Strategy to sequence the genome of *Corynebacterium glutamicum* ATCC 13032: use of a cosmid and a bacterial artificial chromosome library. *Journal of Biotechnology*, 95(1):25–38, 2002.

[TNG+01] R. L. Tatusov, D. A. Natale, I. V. Garkavtsev, T. A. Tatusova, U. T. Shankavaram, B. S. Rao, B. Kiryutin, M. Y. Galperin, N. D. Fedorova, and E. V. Koonin. The COG database: new developments in phylogenetic classification of proteins from complete genomes. *Nucleic Acids Res.*, 29(1):22–8, January 2001.

[WC53] J. D. Watson and F. H. C. Crick. A structure for deoxyribose nucleic acid. *Nature*, pages 171–173, 1953.

[WL02] M. D. Wilkinson and M. Links. BioMOBY: An open source biological web services proposal. *Briefings in Bioinformatics*, 3(4):331–341, December 2002.

[Woe87] Carl Woese. Bacterial evolution. *Microbiological Rev.*, 1987.

[WPH$^+$04] J. Westberg, A. Persson, A. Holmberg, A. Goesmann, J. Lundeberg, K.-E. Johansson, B. Pettersson, and M. Uhlén. The Genome Sequence of Mycoplasma mycoides subsp. mycoides SC Type Strain PG1T, the Causative Agent of Contagious Bovine Pleuropneumonia (CBPP). *Genome Res.*, 14:221–227, 2004.

[WPK03] S. Weidner, A. Pühler, and H. Küster. Genomics insights into symbiotic nitrogen fixation. *Current Opinion in Biotechnology*, 14(2):200–205, 2003.

[WRB$^+$03] A. Wilke, C. Rückert, D. Bartels, M. Dondrup, A. Goesmann, A. T. Hüser, S. Kespohl, B. Linke, M. Mahne, A. McHardy, A. Pühler, and F. Meyer. ProDB: Bioinformatics support for high throughput proteomics. *Journal of Biotechnology*, 106(2-3):147–156, December 2003.

[WY93] P. H. Westfall and S. S. Young. *Resampling-Based multiple testing: examples and methods for p-value*. Wiley Series in Probability and Mathematical Statistics. Wiley, 1993.

[YDLSa] Y. H. Yang, S. Dudoit, P. Luu, and T. P. Speed. Normalization for cDNA Microarray Data.

[YDLSb] Y. H. Yang, S. Dudoit, P. Luu, and T. P. Speed. Normalization for cDNA Microarray Data. Technical report, Department of Statistics, University of California at Berkeley.

[YHR01] K. Y. Yeung, D. R. Haynor, and W. L. Ruzzo. Validating clustering for gene expression data. In *Proceedings of the 9th International Conference on Intelligent Systems For Molecular Biology (ISMB 2001)*, volume 17, pages 309–318. Oxford University Press, 2001.

[Zha99] M. Q. Zhang. Large-scale gene expression data analysis: a new challenge to computational biologists. *Genome Research*, 9:681–688, 1999.