ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze

Semesterarbeit

Datenpool für die Verwaltung von Genom Daten

Andreas Meier

Assistenz: Amela Prelić, Stefan Bleuler
Professor: Dr. Eckart Zitzler

Abgabe: 23. Dezember 2004

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

1.1 Thematischer Hintergrund

Verschiedene neue Methoden und Verfahren haben die molekulare Zellbiologie in den letzten Jahren grundlegend verändert. So ermöglichen so genannte high throughput Methoden die parallele Messung von einigen Tausend Variablen, wie mRNA Konzentrationen oder Protein-Protein-Interaktionen. Diese Messungen mussten vor wenigen Jahren noch einzeln vorgenommen werden. Eines der meist verwendeten Verfahren in diesem Gebiet ist dabei die Mikroarray Technologie. Bei den DNA-Mikroarrays können DNA-Sequenzen mit Hilfe eines Roboters auf eine kleine Glasoberfläche strukturiert aufgetragen werden. Mit einem solchen Mikroarray ist es möglich verschiedene Experimente in kurzer Zeit durchzuführen. Beispielsweise kann man Pflanzen unter speziellen und unter normalen Bedingungen wachsen lassen. Aus bestimmten Teilen der Pflanze werden zu verschiedenen Zeitpunkten mRNA extrahiert, die dann luminesziert und auf dem Mikroarray hybridisiert wird. Anschliessend kann die mRNA-Konzentration gemessen werden.

Andererseits gibt es Forschung, welche sich auf einzelne Gen-Produkte konzentriert und damit sehr spezifische Informationen über einzelne Enzyme, Proteine oder auch Gene produziert. Aktuelle Forschung in der Molekularbiologie befasst sich mit einer Vielzahl von verschiedenen Ansätzen um Gene charakterisieren zu können. So werden Gen-Ontologie Annotation bestimmt, es wird ermittelt welche Proteine miteinander interagieren oder welche Einflüsse Proteine auf den Metabolismus haben.

1.2 Bestehende Projekte

Es existieren bereits verschiedene Projekte und Datenbanken, die sich mit der Sammlung von Genomdaten befassen. Dies hat dazu geführt das viel Datenmaterial redundant und als Folge davon nicht konsistent und zum Teil sogar fehlerhaft vorhanden ist. An dieser Stelle werden die für dieses Projekt wichtigsten Datensammlungen vorgestellt:

- Die *Universal Protein Resource – UniProt* [1] ist die weltweit umfassendste Quelle für Informationen über Proteine. Sie geht aus einem Zusammenschluss der drei Datensammlungen Swiss-Prot [7], TrEMBL [8] und PIR [9] hervor. Die Datensammlung ist sehr umfassend, jedoch ist sie darauf ausgelegt Daten von einzelnen Proteinen zur Verfügung zu stellen. Es gibt aber zum Beispiel keine einfache Möglichkeit alle Synonyme der Proteine aller Gene eines Organismus abzufragen.

- Für die Genexpressionsdaten gibt es den *Gene Expression Omnibus – GEO* [2]. Dieser stellt eine Online Plattform dar, die erlaubt Genexpressionsdaten abzufragen. Diese Plattform ist eine sehr umfangreiche

Datensammlung, die aber nur beschränkt Datentypübergreifende Abfragen erlaubt.

- Das *Gene Ontology Consortium* [3] wird in Abschnitt 2.2 näher beschrieben.
- Das *European Bioinformatics Institute – EBI* [10] verwaltet Datenbanken für biologische Daten wie den *ArrayExpress* [11] für Genexpressionsdaten oder die *Nucleotide Database – EMBL* [12] für Nukleinsäuren.

1.3 Motivation und Zielsetzung

Die Frage ist nun berechtigt, warum wir noch ein weiterer Datenpool erstellen wollen. Im Gegensatz zu den existierenden Datenbanken, soll aber keineswegs eine öffentliche Datenquelle erstellt werden, sondern nur ein intern benutzter Datenpool. Dies hat einen direkten Einfluss an die Anforderungen. Einerseits spielt die Performance nur eine untergeordnete Rolle, da beispielsweise keine Webanfragen an den Datenpool gestellt werden, die möglichst schnell beantwortet sein müssen. Andererseits wollen wir auch kein Daten-Repository aufbauen. Wir wollen nur aktuell benötigte Daten verwalten, welche nach der Verwendung aus Performance Gründen wieder gelöscht werden sollten.

Der Datenpool wird ausschliesslich von einer Arbeitsgruppe hier an der ETH Zürich benutzt, welche sich aus Leuten mit unterschiedlichem Background wie Biologen, Statistiker aber auch Ingenieuren zusammensetzt. Diese Gruppe ist darauf angewiesen Daten verschiedenen Ursprungs lokal zu sammeln, um diese dann mit bestimmten Kriterien auch Datentypübergreifend zu filtern. Beispielsweise sollen anhand einer Gen-Liste alle Protein-Protein-Interaktionen ausgewählt werden, dessen Proteine mit der Gen-Liste korrelieren.

Folgende Anforderungen werden an den Datenpool gestellt:

- Fünf verschiedene Datentypen werden in einer ersten Phase unterstützt: Genexpressionsdaten, Protein-Protein-Interaktionen, Produkte aus metabolischen Pfaden, Synthetic Lethal und Gen Ontologie Annotationen.
- Abfragen zwischen diesen Datentypen sind möglich. Wähle beispielsweise alle Genexpressionsdaten aus, dessen Gene (beziehungsweise deren Genprodukte) Protein-Protein-Interaktionen eingehen.
- Der Datenpool soll einfach strukturiert sein, und soll durch weitere Datentypen erweiterbar sein.
- Der Datenpool wird nur lokal benutzt.
- Einfache Benutzerschnittstelle.

1.4 Gliederung der Arbeit

Abschnitt 2 beschreibt die unterstützten Datentypen. Dabei wird der biologische Hintergrund erläutert und es wird auf die Probleme hingewiesen, die bei der Verwaltung dieser Daten auftreten können. Zum Schluss dieses Abschnittes folgen einige allgemeine Anmerkungen, die alle Datentypen betreffen.

Abschnitt 3 befasst sich mit den Anforderungen an den Datenpool. Neben allgemeinen Bemerkungen zur Verwaltung der Daten, werden die Abfragemöglichkeiten der verschiedenen Datentypen aufgelistet.

Abschnitt 4 beschäftigt sich mit der Architektur des Datenpools. Es wird erläutert auf welche Art die Daten gespeichert werden sollen und warum für die Gen Ontologie Daten speziell behandelt werden. Abschliessend wird der Aufbau der Applikation beschrieben.

Abschnitt 5 beschreibt wie die in Abschnitt 4 beschriebene Architektur umgesetzt wird. Zuerst wird die Struktur der Datenbank erläutert und anschliessend werden die wichtigsten Aspekte zur Applikation besprochen.

Abschnitt A befasst sich mit der Benutzung der Applikation. Dieser Abschnitt enthält eine Anforderungsliste an die Applikation. Es folgt eine Installations- und eine Bedienungs-Anleitung.

Abschnitt B erklärt anhand einiger SQL-Beispiele den Umgang mit SQL.

Abschnitt C enthält eine Beschreibung der Dateiformate.

Abschnitt D enthält die Aufgabenstellung dieser Arbeit.

2 Datentypen

Im Datenpool sollen verschiedene Arten von Daten gespeichert werden, die allesamt mit Genen und deren Produkten, hauptsächlich mit Proteinen, in Verbindung stehen. Nachfolgend werden die fünf verwendeten Datentypen kurz vorgestellt. Dazu gehören die einfachen Datentypen Protein-Protein-Interaktionen, metabolische Pfade sowie Synthetic Lethal und die etwas komplexeren Datentypen der Genexpressionsdaten und der Annotationen. Es wird einerseits der biologische Hintergrund kurz erläutert und andererseits wird auf die Probleme hingewiesen, die bei der Verwaltung dieser Daten auftreten können. In einem letzten Abschnitt werden einige allgemeine Anmerkungen, die alle Datentypen betreffen, erläutert.

2.1 Genexpressionsdaten – GE-Daten

Unter Genexpression versteht man die Übersetzung der Information, die in der DNA kodiert ist in das entsprechende Genprodukt, meist ein Protein. In einem ersten Schritt, der Transkription, wird dabei der DNA-Strang in RNA übersetzt.[1] In einem zweiten Schritt, der Translation, wird mit der Hilfe der RNA das Protein synthetisiert. Weiterführende Information ist auf dem 'Gene Expression Omnibus' [2] zu finden.

In der Molekularbiologie ist es von grosser Wichtigkeit zu wissen, wie gross die Konzentration der einzelnen Proteine in einer Zelle ist. Bis vor wenigen Jahren war diese Bestimmung eine sehr aufwändige und sehr teure Angelegenheit. Dies hat sich schlagartig geändert, als die Mikroarray-Technologie entwickelt wurde. Mit Hilfe eines Mikroarrays, auch Chips genannt, kann die mRNA-Konzentration bestimmt werden, welcher in erster Näherung der Protein-Konzentration der Zelle entspricht. Es ist nun auch möglich die Zellen unter verschiedenen Einflussfaktoren zu betrachten. Der Organismus wird beispielsweise einem Stress ausgesetzt und es wird analysiert wie sich die Konzentration der mRNA im Verlauf der Zeit ändert, oder es wird verglichen, wie sich eine mutierte Zelle im Vergleich zu einem Wildtypen verhält.

Der Datentyp der Genexpressionsdaten (nachfolgend mit GE-Daten bezeichnet) sieht folgendermassen aus. Es gibt eine Liste von Genen, denen jeweils zu jedem Chip ein Wert zugewiesen ist. Jeder möglichen Gen-Chip Kombination wird also ein einzelner Wert zugewiesen. Vereinzelt kann es vorkommen, dass einer dieser Werte nicht bekannt ist. In diesem Fall spricht man von einem 'missing value'.

[1] Bei Eukaryoten kann es vorkommen das aus der ein und derselben DNA-Sequenz verschiedene RNA-Sequenzen entstehen. Dieses Phänomen wird als alternatives Splicing bezeichnet.

2.2 Gen Ontologie Annotationen – GO

Das Gene Ontology Consortium [3] ist eine gemeinschaftliche Anstrengung mit dem Ziel ein einheitliches Vokabular für Genprodukte durchzusetzen. Da es sich nicht um einen Standard handelt, ist die Beteiligung an diesem Projekt freiwillig. Jedoch richten sich die meisten Forschergruppen aus Eigeninteresse danach, um die eigenen Resultate mit anderen vergleichen zu können.

Die Beteiligten dieses Projektes entwickeln drei strukturierte Ontologien, um Genprodukte in Form ihrer molekularen Funktionen, den biologischen Prozessen an denen sie beteiligt sind und den zellulären Komponenten denen sie zugehören, zu beschreiben. Die drei Ontologien sind in direkten azyklischen Graphen strukturiert. Es gibt genau drei dieser Graphen. Jeder ist hierarchisch gegliedert, d.h. ein Kind ist immer spezialisierter als der Elternterm. Es ist aber möglich, im Gegensatz zu einer Baumstruktur, dass ein Kind mehrere direkte Eltern besitzt. Die Proteine sind an den Knoten dieser Graphen angehängt.

Das GO Konsortium hat zwei verschiedene Aufgaben. Einerseits werden die Ontologien gepflegt und anhand aktueller Resultate in der Forschung ergänzt. Andererseits werden die Assoziationen der Genprodukte zu den Ontologien verwaltet. Zusätzlich stellt das GO-Projekt Hilfsprogramme zur Verfügung, welche Verwaltung und Benutzung der Ontologien vereinfacht.

Der Datentyp der Gen Ontologie Annotationen (kurz GO) sieht folgendermassen aus: es gibt drei direkte azyklische Graphen (zelluläre Komponenten, biologische Prozesse und molekulare Funktionen), deren Knoten Gene zugeordnet sind.

2.3 Protein-Protein-Interaktionen – PPI

Bei sehr vielen Prozessen innerhalb einer Zelle spielen Protein Interaktionen eine wichtige Rolle. Die Proteine gehen dabei stabile langfristige oder auch nur kurzzeitige Komplexe ein, um zusammen eine Aufgabe zu erfüllen. Es wird geschätzt, dass jedes Protein Interaktionen mit durchschnittlich drei weiteren Proteinen eingeht. Beim Menschen mit seinen über 30'000 Proteinen sind das gegen 100'000 Interaktionen. Experimentell wurden davon aber erst einige Tausend identifiziert [4].

Beim Datentyp der Protein-Protein-Interaktionen (nachfolgend mit PPI bezeichnet) geht es darum Paare von Proteinen zu finden, welche sich zusammenschliessen um eine Funktion zu erledigen. Der Datentyp wird in Form einer Liste von Gen Paaren dargestellt, der jeweils ein Experimenttyp zugewiesen wird. Dieser besteht aus einer Zeichenkette mit einer Länge von maximal drei Bytes. Die Zeichenkette gibt Auskunft darüber, mit welcher Messmethode[2] die Interaktion festgestellt wurde.

[2]Eine solche Messmethode ist zum Beispiel das Yeast2Hybrid Verfahren.

2.4 Metabolische Pfade – MP

Die metabolischen Pfade oder auch Stoffwechselwege stellen die Abläufe des Zellstoffwechsels dar, welche insbesondere durch enzymgesteuerte Reaktionen gekennzeichnet sind. Diese biochemischen Reaktionen des Stoffwechsels werden oft durch Enzyme katalysiert. Die Katalysatoren sind meist Proteine, welche die notwendige Aktivierungsenergie herabsetzen können und so die Reaktionen beschleunigen.[5]

Der Datentyp der Metabolischen Pfade (kurz MP) ist schlicht und einfach eine Liste von Enzym Paaren. Ein solches Paar sagt aus, dass die beiden Enzyme, meist Proteine, an einem Prozess beteiligt sind.

2.5 Synthetic Lethal – SL

Zwei Mutationen sind 'synthetically lethal' wenn die Zelle mit einer Mutation lebensfähig ist, mit beiden aber zugrunde geht. Dies deutet häufig an, dass die beiden Mutationen eine einzelne Funktion oder auch einen metabolischen Pfad beeinflussen. Der Synthetic Lethal kann auch Informationen darüber geben, welche Proteine gebraucht werden um einen bestimmten Prozess in der Zelle auszuführen [6].

Der Datentyp Synthetic Lethal (kurz SL) besteht aus einer Liste von Gen Paaren denen ein ganzzahliger Wert zugewiesen wird. Dieser Zahlenwert präzisiert die Ausprägung des Synthetic Lethal. Teilweise wird nicht nur zwischen lebensfähig und nicht lebensfähig unterschieden, sondern es werden noch Zwischenstufen angegeben.

2.6 Anmerkungen

Die Datentypen haben alle gemeinsam, dass sie sich mit Genen oder deren Genprodukten beschäftigen. Innerhalb dieses Projektes werden diese Typen alle gleich behandelt. Es wird also nicht unterschieden, ob es sich um Gene, Proteine oder auch Enzyme handelt. Wenn also ein Gen ein bestimmtes Protein synthetisiert, werden dieses Gen und sein Protein genau gleich behandelt. So werden in diesem Projekt die Gene direkt mit Proteinen oder auch Enzymen verglichen.

Um diesen Vergleich anstellen zu können, muss die Information zur Verfügung stehen, welche Gene welche Proteine synthetisieren oder welche Enzyme Proteine sind. Bei vielen Genen ist aber zum Beispiel das zugehörige Protein nicht bekannt oder ein solches Protein existiert gar nicht. Auch kann es vorkommen, dass verschiedene Gene das gleiche Protein synthetisieren. Das hat dazu geführt, dass es Proteine gibt, die unter unterschiedlichen Namen bekannt sind. Oder es kommt auch vor, dass verschiedene Forschergruppen das gleiche Gen oder auch Protein etwa zeitgleich entdeckten und diesem dann unabhängig voneinander einen unterschiedlichen Namen geben haben.

Um Vergleiche zwischen Daten unterschiedlicher Herkunft möglich zu machen, wird eine Übersetzungsliste benötigt. Diese muss die Informationen darüber enthalten, welche Gene welchen Proteinen entsprechen oder was für Synonyme zu einem bestimmten Gen existieren. Fehlt eine solche Übersetzungsliste, ist ein Vergleich der Daten nur sehr beschränkt möglich.

3 Anforderungen an der Datenpool

Ziel dieses Projektes ist es diese fünf Datentypen zu verwalten um dann mittels typenübergreifender Abfragen Teilmengen dieser Daten exportieren zu können.

3.1 Verwaltung der Daten

Die Daten stehen alle in Form von Textdateien zur Verfügung. Diese enthalten jeweils einen Satz Daten eines der fünf Datentypen oder andernfalls eine Übersetzungsliste der Gene in die Proteine. Bei den GE-Daten ist das beispielsweise eine Datei mit 100 Chips und 6'000 Genen, dessen möglichen Kombinationen jeweils ein Wert zugewiesen wird.

Eine Datei besteht aus einem Satz von Daten eines Datentypen.[3] Der Datenpool muss die Möglichkeit bieten, mehrere Datensätze eines bestimmten Datentypen zu verwalten.

Ziel ist nicht eine möglichst komplette Sammlung von Daten in diesem Datenpool zu vereinigen. Es sollen nur die Daten enthalten sein, die gerade für Analysezwecke gebraucht werden. Nicht mehr gebrauchte Datensätze werden durchaus wieder aus dem Datenpool entfernt.

3.2 Abfragemöglichkeiten Genexpressionsdaten

Der Datentyp der GE-Daten ist dadurch komplex, da ein GE-Datum aus einer Kombination von einem Gen und einem Chip besteht. Die Filterkriterien betreffen dann auch die Gene sowie die Chips.

Die Chips sind eine Art von Datentyp für sich. Zu ihnen gehören die Zusatzinformationen, handelt es sich um einen Mutanten oder einen Wildtypen, wurde ein Stress ausgeübt und handelt es sich um eine Zeitreihe. Diese Informationen müssen auf eine geeignete Art gespeichert werden, um dann anhand von diesen Zusatzinformationen Chips selektieren zu können. So zum Beispiel nach allen Zeitreihen von Wildtypen.

Die Gene besitzen keine direkten Zusatzinformationen nach denen selektiert werden könnte. Bei ihnen wird anhand der anderen Datentypen gefiltert.

Die GE-Daten sollen nach folgenden Kriterien ausgesucht werden können:

- Wähle alle Gene die keine fehlenden Werte besitzen. Diese Abfrage wird gebraucht, da für gewisse Analysen keine missing values vorkommen dürfen.
- Wähle eine zufällige Anzahl von Genen und/oder Chips aus. Dies wird benötigt um Analysemethoden auf zufällig selektierten Daten testen zu können um als Vergleichswert zu dienen.

[3] Zu den GE-Daten kann eine zweite Datei dazukommen, welche Zusatzinformationen zu den Chips enthält.

- Wähle alle Gene die Protein-Protein-Interaktionen eingehen.
- Wähle alle Gene zu denen Metabolische Pfade bekannt sind.
- Wähle alle Gene zu denen Synthetic Lethal bekannt sind.
- Wähle alle Gene zu denen eine Assoziation zu einem GO-Graphen existiert.
- Wähle alle Chips die Mutanten analysiert haben.
- Wähle alle Chips denen Stress zugeführt wurde.
- Wähle alle Chips die Zeitreihen darstellen.

Alle diese Kriterien müssen beliebig kombinierbar sein. So zum Beispiel: Wähle eine Untermenge der GE-Daten aus, bei denen keine 'missing values' vorkommen, zu dessen Genen Synthetic Lethals bekannt sind und auf deren Chips Stress ausgeübt wurde.

3.3 Abfragemöglichkeiten Gen Ontologie Annotationen

Die GOA-Daten bestehen einerseits aus dem Graphen und andererseits aus den Assoziationen der Genprodukte zu diesen Graphen. Folgende Abfragemöglichkeiten sollten möglich sein:

1. Gegeben eine Liste von Proteinen. Suche nach allen Termen zu welchen diese Gene direkt assoziiert sind.
2. Suche alle direkten Nachfolger einer Liste von Termen.
3. Suche alle transitiven Nachfolger einer Liste von Termen.
4. Suche nach allen Proteinen, die zu einer Liste von Termen assoziiert sind. Da die Terme hierarchisch gegliedert sind muss diese Suche transitiv sein.
5. *Shared Parent:* Anhand einer Liste von Termen soll der kürzest entfernte Vorgängerknoten dieser Terme gesucht werden.
6. *Union:* Abfolge der Punkte 1, 2 und 4.
7. *Intersection:* Abfolge der Punkte 1, 5, 2 und 4.

3.4 Abfragemöglichkeiten der einfachen Datentypen

Die einfacheren Datentypen (PPI, MP und SL) bestehen alle aus einem Proteinpaar, welchem je nach Datentyp noch Zusatzinformationen zugeordnet sind. Gefiltert werden die Daten anhand einer Liste von Proteinen. Es sollten dabei alle Paare herausgesucht werden, bei denen entweder mindestens eines oder andererseits beide Proteine des Paares in der Liste enthalten sind.

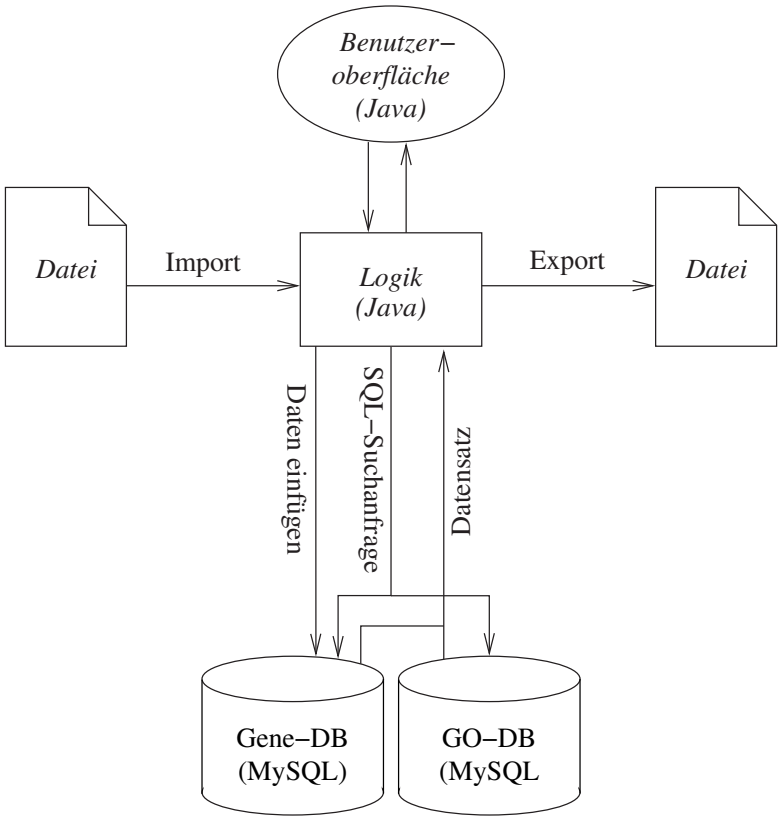

Abbildung 1: Schema des Datenpools. Im Zentrum steht dabei der Logik-Teil. Dieses Modul ist für die Kommunikation mit der Datenbank sowie dem GUI zuständig und ist für das Lesen und Schreiben der Dateien verantwortlich.

4 Architektur

In diesem Abschnitt wird die Architektur des Datenpools, dargestellt in Abbildung 1, erläutert. In Abschnitt 4.1 wird darauf eingegangen in welcher Form die Daten gespeichert werden. In Abschnitt 4.2 wird erläutert, wieso die GO-Daten speziell behandelt werden. Abschliessend wird in Abschnitt 4.3 der Aufbau der Applikation erklärt.

4.1 Datenbank

Für die Speicherung und Verwaltung der Daten wird eine relationale Datenbank benutzt. Diese bietet alle Funktionen, die für dieses Projekt gebraucht werden:

- Die Datentypen können unabhängig von ihrer Quelle in einheitlicher Form gespeichert werden.

- Alle bisher benötigten Filter können mit einer einzelnen SQL-Abfragen umgesetzt werden[4]. Dadurch wird die Applikation sehr flexibel wenn es darum geht, neue Anfragen zu definieren.
- Unterstützung aller gängigen Betriebssysteme.
- Relationale Daten sind sehr weit verbreitet, somit ist Support bei Problemen garantiert.

Relationale Datenbanksysteme gibt es von verschiedenen Anbietern. Wie Oracle von der gleichnamigen Firma, DB2 von IBM, Access von Microsoft, FileMaker von FileMaker Inc. oder MySQL die von der Open Source Gemeinschaft entwickelt wird. Die Wahl fiel schliesslich auf die Open Source Datenbank MySQL, da diese alle benötigten Möglichkeiten bietet und keine Lizenzkosten zu tragen sind. Die MySQL Datenbank birgt aber auch Nachteile:

- Die Performance der MySQL Datenbank kommt nicht an die einer Oracle Datenbank heran.
- Im Vergleich zur Oracle Datenbank ist MySQL im Funktionsumfang limitiert. So ist es mit der aktuellen Version 4.1 nicht möglich *views* zu definieren, was für das Projekt noch nützlich gewesen wäre.
- Bei einer proprietären Datenbank wird alle benötigte Software in einem Packet geliefert. Bei MySQL muss die benötigte Software zusammengesucht werden.

Da wie erwähnt die Performance in dieser Applikation nicht eine übergeordnete Rolle spielt, wiegt dieser Nachteil nicht so schwer. Und auch der etwas begrenzte Funktionsumfang ist nicht weiter problematisch, da die fehlenden Komponenten ebenfalls hauptsächlich zur Performance-Steigerung nützlich gewesen wären. Die in diesem Projekt erstellte Datenbank wird zukünftig als Gen-Datenbank bezeichnet.

4.2 Gene Ontology Database

Die GO-Daten bestehen einerseits aus drei Ontologien, dargestellt in gerichteten azyklischen Graphen und andererseits aus den Zuordnungen von Genen zu den Knoten des Graphen. Siehe Abschnitt 2.2 für Details. Die erforderlichen Suchanfragen, beschrieben in Abschnitt 3.3, erfordern teilweise rekursives Suchen durch den Graphen. Solche rekursiven Anfragen werden zwar seit dem SQL:1999 Standard unterstützt, jedoch ist dieser noch in den wenigsten Datenbanksystemen, so auch nicht von MySQL, vollständig implementiert. Es gibt zwei Möglichkeiten diese Abfragen trotzdem auszuführen.

[4]Das mit der Ausnahme der 'Shared Parents' für die GO-Daten, die eine Rekursion bedingt, was in SQL noch nicht realisierbar ist.

Entweder wird die Rekursion mit Hilfe von Software-Logik ausgeführt oder es werden alle möglichen Pfade durch den Graphen im Voraus berechnet und in geeigneter Form gespeichert.

Das GO Consortium [3], stellt eine Datenbank zur Verfügung, welche alle vorhandenen GO-Daten, inklusive aller möglichen Pfade durch den Graphen, enthält. Mit dieser Datenbank sind alle in Abschnitt 3.3 beschriebenen Abfragen mit einem einzigen SQL-Befehl realisierbar.[5]

Da die GO-Datenbank alle gewünschten Funktionen bereitstellt, wird diese in den Datenpool integriert. Dafür wird lokal eine Kopie der GO-Datenbank gespeichert. Diese enthält nur diejenigen Daten, die für die gewünschten Anfragen gebraucht werden. Es wird also nicht die komplette GO-Datenbank gespeichert.

Das Benutzen der GO-Datenbank bringt sehr viele Vorteile mit sich, birgt aber einen nicht unerheblichen Nachteil. Die Gene sind in der GO-Datenbank mit einem anderen Schlüssel referenziert als in der Gen-Datenbank. Das hat zur Folge, dass Suchanfragen zwischen den beiden Datenbanken nur indirekt möglich sind. Um die Kompatibilität zu gewährleisten, müsste die GO-Datenbank mit den Schlüsseln der Gen-Datenbank gefüttert werden. Das ist prinzipiell möglich, konnte aber aus Zeitgründen nicht mehr implementiert werden.

4.3 Applikation

Auf die Datenbank wird mittels einer Applikation zugegriffen. Als Entwicklungssprache wird Java benutzt. Java verfügt über eine gute Anbindung an MySQL, ist Plattform unabhängig, bietet eine Vielfalt von freien Entwicklungsumgebungen und ermöglicht eine relativ schnelle Entwicklung von Oberflächen. Java ist jedoch nicht gerade für seine Performance bekannt. Da die rechenintensiven Berechnungen alle von der Datenbank erledigt werden, ist das soweit kein Problem.

Die Applikation wird in zwei Module, die Logik und die Benutzerschnittstelle, unterteilt. Die Logik ist dabei die zentrale Einheit und ist für die ganze Kommunikation verantwortlich. Über sie werden die Daten eingelesen und exportiert und es wird auf die Datenbank zugegriffen. Über die Benutzerschnittstelle kann der Benutzer den Datenpool benutzen.

[5] Dies mit Ausnahme der 'shared parents', welche nur für zwei Terme mit einem SQL-Befehl realisierbar ist. Sind mehr als zwei Terme gegeben, muss die Abfrage rekursiv bearbeitet werden.

5 Umsetzung

In diesem Abschnitt wird die genaue Umsetzung des Datenpools erläutert. In Abschnitt 5.1 wird die Struktur der Datenbank erklärt und in Abschnitt 5.2 die wichtigsten Aspekte zur Applikation besprochen.

5.1 Design Datenbank

Neben den in Abschnitt 2 beschriebenen Datentypen werden für die Umsetzung noch weitere Daten gebraucht.

- Eine Übersetzungsliste für die Gene in Proteine, bzw. auch Gene in Gene, da wie in Abschnitt 2.1 beschrieben das gleiche Gen unter verschiedenen Namen bekannt sein kann.
- Die Daten gehören jeweils zu einem Organismus. Es sollte also eine Liste mit den Organismen geben.
- Praktisch wäre noch, wenn die importierten Datensätze einem Benutzer zugewiesen werden könnten. Dafür ist eine kleine Benutzerverwaltung notwendig.

Die letzten beiden Punkte sind sehr einfach umgesetzt. Die Organismen sind in einer der Tabelle '*organism*' abgespeichert. Das Format dieser Tabelle ist in Tabelle 1 abgebildet. Das Feld *name* ist für die Menschen lesbare Bezeichnung gedacht, während das Feld *id* für die interne Referenzierung gebraucht wird.

Die Benutzer werden in der Tabelle '*user*' verwaltet, dessen Struktur in Tabelle 2 gezeigt ist. Der Benutzername dient dabei direkt als Referenz. Es gibt kein Passwort oder ähnliche Zusätze. Ziel ist es, eine einfache Möglichkeit zu haben, die Datensätze der verschiedenen Benutzer bei der Selektion unterscheiden zu können.

5.1.1 Synonyme der Proteine

Wie bereits angesprochen gibt es keine direkte Gen-Protein Korrelation. Einerseits gibt es Gene die keine Proteine synthetisieren und andererseits kann es vorkommen, dass das gleiche Protein unter verschiedenem Namen bekannt ist. Siehe Abschnitt 2.6 für weitere Details.

Zur Zeit existiert leider noch kein Standard in der Genbezeichnung der sich durchsetzen konnte. Zumindest gibt es aber für die am meisten erforschten Organismen einen eigenen Standard. So zum Beispiel die SGD-Bezeichnung bei der Hefe [13]. Eine solche Standardbezeichnung, sofern vorhanden, dient als Referenz für die Gene. Verwaltet werden diese in der Tabelle '*synonym*', dessen Struktur in Tabelle 3 gezeigt wird. Falls eine solche

Referenz fehlt, ist jede Bezeichnung für sich selbst die Referenz.[6]

Häufig ist zu dieser Standardbezeichnung eine Liste mit allen gebräuchlichen Synonymen für die verschiedenen Gene verfügbar. Diese Synonyme werden in der Tabelle '*designation*' gespeichert, dessen Format in Tabelle 4 abgebildet ist. So können beliebig viele Gennamen der vorhin erwähnten Referenzbezeichnung zugeordnet werden. Damit wird es möglich das gleiche Gen, welches in verschiedenen Datensätzen unterschiedlich bezeichnet wird, als ein und dasselbe zu identifizieren.

Die Gene werden in der Datenbank anhand eines internen Schlüssels referenziert. Sollen Daten exportiert werden, kann nur nach dem Referenzwert gemäss Tabelle '*synonyms*' übersetzt werden. Die Bezeichnungen in der Tabelle '*designation*' besitzt je nach Gen gar keine oder auch mehrere Übersetzungen. Dies verunmöglicht eine Übersetzung der internen Schlüssel nach diesen Zusatzbezeichnungen.

5.1.2 Datensätze

Bei den Datentypen GE, PPI, SL und MP die Daten in Form von Datensätzen zusammengefasst.

Die Datensätze werden für jeden Datentyp in einer separaten Tabelle verwaltet, welche unabhängig vom Datentyp immer die gleiche Form besitzt. Diese ist am Beispiel der PPI-Datensätze in Tabelle 5 gezeigt. Zu jedem Datensatz gehört: ein interner Schlüssel, der als Referenz dient, eine Beschreibung, das Importdatum, der Organismus und der Benutzer der den Datensatz erstellt hat. Die Datensatz-Tabellen bieten eine gute Übersicht über die vorhandenen Daten der verschiedenen Datentypen.

Da diese Tabellen alle die gleiche Struktur besitzen ist es prinzipiell möglich diese Tabellen in eine zusammenzufassen. Das hätte zur Folge, dass die '*id*' der Datensatztabelle auf verschiedene Tabellen referenziert, was designtechnisch unschön ist.

Zusätzlich zu den Datentypen können Listen von Genen in der Datenbank verwaltet werden. Eine solche Liste wird, wie bei den Datentypen, einem Datensatz zugeordnet. Alle Gene des einen Datensatzes gehörten dann zu einer solchen Liste von Genen. Diese werden in der Tabelle '*gene*' verwaltet, dessen Struktur gemäss Tabelle 6 gegeben ist. Die Datensätze der Genlisten werden in der Tabelle '*gene_set*' gemäss Tabelle 5 verwaltet.

5.1.3 Einfache Datentypen

Die einfachen Datentypen (PPI, MP und SL) haben alle eine sehr ähnliche Struktur. Es ist jeweils ein Proteinpaar welchem je nach Datentyp Zusatz-

[6] In diesem Fall ist es nicht möglich zwei unterschiedlich bezeichnete Gene als das gleiche zu identifizieren.

informationen in Form einer Zeichenkette oder eines ganzzahligen Wertes zugeordnet ist.

Diese drei Datentypen liegen alle einem Genpaar zu Grunde. Die Datenstruktur ist demnach für diese Typen sehr ähnlich. Für die PPI-Daten ist sie in Tabelle 7 gezeigt. Zu jeder Gen-Kombination, dargestellt durch die beiden Referenzen '*gene1_id*' und '*gene2_id*', gehört die '*set_id*' welche das Genpaar einem Datensatz zuordnet. Die Zusatzinformation zu den PPI-Daten wird im Feld *experiment_type* gespeichert. Die Datenstruktur sieht für die MP-Daten (siehe Tabelle 8) und die SL-Daten (siehe Tabelle 9) sehr ähnlich aus.

Wegen ihrer Ähnlichkeit, könnte man die drei Datentypen problemlos in einer Tabelle verwalten. Dies ist vor allem für den Fall sinnvoll, wenn es sehr viele solche einfachen Datentypen gäbe. Da es aber nur drei dieser Datentypen sind, ist es einiges übersichtlicher diese auf drei[7] Tabellen zu verteilen.

5.1.4 Genexpressionsdaten

Die GE-Daten bestehen aus einer Kombination von einem Gen mit einem Chip denen ein Zahlenwert zugeordnet ist. Es können jedoch 'missing values' auftreten, bei denen dieser Zahlenwert fehlt. Siehe Abschnitt 2.1 für Details.

Für die Speicherung der GE-Daten wurden drei verschiedene Modelle in Betracht gezogen:

1. Die Daten werden als Kombination eines Genes, eines Chips und einem Zahlenwert gespeichert.

   + Es können sehr flexible Suchanfragen gestellt werden, welche die Gene und die Chips betreffen
   − Die Tabelle wird schnell einige Millionen Zeilen enthalten, was zu Performance Problemen führen kann.

2. Alle Informationen über einen einzelnen Chip werden in einer eigenen Tabelle gespeichert. Die GE-Daten sind also auf hunderte verschiedene Tabellen aufgeteilt.

   + Sehr schnelle Abfrage aller Informationen eines einzelnen Chips möglich.
   − Es ist aufwändig Daten von mehreren Chips zu selektieren, da dafür für jeden Chip eine Suchanfrage für die betreffende Tabelle abgesetzt werden muss.

3. Die Chips entsprechen den Kolonnen der Tabelle, die Gene den Zeilen. Für jedes Gen gibt es einen Datensatz mit mehreren hundert Feldern.

[7] Beziehungsweise auf sechs, wenn man die Datensatz Tabellen dazu zählt.

+ Alle Daten sind in einer einzigen Tabelle, welche nur wenige Zeilen enthält.
– Die Selektion der Chips ist sehr aufwändig.
– Wird ein neuer Chip hinzugefügt, muss die Struktur einer bereits mit Daten gefüllte Tabelle geändert werden.

Implementiert wurde die erste Variante, da diese am Flexibelsten ist wenn es um die Suchanfragen geht. Das Problem mit der Performance darf aber nicht unterschätzt werden, bekommt man aber mit richtig gesetzten Indizes gut in den Griff. Das Schema für die Tabelle mit den GE-Daten ist in Tabelle 10 abgebildet. Zu jedem Gen-Chip Paar gibt es eine '*set_id*' welche den Eintrag einem Datensatz zuordnet.

Wie bereits erwähnt, müssen auch fehlende Werte als solche erkannt werden können. Für die Umsetzung dieser 'missing values' wurden drei Varianten analysiert:

1. Eine 'Null' anstelle des Wertes einfügen.
2. Einen unmöglich vorkommenden Wert einfügen, der als ein 'missing value' gilt.
3. Die fehlenden Werte in einer separaten Tabelle verwalten.

Da Null-Werte in der Datenbank häufig zu Problemen führen, speziell wenn es um Suchanfragen geht, und ein Standardwert zu setzten sehr unsauber ist, wurde die dritte Variante umgesetzt. Die fehlenden Werte werden in einer zusätzlichen Tabelle '*ge_missing*' verwaltet, dessen Struktur in Tabelle 11 gezeigt wird.

Die Chips verfügen oft über zusätzliche Informationen, zum Beispiel, ob es sich um einen Wildtypen handelt oder nicht. Diese Zusatzinformationen zu den Chips werden in der Tabelle '*chip*' verwaltet. Die Struktur ist in Tabelle 12 abgebildet. Zu jedem Chip gibt es eine interne Referenz, eine Beschreibung in Menschen lesbarer Form, und den Zusatzinformationen über Mutant, Stress und Zeitreihe. Zudem wird jeder Chip über die '*set_id*' einem Datensatz zugewiesen. Damit wird es möglich Chips mit speziellen Eigenschaften zu selektieren.

Die Informationen über den Datensatz sind in der Tabelle '*chip*' und in der Tabelle '*ge*' bzw. '*ge_missing*' gespeichert. Diese Information ist also doppelt in der Datenbank vorhanden, da die Tabellen miteinander verknüpft sind. Das ist unschön, steigert aber die Performance ungemein und erleichtert die Suchanfragen.

5.2 Applikation

Bei der Entwicklung der Applikation wurden folgende Design-Ziele verfolgt:

Feld	Typ	Null	Referenz
id	smallint(6)	Nein	
name	varchar(50)	Nein	

Tabelle 1: Struktur der Tabelle *organism*

Feld	Typ	Null	Referenz
id	varchar(10)	Nein	

Tabelle 2: Struktur der Tabelle *user*

Feld	Typ	Null	Referenz
id	mediumint(9)	Nein	
reference	varchar(128)	Nein	
organism_id	smallint(6)	Nein	organism (id)

Tabelle 3: Struktur der Tabelle *synonym*

Feld	Typ	Null	Referenz
gene_id	mediumint(9)	Nein	synonym (id)
name	varchar(128)	Nein	

Tabelle 4: Struktur der Tabelle *designation*

Feld	Typ	Null	Referenz
id	smallint(6)	Nein	
organism_id	smallint(6)	Nein	organism (id)
timestamp	date	Nein	
user_id	varchar(10)	Nein	user (id)
description	varchar(255)	Nein	

Tabelle 5: Struktur der Tabelle *ppi_set*

Feld	Typ	Null	Referenz
set_id	smallint(6)	Nein	gene_set (id)
gene_id	mediumint(9)	Nein	synonym (id)

Tabelle 6: Struktur der Tabelle *gene*

Feld	Typ	Null	Referenz
set_id	smallint(6)	Nein	ppi_set (id)
gene1_id	mediumint(9)	Nein	synonym (id)
gene2_id	mediumint(9)	Nein	synonym (id)
experiment_type	varchar(3)	Nein	

Tabelle 7: Struktur der Tabelle *ppi*

Feld	Typ	Null	Referenz
set_id	smallint(6)	Nein	ppi_set (id)
gene1_id	mediumint(9)	Nein	synonym (id)
gene2_id	mediumint(9)	Nein	synonym(id)

Tabelle 8: Struktur der Tabelle *mp*

Feld	Typ	Null	Referenz
set_id	smallint(6)	Nein	ppi_set (id)
gene1_id	mediumint(9)	Nein	synonym (id)
gene2_id	mediumint(9)	Nein	synonym (id)
value	tinyint(4)	Nein	

Tabelle 9: Struktur der Tabelle *sl*

Feld	Typ	Null	Referenz
set_id	smallint(6)	Nein	ge_set (id)
gene_id	mediumint(9)	Nein	synonym (id)
chip_id	int(11)	Nein	chip (id)
value	float	Nein	

Tabelle 10: Struktur der Tabelle *ge*

Feld	Typ	Null	Referenz
set_id	smallint(6)	Nein	ge_set (id)
gene_id	mediumint(9)	Nein	synonym (id)
chip_id	int(11)	Nein	chip (id)

Tabelle 11: Struktur der Tabelle *ge_missing*

Feld	Typ	Null	Referenz
id	int(11)	Nein	
caption	varchar(100)	Nein	
set_id	smallint(6)	Nein	ge (set_id)
mutant	tinyint(4)	Nein	
stress	tinyint(4)	Nein	
timerank	tinyint(4)	Nein	

Tabelle 12: Struktur der Tabelle *chip*

- Gute Übersicht über die vorhandenen Daten.
- Häufig gestellte Abfragen sollen einfach und schnell absetzbar sein.
- Spezielle und nicht vorhergesehene Abfragen sollen ausführbar sein.
- Schlank und übersichtlich.

Wie in Abbildung 1 dargestellt besteht die Applikation aus einem Logikteil und einer graphischen Benutzeroberfläche.[8] In den nächsten beiden Abschnitten werden diese beiden Module besprochen.

5.2.1 Logik Modul

Das Logikmodul ist die zentrale Einheit des Datenpools. Sie bildet die Verbindung zwischen der Benutzeroberfläche und der Datenbank, ist für den Import und Export der Daten zuständig, und enthält einige wenige Algorithmen.

Um mit der Datenbank kommunizieren zu können, muss eine Verbindung aufgebaut werden. Da zusätzlich auf die GO-Datenbank zugegriffen wird, müssen zwei solche Verbindungen aufrechterhalten werden. Über diese Verbindungen werden alle Suchanfragen und Datenmodifikationen ausgeführt. Sie werden bei Applikationsstart aufgebaut und erst beim Beenden wieder getrennt. Sollte eine Verbindung einmal getrennt werden, ist ein Neustart der Applikation erforderlich.

Die Daten stehen in Form von Tabulator getrennten Dateien zur Verfügung. Das Format dieser Dateien ist in Abschnitt C beschrieben. Neben den Datentypen können Dateien gelesen werden, welche die Synonyme der Gene enthalten. Im Gegensatz zu den Datentypen ist diese Information in der Datenbank aktualisierbar. Die Datentypen hingegen können nur als ganzes Importiert werden. Gibt es Änderungen in einem Datensatz, kann anstelle eines Updates die ganze Datei neu eingelesen und damit der alte Datensatz überschrieben werden.

Die Suchanfragen an die Datenbank können mit einem einzigen SQL-Befehl ausgeführt werden.[9] Mit einer solchen Suchanfrage wird häufig eine Untermenge eines Datensatzes eines Datentypen selektiert. So zum Beispiel die Anfrage: Wähle alle Genexpressionsdaten dessen Gene entweder in einer Liste von Genen enthalten ist oder Protein-Protein-Interaktionen eingehen. Der SQL-Befehl dieser Abfrage ist im Listing 3 zu finden. Ein solcher SQL Befehl wird von Logik an die Datenbank geleitet und dort ausgeführt. Je nach Komplexität der Suchanfrage kann das einige Minuten Zeit in Anspruch nehmen. Die Datenbank liefert anschliessend das Resultat zurück an die Logik,

[8] Es ist auch denkbar das GUI durch eine Konsole zu ersetzen die Batch-Verarbeitung erlaubt.

[9] Es kann natürlich sein, dass künftige Anforderungen mehrere Befehle benötigen.

welche die Information entweder an die Benutzeroberfläche weiterleitet oder in eine Datei Exportiert. Beim Export ist es unter Umständen nötig, dass der interne Schlüssel der Gene in Menschen lesbare Form umgesetzt wird. Dafür werden je nachdem weitere SQL-Befehle an die Datenbank geschickt.[10]

5.2.2 Graphische Benutzeroberfläche

An die Benutzeroberfläche werden folgende Ansprüche gestellt:

- Sehr häufig vorkommende Aufgaben sollten schnell und einfach erledigt werden können.
- Die Benutzeroberfläche soll flexibel genug sein, auch selten vorkommende Aufgaben auszuführen.
- Einfach und verständlich für den Benutzer.

Um das zu realisieren gibt es Dialoge die ermöglichen sehr frei, in Form eines SQL-Befehls, Daten zu selektieren. Andere Dialoge erlauben schon vorgefertigte Anfragen abzusetzen. Dabei können diese Befehle problemlos von Hand speziellen Gegebenheiten angepasst werden. Eine Beschreibung der verschiedenen Dialoge ist in Abschnitt A zu finden.

[10] Es ist natürlich möglich diese Übersetzung direkt in die Ursprüngliche Suchanfrage zu integrieren. Das würde aber die SQL Befehle markant komplexer machen, was dem Endbenutzer nicht zugemutet werden soll.

Literatur

[1] UniProt *http://www.ebi.ac.uk/uniprot*

[2] Gene Expression Omnibus *http://www.ncbi.nlm.nih.gov/geo*

[3] Gene Ontology Consortium *http://www.geneontology.org*

[4] P. Uetz und E. Pohl. Protein-Protein- und Protein-DNA-Interaktionen. *In: M. Wink Molekulare Biotechnologie,* Wiley-VCH, pp. 385–407, 2004.

[5] Prof. Dr. Erhard Rahm. Pathway-Datenbanken. *Universität Leipzig Institut für Informatik,* Januar 2003.

[6] Synthetic Lethal Mutations *http://www.sci.sdsu.edu/ smaloy/MicrobialGenetics/topics/rev-sup/synthetic.html*

[7] Swiss-Prot *http://www.ebi.ac.uk/swissprot*

[8] TrEMBL *http://www.ebi.ac.uk/trembl*

[9] Protein Information Resource *http://pir.georgetown.edu*

[10] European Bioinformatics Institute *http://www.ebi.ac.uk*

[11] Array Express *http://www.ebi.ac.uk/arrayexpress*

[12] Nucleotide Sequence Database *http://www.ebi.ac.uk/embl*

[13] Saccharomyces Genome Database *http://www.yeastgenome.org*

[14] W. Miriam und K. H. Scharf. Biologie heute SII *Schroedel, ISBN: 3–507–10540–3, Neuberarbeitung,* 1997.

[15] C. Ullenboom. Java ist auch eine Insel *Galileo Press GmBH, ISBN: 3–89842–304–2, 3. Auflage,* 2003.

[16] MySQL *http://www.mysql.com*

[17] phpMyAdmin *http://www.phpmyadmin.net*

A Softwarebeschreibung

In diesem Abschnitt werden die Anforderungen an die Applikation aufgelistet. Danach folgt eine Installations- und Bedienungsanleitung für die Applikation.

A.1 Anforderungen

Die Software benötigt eine MySQL Datenbank. Diese kann lokal auf der Arbeitsstation oder auf einem Server installiert sein. Folgende Anforderungen gelten für die Datenbank:

- MySQL Version 4.1 oder höher
- Default Character Set: utf8. Oder es muss darauf geachtet werden, dass die für diese Applikationen benutzten Datenbanken auf uft8 gesetzt sind.
- Zwei leere Datenbanken '*gene*' und '*go*' und ein Benutzer der über vollständige Lese- und Schreibrechte auf diese beiden Datenbanken verfügt. Alternativ genügt auch ein Benutzer der das Recht hat neue Datenbanken zu erstellen.
- phpMyAdmin mit pma-Unterstützung oder eine andere Anwendung die erlaubt direkt auf die Datenbank zuzugreifen. Alternativ kann auch die bei MySQL Konsole benutzt werden. Dies erfordert aber sehr gute SQL Kenntnisse
- Es sollte mindestens 3 GByte freier Speicherplatz vorhanden sein.
- Die Suchanfragen können den Rechner ziemlich auslasten. Es sollten also keine zeitkritischen Applikationen neben der Datenbank am laufen sein.

Auf der Arbeitsstation muss Java 2 lauffähig sein.

A.2 Installation

1. Schritt Für die in diesem Projekt entwickelte Gen-Datenbank müssen als erstes die Tabellen erstellt werden. Die passenden SQL-Befehle sind auf der CD unter:

/installation/gene_db/createtables.sql

zu finden.

2. Schritt Die Benutzer und die Organismen werden nicht von der Applikation aus verwaltet. Die Verwaltung erfolgt direkt auf der Datenbank. Benutzer und Organismen können jederzeit hinzugefügt werden. Beim Löschen hingegen muss man sehr vorsichtig sein, da unter Umständen noch Daten in der Datenbank mit diesen Einträgen verlinkt sind.

3. Schritt Die Benutzer der Applikation werden in die Tabelle '*user*' werden. Dabei kann ein beliebiger alphanumerischer Wert, der maximal zehn Zeichen lang ist verwendet werden.

4. Schritt Die Organismen werden direkt in der Datenbank eingetragen. Dazu wird dem Organismus im Feld '*id*' ein freier Schlüssel und ihm im Feld '*name*' ein Name zugewiesen.

5. Schritt Soll auch auf die GO-Daten zugegriffen werden können muss eine lokale Kopie der Datenbank installiert werden. Die aktuelle Datenbank ist unter

http://www.godatabase.org/dev/database/archive/latest

zu finden. Falls der Link geändert hat gelangt man über

http://www.geneontology.org

auf die richtige Seite. Für die Applikation wird nicht die die komplette Datenbank benötigt. Die verwendeten Daten sind alle in folgender Datei vorhanden:

go-<yyyymm>-assocdb-tables.tar.gz.

Eine Kopie dieser Datei vom November 2004 ist auf der CD unter

/installation/go_ db/go_ 200411-assocdb-tables.tar.gz

zu finden. Auf Unix ist die Datenbank mit den Befehlen gemäss Listing 1 installierbar. Für Windows helfen unter Umständen die Batch-Dateien die auf der CD unter

/installation/go_ db/windows/

zu finden sind weiter.

Listing 1: Installation der GO-Datenbank unter Unix.

```
tar -zxvf go-<yyyymm>-assocdb-tables.gz
cd <releasedir>
echo "create database go" | mysql
cat *.sql | mysql go
mysqlimport -L go *.txt
```

A.3 Graphische Benutzeroberfläche – GUI

Die Applikation wird über die Klasse *genedb.class* aus dem Packet *genedb* gestartet. Über Befehlszeilenargumente können die Argumente für das Login übergeben werden. Der erste Wert entspricht dem Benutzernamen. Als

zweites Argument wird der Host übergeben auf dem die MySQL-Datenbank zu finden ist. Als drittes Argument kann der Benutzername für die Datenbank und schliesslich als viertes Argument, kann noch das Passwort für die Datenbank angegeben werden.

Sobald die Applikation gestartet wurde erscheint der *Login Dialog*. Siehe Abbildung 2. Falls eine Verbindung zur Gen-Datenbank und zur GO-Datenbank aufgebaut werden kann und der angegebene Benutzer in der Gen-Datenbank eingetragen ist, erscheint das Hauptfenster. Siehe Abbildung 3. Von diesem Fenster aus können alle weiteren Dialoge geöffnet werden. Im oberen Teil des Fensters wird das Nachrichten-Log angezeigt während im unteren Feld das Error-Log zu sehen ist. Wenn im Error-Log eine Zeile hinzugefügt wird, ertönt zusätzlich ein akustisches Signal, das akustisch auf den Fehler aufmerksam macht.

A.4 Datensätze

Der in Abbildung 4 abgebildete Dialog bietet eine gute Übersicht über die vorhandenen Daten in der Datenbank.

A.4.1 Import

Der Dialog für den Datenimport ist in Abbildung 5 abgebildet. Damit die Übersetzung der Gene richtig funktioniert, ist es zwingend notwendig, dass die Übersetzungsliste eines Organismus vor allen anderen Daten in die Datenbank eingelesen wird. Sonst ist der Vergleich von Daten verschiedener Herkunft sehr limitiert.

Die Liste mit den Synonymen kann jederzeit neu importiert werden. Neue Einträge in der Liste werden dabei hinzugefügt, die alten beigelassen. Es wird aber auf keinen Fall ein Eintrag gelöscht, auch wenn er nicht mehr in der Liste sein sollte. Dies um Dateninkonsistenz zu vermeiden.

Werden GE-, PPI-, MP-, SL-Daten oder eine einfache Gen-Liste importiert, werden diese Daten einem Datensatz zugeordnet. Die Datensatznummer *setID*, kann dabei vom Benutzer selbst gewählt werden. Existiert bereits ein Datensatz mit dieser Nummer, wird der alte Datensatz komplett gelöscht. Aktualisiert kann man diese Daten nicht. Es kann aber, wie oben angedeutet, ein alter Datensatz komplett überschrieben werden.

Die Gene müssen beim Import in einen internen Schlüssel übersetzt werden. Dafür muss angegeben werden, ob die Gene in der einzulesenden Datei mit dem Referenzwert oder einem Synonym davon bezeichnet sind. Sollte beim Einlesen einige der Gene nicht bekannt sein, werden sie automatisch als neue Referenzwerte in die Datenbank eingefügt.

Für die GE-Daten können noch Zusatzinformationen über die Chips angegeben werden. Dazu gehören die Informationen *Mutant*, *Stress* und *Timerank*. Der Wert *Timerank* ist dabei nicht ganz selbsterklärend. Wird

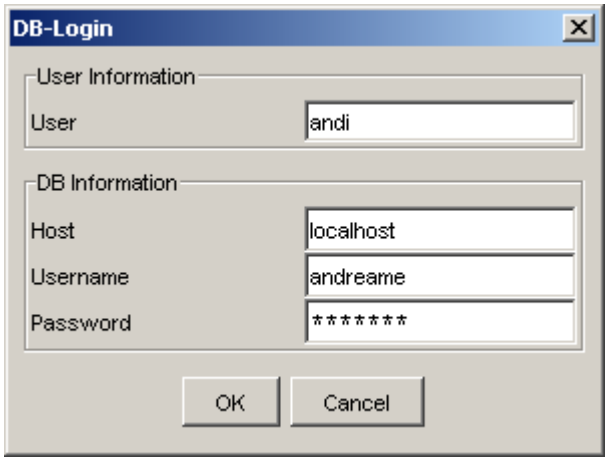

Abbildung 2: In diesem Dialog wird der Username für die Applikation, sowie die Details zur Datenbankverbindung angegeben.

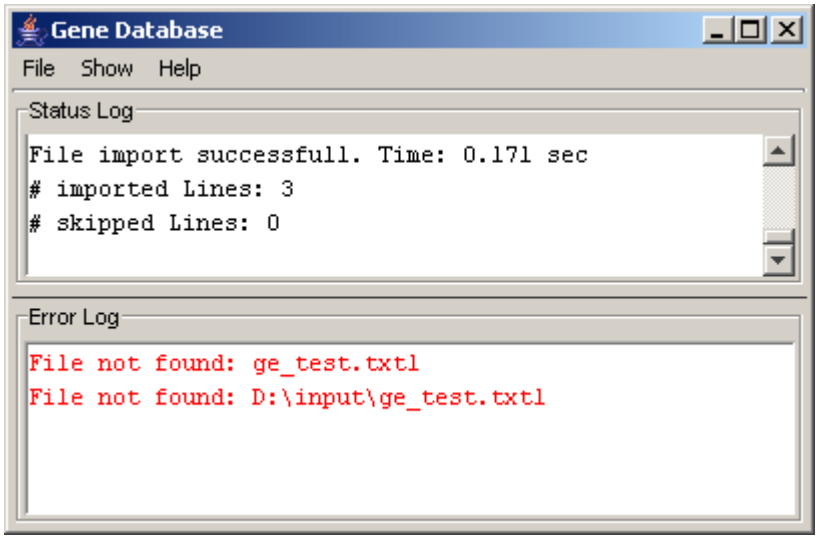

Abbildung 3: Von diesem Fenster aus werden alle weiteren Dialoge geöffnet. Im oberen Bereich wird das Nachrichten-Log angezeigt, unterhalb das Error-Log.

er ausgewählt, wird dem ersten Chip rank 0, dem zweiten rank 1 usw. zugeordnet.

Die Chips können auch in einer zusätzlichen Datei beschrieben sein. Falls eine solche Datei vorhanden ist, muss deren Pfad angeben werden und den Chips werden die Zusatzinformationen anhand dieser Datei zugewiesen. Ist keine solche Datei angegeben, werden die Chips gemäss der aktuellen Einstellung erfasst.

A.4.2 Export der einfachen Datentypen

Dieser Dialog ist in Abbildung 6 abgebildet. Die einfachen Datentypen können anhand einer Gen-Liste gefiltert werden. Dafür muss die Datensatznummer dieser Gen-Liste angegeben werden. Für die Selektion gibt es drei Modi.

- Alle Daten eines Datensatzes werden exportiert.
- Es werden alle Daten exportiert, bei denen mindestens eines der beiden Gene in der Gen-Liste vorhanden ist.
- Es werden alle Daten exportiert, bei denen beide Gene in der Gen-Liste vorhanden sind.

Die Schaltfläche *Create Query* erstellt anhand der gewählten Eigenschaften den auszuführenden SQL-Befehl. Die Schaltfläche *Count* zählt die Records die mit dem aktuellen SQL-Befehl selektiert werden.

A.4.3 Export der Synonyme

Der in Abbildung 7 gezeigte Dialog erlaubt den Export einer Liste aller Synonyme eines Organismus.

A.4.4 Export Genexpressionsdaten

Der Dialog für den Export der Genexpressionsdaten ist in Abbildung 8 abgebildet. Die GE-Daten bestehen aus einer Gen-Chip Kombination. Bei diesem Datentyp kann also anhand der Gene und/oder anhand der Chips gefiltert werden.

Die Chips werden nur mit der *where_definition* des SQL-Befehls gefiltert. Ein Beispiel dafür ist bereits im entsprechenden Feld eingetragen. Diese Filterkriterien werden mittels booleschen Operatoren verknüpft. Dabei kann anhand der Zusatzinformation *Mutant*, *Stress* und *Timerank* gefiltert werden. Bei *Mutant* und *Stress* stehen jeweils drei Werte zum Filtern zur Verfügung:

-1 Es ist nicht bekannt ob es sich um einen Mutanten handelt respektive ob dem Organismus Stress hinzugefügt wurde.

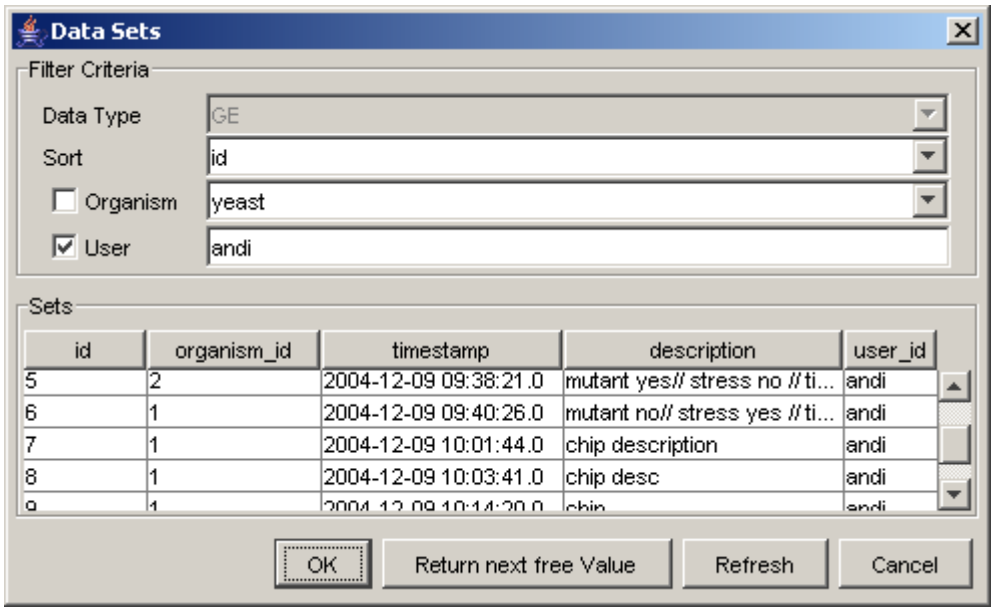

Abbildung 4: Dieser Dialog zeigt eine Übersicht über die vorhandenen Datensätze der verschiedenen Datentypen an.

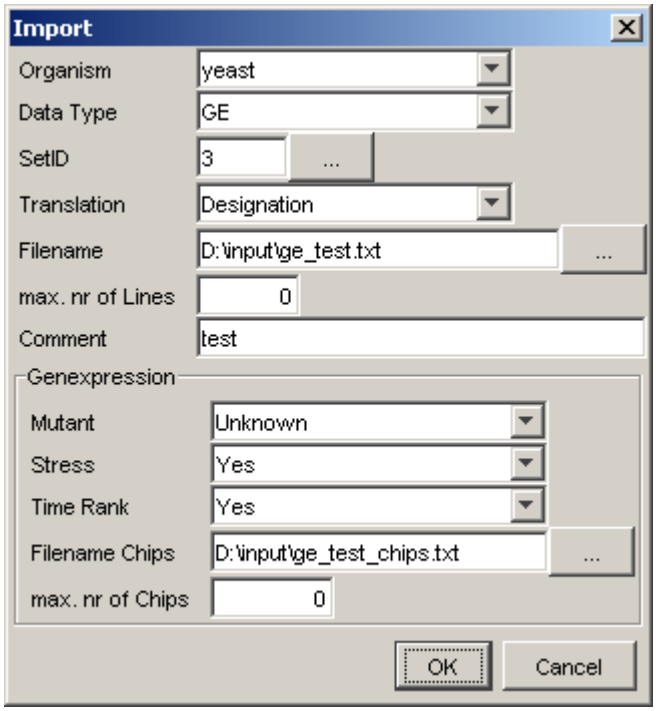

Abbildung 5: Über diesen Dialog werden Daten in Form von Textdateien importiert.

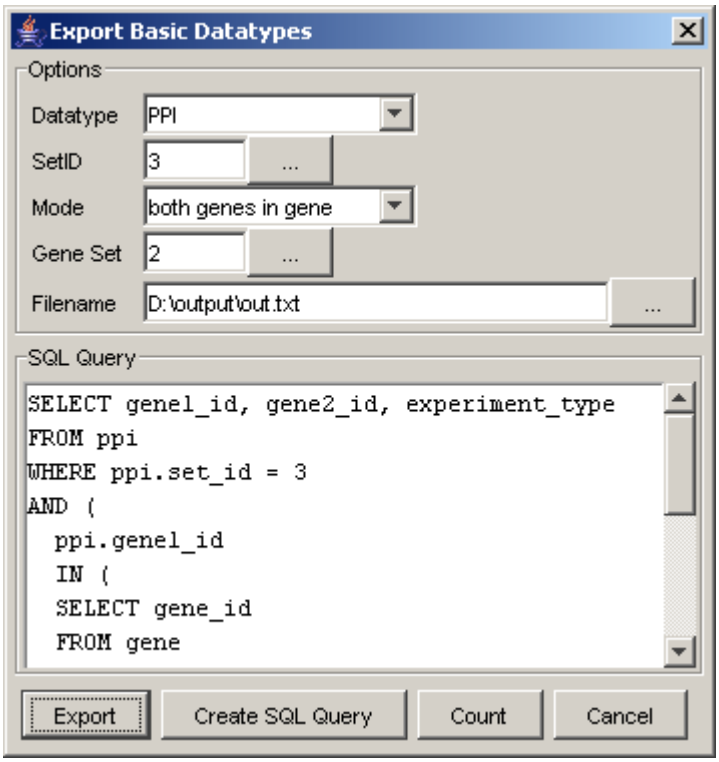

Abbildung 6: Mit Hilfe dieses Dialoges ist es möglich die Datentypen PPI, MP und SL zu filtern und zu exportieren.

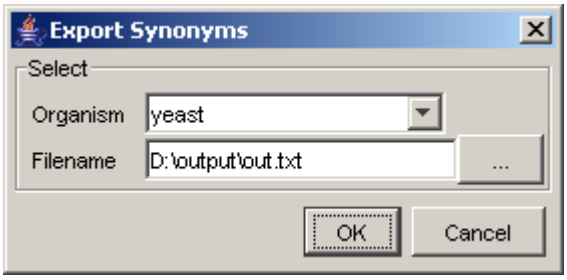

Abbildung 7: Es wird eine Liste mit allen Synonymen eines Organismus erstellt.

0 Es handelt sich um einen Wildtypen; Dem Organismus wurde kein Stress beigefügt.

1 Es handelt sich um einen Mutanten. Dem Organismus wurde Stress beigefügt.

Beim *Timerank* bedeutet eine -1 das es sich um keine Zeitreihe handelt. Eine 0 ist der erste, eine 1 der zweite, eine 2 der dritte Wert usw. der Zeitreihe.

Die Checkbox *Random Chips* ermöglicht, dass aus den selektierten Chips die angegebene Anzahl zufällig ausgewählt wird.

Mit Hilfe der Schaltfläche *Count Chips*, wird die mit dem aktuellen SQL-Befehl ausgewählte Anzahl Chips angezeigt.

Die Gene werden mit einem SQL-Befehl gefiltert. Dieser wird entweder direkt eingegeben oder er wird mit Hilfe des Query Wizards generiert.

Der Query Wizard wird in Abbildung 9 dargestellt. Über die *Set ID* muss ein Datensatz ausgewählt werden. Zusätzlich kann bestimmt werden, dass keine 'missing values' vorkommen soll, oder dass die Gene in einem Datensatz eines anderen Datentypen (GE, MP, SL oder Gene) vorhanden sind. Es ist auch möglich, aus den so ausgewählten Genen, eine beliebige Anzahl Gene zufällig auszuwählen. Mit der Schaltfläche *Count*, wird die, mit dem aktuellen SQL-Befehl ausgewählte, Anzahl Gene angezeigt.

Damit der GE-Export korrekt funktioniert, muss sichergestellt sein, dass der Gen-Filter und der Chip-Filter auf den gleichen Datensatz zugreifen. Um das zu gewährleisten, muss die *Set ID* bei beiden Filter die Gleiche sein.

A.4.5 Flexibler Datenexport

Falls die bisher vorgestellten Dialoge die gewünschte Operation nicht erlauben, hilft wahrscheinlich der in Abbildung 10 gezeigte Dialog weiter. Mit ihm kann eine beliebige Suchanfrage in Form eines SQL-Befehls formuliert werden.

Das Resultat dieser Anfrage kann in eine Datei oder in eine Gen-Liste gespeichert werden.

Beim Dateiexport wird das Suchresultat Tabulator getrennt in die angegebene Datei geschrieben. Ist die Checkbox *Translation* ausgewählt, werden alle ausgewählte Gene des Suchresultates in die Referenzbezeichnung übersetzt. Dabei werden alle Feldbezeichnungen, die auf '*gene_id*', '*gene1_id*' und '*gene2_id*' enden, automatisch übersetzt.

Beim Export einer Gen-Liste in die Datenbank, werden alle im Suchresultat enthaltenen Gene in eine Gen-Liste zusammengefasst. Diese kann anschliessend für weitere Anfragen benutzt werden. Die Gene werden wie beim Dateiexport anhand ihrer Feldbezeichnung erkannt.

Der in Abbildung 10 enthaltene SQL-Befehl kreiert beispielsweise eine Liste von Referenzbezeichnungen eines Organismus, welche zu jedem Gen die Anzahl Synonyme angibt.

A.4.6 GO Wizard

Der GO-Wizard bietet verschiedene Möglichkeiten, Abfragen an die GO-Datenbank zu schicken. Der Dialog ist in Abbildung 11 abgebildet. Da die GO-Datenbank nicht direkt mit der Gen-Datenbank kompatibel ist, können keine direkten Abfragen zwischen den beiden Datenbanken gestellt werden.[11]

Bei den Abfragen mit dem GO Wizard wird eine Liste von Termen oder Genen über eine Datei eingelesen und das Resultat wiederum in eine Datei geschrieben. Enthält diese Datei eine Gen-Liste, kann sie anschliessend in die Gen-Datenbank eingelesen werden und erlaubt so indirekt die Verbindung der GO-Daten mit den anderen Datentypen.

Die Dateien müssen im folgenden Format vorliegen. Alle Terme beziehungsweise Gene sind Zeilenweise voneinander getrennt. Die Datei kann dabei im Windows Format $<CR><LF>$ oder im Unix Format $<LF>$ vorliegen.

Die möglichen Anfragen sind in Abschnitt 3.3 beschrieben. Wenn bei der Anfrage Terme in Gene übersetzt werden, muss die *ID* des Organismus angegeben werden. Eine Liste der vorhandenen Organismen ist in der GO-Datenbank in der Tabelle '*species*' zu finden.

[11] Mit Hilfe eines ziemlich komplexen SQL-Befehls wäre dies möglich. Besser wird eine solche Anfrage mit einem Zwischenschritt realisiert. Mit dem GO-Wizard wird eine Gen-Liste erstellt, die dann anschliessend in die Gen-DB eingelesen wird.

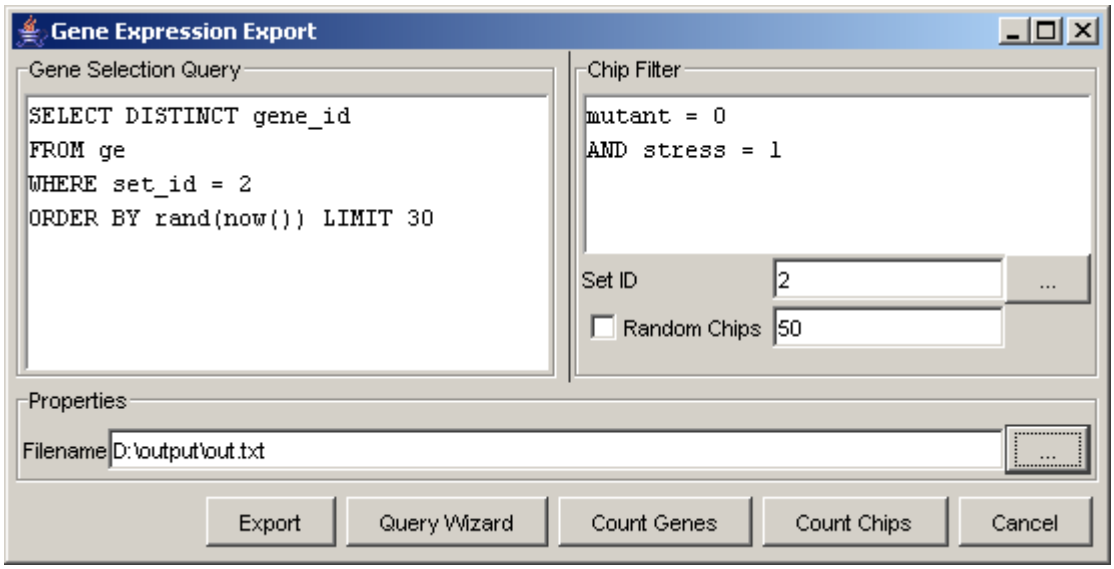

Abbildung 8: Dieser Dialog erlaubt das Exportieren der Genexpressionsdaten. Die Gene können anhand eines SQL-Statements selektiert werden. Die Chips werden anhand ihrer Zusatzinformationen ausgewählt. So ist es möglich eine Untermenge eines Satzes von Genexpressionsdaten zu exportieren.

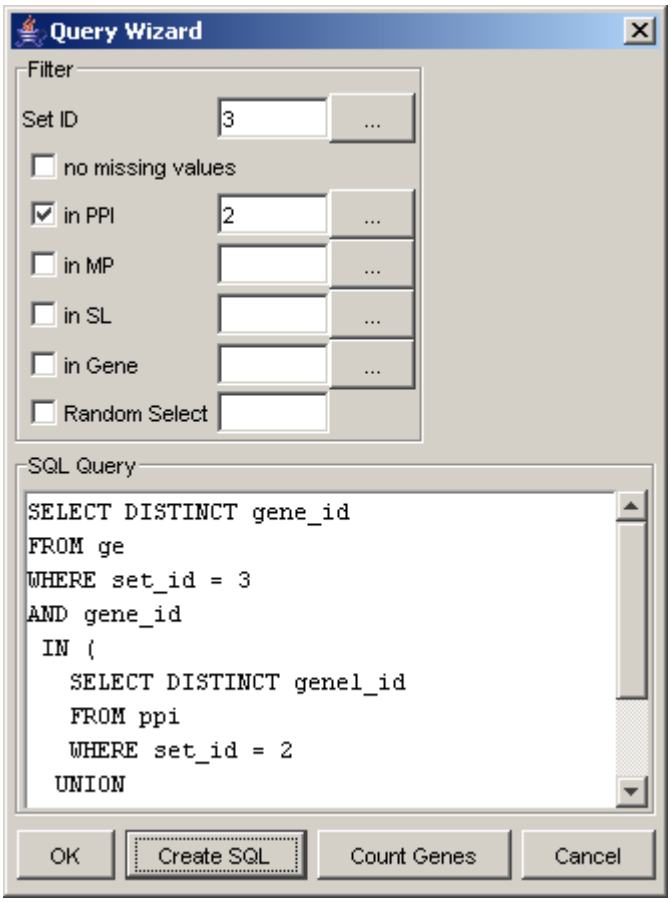

Abbildung 9: Der Query Wizard erlaubt das einfache Erstellen eine Gen-Filters für die Genexpressionsdaten.

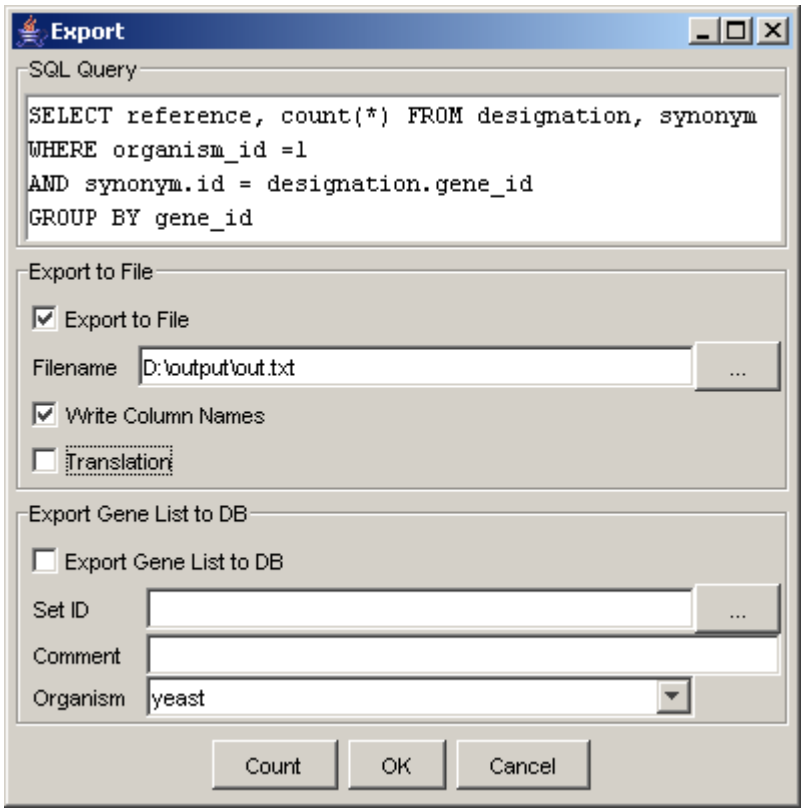

Abbildung 10: Mit diesem Dialog ist es möglich allgemeine Abfragen in Form eines SQL-Befehls zu formulieren.

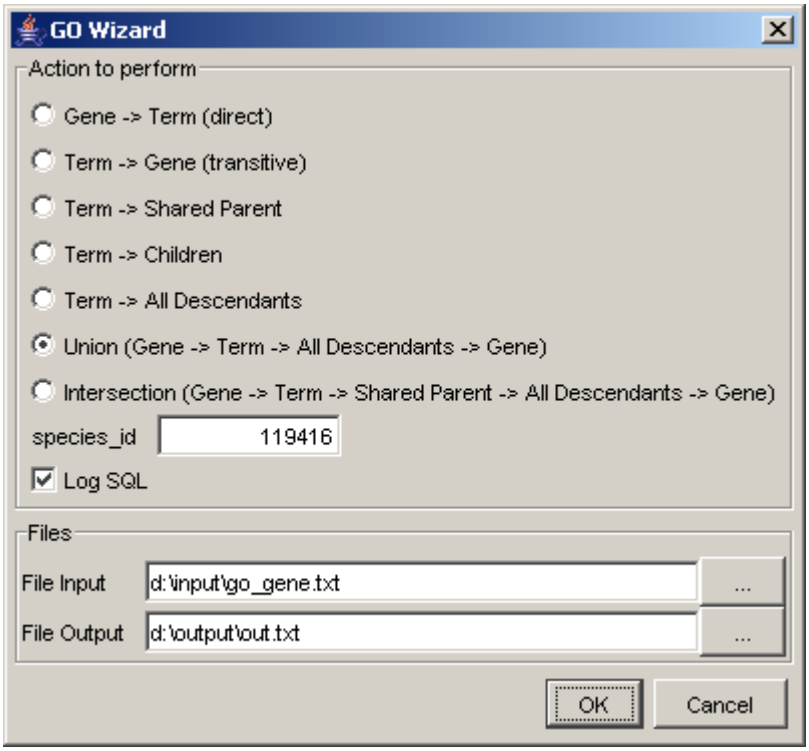

Abbildung 11: Der GO-Wizard erlaubt alle möglichen Anfragen an die Gene Ontology Datenbank.

B SQL-Suchanfragen

Die Selektion von Daten geschieht ausschliesslich über SQL-Befehle. Nachfolgend sind einige SQL-Befehle gezeigt, welche in geeigneter Kombination die meisten benötigten Anfragen erlaubt.

Folgende Punkte sollten beachtet werden:

- Normalerweise wird immer von einem Datensatz ausgegangen, für den die Filterung gilt. Siehe Listing 2.
- Mit der Kombination *rand(now()) limit n* werden n Elemente zufällig gewählt. Siehe Listing 2.
- Sollen bei GE-Daten keine 'missing values' vorkommen, kann dies gemäss Listing 2 realisiert werden.
- Sollen GE-Daten mit einem anderen Datentypen kombiniert werden, funktioniert das wie in Listing 3. Die beiden Gen-Felder der MP-Daten werden mit dem **UNION** Operator verbunden. Für die Selektion der Gene der MP-Daten werden Subqueries verwendet.
- Die einzelnen Abfragen können mittels booleschen Operatoren **AND**, **OR**, **NOT** und **XOR** beliebig verschachtelt werden. Siehe Listing 3.
- Sollen SL-, MP- oder PPI-Daten anhand einer Liste gefiltert werden, kann dies gemäss Listing 4 umgesetzt werden. Werden die beiden Anfragen mit einem **AND** verknüpft müssen beide Gene in der Gen-Liste enthalten sein bei, einer **OR** Verknüpfung nur eines der Gene.
- Für die Operationen der GO-Daten müssen oft Tabellen verknüpft werden. Das wird mit dem **JOIN** Befehl gemacht. Ein Beispiel dazu wird in Listing 5 gezeigt. Die Terme werden dort in einer Liste angegeben. Stattdessen könnte eine Subquery verwendet werden. Subqueries sind aber generell langsamer bei der Ausführung als solche Listen.
- Es können auch sehr viele verschiedenen Tabellen miteinander verknüpft werden. So kann prinzipiell durch das Schema der Datenbank 'gewandert' werden. Ein Beispiel dazu wird in Listing 6 gezeigt mit welchem der *Union* Befehl realisiert ist.

Eine vollständige Referenzliste aller Unterstützten Befehle ist unter

http://dev.mysql.com/doc

zu finden.

Listing 2: Es werden zufällig 50 Gene aus dem GE-Datensatz 3 selektiert. Die Gene sollen dabei keine 'missing values' besitzen.

```
SELECT DISTINCT gene_id
FROM ge
WHERE set_id = 3
AND gene_id NOT IN (
    SELECT DISTINCT gene_id
    FROM ge_missing
    WHERE set_id = 3)
ORDER BY rand(now()) LIMIT 50
```

Listing 3: Die Suchanfragen können beliebig komplex sein. Die einzelnen Anfragen werden mit den logischen Operatoren **AND**, **OR**, **NOT** und **XOR** verknüpft.

```
SELECT DISTINCT gene_id
FROM ge
WHERE set_id = 3
AND
(
    gene_id IN (
        SELECT DISTINCT gene1_id
        FROM mp
        WHERE set_id = 2
    UNION
        SELECT DISTINCT gene2_id
        FROM mp
        WHERE set_id = 2)
OR
    gene_id IN (
        SELECT DISTINCT gene_id
        FROM gene
        WHERE set_id = 3)
)
```

Listing 4: SL-Daten werden anhand einer Gen-Liste ausgewählt.

```
SELECT gene1_id, gene2_id, value
FROM sl
WHERE sl.set_id = 3
AND (
  sl.gene1_id
  IN (
  SELECT gene_id
  FROM gene
  WHERE gene.set_id = 2
  )
 OR
  sl.gene2_id
```

```
  IN (
  SELECT gene_id
  FROM gene
  WHERE gene.set_id = 2
 )
)
```

Listing 5: Es werden alle GO-Terme ausgegeben, zu welchen die aufgelisteten Gene assoziiert sind.

```
SELECT DISTINCT term.acc
    FROM gene_product
INNER JOIN association
    ON (gene_product.id = association.gene_product_id)
INNER JOIN term
    ON (term.id = association.term_id)
WHERE gene_product.symbol IN
    ('YOL022C', 'YOL007C', 'YNR004W')
```

Listing 6: Die Union Operation wird mit Hilfe einer Verkettung von **Join** Befehlen erreicht.

```
SELECT DISTINCT gp_out.symbol
FROM gene_product as gp_in
INNER JOIN association
    ON ( gp_in.id = association.gene_product_id )
INNER JOIN term as parent
    ON (parent.id = association.term_id
        AND parent.id = association.term_id)
INNER JOIN graph_path
    ON (parent.id = graph_path.term1_id)
INNER JOIN term as child
    ON (child.id  = graph_path.term2_id)
INNER JOIN association as asc_out
    ON (child.id = asc_out.term_id)
INNER JOIN gene_product as gp_out
    ON (asc_out.gene_product_id = gp_out.id)
WHERE gp_out.species_id = 119416
    AND graph_path.distance != 0
    AND gp_in.symbol IN
        ('YOL022C', 'YOL007C', 'YNR004W')
```

C Dateiformate

In diesem Abschnitt werden die Dateiformate der Import Dateien beschrieben. Alle Dateien werden als Tab-Delimited Text Dateien erwartet. Jedem Datum ist eine Zeile zugewiesen. Die Dateien können im Windows- oder im Unix- Format vorliegen.

C.1 Übersetzungsliste

Der Referenzname steht zuerst gefolgt von einem Tabulator und einer mit 'Spaces' getrennte Liste der Synonyme.

```
Gen1<tab>Synonym1 Synonym2 Synonym3 ... <LF>
Gen2<tab><LF>
Gen3<tab>Synonym1<LF>
...
```

C.2 Gen-Liste

Die einzelnen Gene sind durch Zeilenumbrüche voneinander getrennt.

```
Gen1<LF>
Gen2<LF>
Gen3<LF>
...
```

C.3 Term-Liste

Die einzelnen Terme sind durch Zeilenumbrüche voneinander getrennt.

```
Term1<LF>
Term2<LF>
Term3<LF>
...
```

C.4 PPI-Daten

Zu einem PPI-Datum gehören eine Kombination von Genen und der Experimenttyp in Form eines maximal drei Byte langen Zeichenkette.

```
Gen11<tab>Gen12<tab>Experimenttyp1<LF>
Gen21<tab>Gen22<tab>Experimenttyp2<LF>
Gen31<tab>Gen32<tab>Experimenttyp3<LF>
...
```

C.5 SL-Lethal

Das Format sieht praktisch gleich aus wie das der PPI-Daten nur das anstelle des Experimenttypes ein ganzzahliger Wert zu finden ist.

```
Gen11<tab>Gen12<tab>Wert1<LF>
Gen21<tab>Gen22<tab>Wert2<LF>
Gen31<tab>Gen32<tab>Wert3<LF>
...
```

C.6 MP-Daten

Dieser Datentyp besteht aus einer Liste von Gen-Paaren.

```
Gen11<tab>Gen12<LF>
Gen21<tab>Gen22<LF>
Gen31<tab>Gen32<LF>
...
```

C.7 GE-Daten

Die Genexpressionsdaten sind in einem Array gespeichert. Auf der ersten Zeile sind die Chips aufgelistet. Auf den Nachfolgenden Zeilen werden jeweils zuerst das Gen und anschliessend die Werte zu den zugehörigen Chips aufgelistet. 'Missing Values' werden durch zwei aufeinander folgende Tabulatoren gekennzeichnet.

```
ORF <tab>Chip1 <tab>Chip2 <tab>Chip3 <tab>Chip4  ... <LF>
Gen1<tab>Wert11<tab>Wert12<tab>Wert13<tab>Wert14 ... <LF>
Gen2<tab>Wert21<tab>Wert22<tab>      <tab>Wert24 ... <LF>
Gen3<tab>Wert31<tab>      <tab>Wert33<tab>Wert34 ... <LF>
...
```

D Aufgabenstellung

Task Description for the Term Project
Database System for High-throughput Genomics Data WS 2004/05

Student: Andreas Meier
Tutors: Amela Prelić, Stefan Bleuler
Professor: Eckart Zitzler

Background

Molecular cell biology has changed in the last few years, mostly due to a number of emerging technologies that allow to simultaneously measure different quantities witnessing the activities in the cell. Examples of such are the new microarray technologies, measuring the expression levels of the entire genome. By measuring the concentration of thousands of different m-RNA molecules it is possible to determine activity levels for all the respective genes. Further measurements are possible that monitor the activities in the cell at a post-transcriptional level (proteomics), metabolic level (fluxes), and others that provide structural informations, e.g., DNA sequences and 3D characterizations of protein structure. A variety of genomics data is being accumulated at an increasing rate, and an important issue is to store/manage the data in a way that will allow a systematic approach in combining the different data types for further analysis.

On the other hand, continuous research effort produces a wealth of information in form of annotations of previously uncharacterized genes, descriptions of interactions between their products (either as protein-to-protein, or interactions at enzymatic level, observable through their co-participation in chemical reactions). Beside these, a huge number of publications contain natural language description of processes in the cell experimentally verified or revealed through existing analysis methodology. Finding a way to store the available knowledge data in an organized, possibly machine-readable, form is a challenging task. Some of the most important questions in this context are how to keep these data up to date, error-free and consistent.

Starting Point

Currently, a large number of genomics databases exist. They offer information on one type of data only. Standards for storing the available information are rare and continuously changing. Furthermore, the gene nomenclature for genomes of different organisms is not well defined: several names denote the same entity, and a one-to-one correspondence to Uniprot, [Apweiler *et al.*, (2004)] or Swissprot, [Boeckmann *et al.*, (2003)] accession numbers is not available.

In the Reverse Engineering Project of Genetic Regulatory Networks (REP) project, people from different disciplines such as biology, computer science, statistics and electrical engineering are working together in order to devise new methods for analysis of gene expression data. Further genomic data types (noted below) are being used, for the validation purposes, but also as integrative parts of the methodology being developed. In a concrete scenario, we would like to analyze genes having complete functional annotations and being represented within the protein interactions data. It is important that an organization of the different data types exists, which allows to better manage the data and facilitates their access.

At the moment, different data types used are organized in a repository of gene expression data for different organisms, [REP, Stuart *et al.*, (2003)], selection of experimentally verified, publicly available protein-to-protein interactions [Xenarios *et al.*, (2002)], map representations of the main metabolic pathways and regulatory processes in the plant *Arabidopsis thaliana*, [Laule, 2004, Fürholz, 2004] and GO annotations, [The Gene Ontology Consortium, 2000].

Problem Definition

The main goal of this project is to develop a concept for management of diverse genomics data and implement an interface to ease access to datasets needed for analysis. Data considered will include high-throughput measurements and annotations representing prior knowledge available for studied organisms.
In order to reach this goal the project should proceed in the following steps:

1. Introduction - Setting the Background

   Relevant work (existing databases and data types) should be considered. The open problems should be defined and their potential solutions proposed and discussed.

2. Design - Concept Decisions

   In this step the requirement met by the database system should be determined in cooperation with the tutors based on the needs of the biologists. Among others the following issues should be addressed: What format should the input and output data take? How to best integrate different datasets in order to retrieve them by need? How to design an interface offering routine import/retrieve/store operations?

3. Database Design

   Different design variants should be considered, with focus on finding a solution that provides a general framework with modular and (easily) extensible organization. Performance issues are less relevant.

4. Formal Specification of the Database Model

   The set of functionalities offered by the database system should be defined and documented formally, their specifications should be provided (semantics and parameter list).

5. Implementation/Setting of the Database

   The database should be implemented (our choice was to use MySQL database and Java/Java Swing - GUI, to provide a user-friendly interface), offering functionalities set in the previous step. The implementation of the user interface should provide the set of routine queries established in the design step. Still, new kind of queries (not predicted) should be easy to integrate/add.

6. Testing

   Datasets should be partially stored in the system so to demonstrate the functionalities offered by the final implementation.

Organization

- Starting - Ending Date:

  2.11.2004 - 2.01.2005
  The final Report should be handed in by 12:00h, Friday 24.12.2004.

- Duration of the Work:

  7 weeks; full time.

- Time Schedule:

  You are asked to make a schedule for the project at the beginning. Continuously record the progress.

- Weekly Meetings:

  Two weekly meetings with the tutors will be held (Monday at 9 am, Wednesday at 5 pm) in order to discuss the current state of the work, potential difficulties as well as the future directions.

- Beginners Presentation:

  Approximately two to three weeks after the start you will shortly present the objectives of the work as well as some background on the topic. The presentation should be no longer than 5 minutes and consist of maximally two slides.

- Final Presentation:

  By the end of the project, you will present the results achieved during this term project. The presentation should not exceed 20 minutes (15 minutes talk + 5 minutes discussion).

- Documentation:

  At the end of the project you will have to hand in a written report. Together with the system implementation/software this report is the main outcome of the project. Document your work accurately. Besides the results describe also alternative solutions and justify your decisions. A second part of the report should consist of a user manual for the implemented system/software. Additionally make sure to comment your code extensively.

- Evaluation of the work

  The criteria for the final evaluation of the term project are described in the attached document "Notengebung bei Studien- und Diplomarbeiten".

References

[Apweiler *et al.*, (2004)] Apweiler R., Bairoch A., Wu C.H., Barker W.C., Boeckmann B., Ferro S., Gasteiger E., Huang H., Lopez R., Magrane M., Martin M.J., Natale D.A., O'Donovan C., Redaschi N., Yeh L.S. (2004) UniProt: the Universal Protein Knowledgebase. *Nucleic Acids Res.* 32:D115-D119.

[Boeckmann *et al.*, (2003)] Boeckmann B., Bairoch A., Apweiler R., Blatter M.-C., Estreicher A., Gasteiger E., Martin M.J., Michoud K., O'Donovan C., Phan I., Pilbout S., Schneider M., (2003) The Swiss-Prot Protein Knowledgebase and Its Supplement TrEMBL in 2003. *Nucleic Acids Res.* 31:365-370.

[Fürholz, 2004] Fürholz, A., (2004) Effects of PFT-mediated Prenylation on the Gene Network in *Arabidopsis thaliana*. *PhD thesis*, *ETH NO. 15675*.

[Laule, 2004] Laule O., (2004) Coordination of MVA and MEP Isoprenoid Pathways in *Arabidopsis thaliana*. *PhD thesis*, *ETH NO. 15674*.

[REP] Reverse Engineering Project at ETH Zurich, http://www.rep.ethz.ch/

[Stuart *et al.*, (2003)] Stuart, J. M., Segal, E., Koller, D., Kim, S.K., (2003) A Gene-Coexpression Network for Global Discovery of Conserved Genetic Modules, *Science*, 302:249-255.

[The Gene Ontology Consortium, 2000] The Gene Ontology Consortium, (2000) Gene Ontology: Tool for the Unification of Biology, *Nature Genetics*, **25**, 93-103.

[Xenarios *et al.*, (2002)] Xenarios, I., Salwinski, L., Duan, X.J., Higney, P., Kim, S.M., Eisenberg, D., (2002) DIP, the Database of Interacting Proteins: a Research Tool for Studying Cellular Networks of Protein Interactions, *Nucleic Acids Res.* , **30(1)**, 303-5.