RUPRECHT- KARLS-
UNIVERSITÄT
HEIDELBERG

HOCHSCHULE HEILBRONN
TECHNIK WIRTSCHAFT INFORMATIK

Studiengang Medizinische Informatik

Entwicklung eines caBIG-Dienstes zur Konvertierung von Genbezeichnungen zwischen den Benennungssystematiken verschiedener Gendatenbanken

DIPLOMARBEIT

vorgelegt von

ELISABETH FORSTER

Betreuer

Prof. Dr. Petra Knaup-Gregori

Cobetreuer

Priv.Doz. Dr. Niels Grabe

MÄRZ 2011

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Diplomarbeit selbständig und ohne unzulässige fremde Hilfe angefertigt habe. Die verwendeten Literaturquellen sind im Literaturverzeichnis vollständig angegeben.

Heilbronn, den

Elisabeth Forster

Zusammenfassung

Im Rahmen der interdisziplinären Leberkrebsforschung des ‚SFB/TRR 77' ‚Leberkrebs von der molekularen Pathogenese zur zielgerichteten Therapie' fallen Genomdaten an, welche in unterschiedlichen Formaten gespeichert werden. Der ‚SFB/TRR 77' wird technisch durch die integrierte Informationsplattform ‚pelican' (platform enhancing livercancer networked research) unterstützt. ‚pelican' erlaubt Forschern ihre (Genom-)Daten zentral zu speichern und Daten von anderen Projekten einzusehen. Die Informationsplattform ist nach dem Prinzip der modernen serviceorientierten Architektur (SOA) aufgebaut, nach welcher Dienste über einheitliche Schnittstellen an unterschiedlichen Orten spezifische (Teil-)Aufgaben erfüllen, um so größere Prozesse zu realisieren.
Damit die Genomdaten für umfassende Analysen einheitlich zur Verfügung stehen, bedarf es eines Genkonvertierungs-Dienstes zur Abbildung von Genbezeichnungen aufeinander, der in die SOA von ‚pelican' integriert werden kann.
Verschiedene Gendatenbanken und biomedizinische Standards wurden auf ihre Eignung untersucht, Gene eindeutig zu identifizieren. Auch die Tools des ‚cancer Biomedical Informatics Grid' (caBIG) werden auf Methoden und Werkzeuge zur Unterstützung einer eindeutigen Genidentifikation analysiert.
Aus den verschiedenen Genidentifikationsmöglichkeiten wird eine optimale Methode ausgewählt, mit der im Genkonvertierungs-Dienst gearbeitet werden kann. Diese ist das ‚Gensymbol', da es Gendatenbank übergreifend ist und von dem Human Genome Organisation (HUGO) Gene Nomenclature Committee (HGNC) standardisiert wird. Die Daten aus der HGNC-Datenbank werden in eine eigene MySQL-Datenbank transferiert. Auf Basis dieser Datenbank wird ein Webservice entwickelt, der dann in die SOA der Informationsplattform eingebunden werden kann. Der erstellte Webservice steht zur Abfrage von Genomdaten bereit. Mittels entsprechenden Tests wird die Funktionalität des neuen Services validiert.

Abstract

In the context of the ‚SFB/TRR 77' ‚Liver cancer from molecular pathogenesis to targeted therapies' which is about interdisciplinary liver cancer research genomic data accumulate and are stored in various data formats. The ‚SFB/TRR 77' is technically supported by the integrated information platform ‚pelican' (platform enhancing livercancer networked research). ‚pelican' permits researchers to store their (genomic) data centrally and to query data from other project resources. The information platform follows the principle of the modern service oriented architecture (SOA). One key feature of this architecture is that different services at various places offer their functionality via unified interfaces to accomplish specific tasks. Thus, these services can be combined, solve different problems and implement bigger processes.
To unify the genomic data and to make them available for comprehensive analyses a gene conversion service is required that maps gene identifier and can be integrated in the SOA of ‚pelican'.
Various gene data bases and biomedical standards have been analyzed regarding their suitability to unambiguously identify genes. In addition it is analysed if existing tools of the ‚cancer Biomedical Informatics Grid' (caBIG) are usable for gene identifications or at least can be applied to support clear gene identification.
From the various options of gene identifications an optimal method is selected and used for the gene conversion service. It is the ‚gene symbol' since it is independent from a gene database and it is standardized within the Human Genome Organisation (HUGO) Gene Nomenclature Committee (HGNC). The data from the HGNC database are transferred to an individual MySQL database. A web service is developed using this database and is integrated in the SOA of the information platform. This new web service is ready for the retrieval of genomic data. Its functionality is validated by adequate tests.

Inhaltsverzeichnis

1 Einleitung

1.1 Gegenstand und Motivation

Krebs zählt zu den häufigsten Todesursachen in der Welt. Zu den Krebsarten mit den meisten Todesfällen pro Jahr zählen Lungen-, Brust-, Darm- und Leberkrebs. Man erwartet, dass die Anzahl der Todesfälle durch Krebs bis 2030 von acht Millionen pro Jahr auf circa zwölf Millionen ansteigen wird.[1]
In den Vereinigten Staaten ist das hepatozelluläre Karzinom (engl. hepatocellular carcinoma (HCC)) mit einer jährlichen Steigung der Anzahl der Neuerkrankungen um 1,75% der Krebs mit der stärksten Inzidenzsteigerung von allen. Auch in Deutschland zeichnet sich ein ähnlicher Trend ab. Bisher gibt es für das hepatozelluläre Karzinom kaum erfolgversprechende Therapiemöglichkeiten, es sterben weltweit rund 400.000 Menschen jährlich an dieser Art von Tumor. Von einem besseren Verständnis der molekularen Basis von HCC verspricht man sich bessere Vorbeugung und Früherkennung von HCC.
Diese Problematik hat zur Einrichtung des Sonderforschungsbereichs (Transregio) ‚SFB/TRR 77' ‚Leberkrebs von der molekularen Pathogenese zur zielgerichteten Therapie' geführt, der von der Deutschen Forschungsgemeinde (DFG) gefördert wird. Kooperationspartner im ‚SFB/TRR 77' sind die Ruprecht-Karls-Universität Heidelberg, das Deutsche Krebsforschungszentrum in Heidelberg, die Medizinische Hochschule Hannover und das Helmholtz-Zentrum für Infektionsforschung in Braunschweig. Das ‚SFB/TRR 77' -Projekt besteht aus 22 verschiedenen Teilprojekten, die unterschiedliche Aspekte der Leberkrebserkrankung HCC untersuchen. Der ‚SFB/TRR 77' hat sich drei Hauptziele gesetzt:

- Generelles Verständnis für spezifische Mechanismen von chronischen Lebererkrankungen (speziell Vireninfektionen und entzündungsvermittelte Prozesse), die zu Leberkrebs führen können und die eventuell neue präventive Gegenmaßnahmen hervorbringen
  (**Research Area A**)
- Identifizierung und Funktionsweise von Schlüsselmolekülen, die Leberkrebs begünstigen bzw. an ihm (in-)direkt beteiligt sind und damit als Marker für tumorrelevante Funktionen dienen. Mit deren Hilfe könnten zum einen zukünftige Behandlungsmaßnahmen durchgeführt und zum anderen möglicherweise der Verlauf von Leberkrebs besser erklärt werden.
  (**Research Area B**)
- Erforschung von spezifischen tumor-relevanten Mechanismen, um in der bereits laufenden Krebsforschung analoge Behandlungsmethoden zu finden bzw. diese so abzuwandeln, dass sie in der Leberkrebsforschung angewendet werden können.
  (**Research Area C**)

Um die Vielzahl der in den 22 wissenschaftlichen Projekten anfallenden Daten für alle Projekte verfügbar machen zu können, umfasst der ‚SFB/TRR 77' das Querschnittsprojekt Z2 'Integrierte Informationsplattform und projektübergreifende biostatistische Auswertungen'. Aufgaben sind unter anderem die Bereitstellung einer integrierten Informationsplattform, die Beratung bezüglich biostatistischer Methoden sowie die Durchführung projektübergreifender Analysen. Durch die Verknüpfung der Daten aus den Teilprojekten, die HCC aus verschiedenen Blickwinkeln untersuchen, erhofft man sich neue Erkenntnisse über die Vorgänge in den Krebszellen und ein besseres Verständnis von der Entstehung und des Verlaufs der Erkrankung.
Verschiedene Forschungsprojekte aus unterschiedlichen Forschungseinrichtungen bringen jedoch durch Anwendung verschiedener Verfahren und Methoden unterschiedliche Datenformate und im Falle des ‚SFB/TRR 77' unterschiedliche Arten der Genidentifikation mit sich. Des Weiteren werden in den Forschungsprojekten für die Analyse von molekularbiologischem Material Microarrays

[1] http://www.who.int/cancer/en/index.html (letzter Zugriff 19. Januar 2011)

von unterschiedlichen Herstellern verwendet. Diese liefern ihre Ergebnisse in verschiedenen Repräsentationsformen. Um die gesammelten Daten jedoch für umfassende Analysen oder fachübergreifende Forschungsfragen verwenden zu können, müssen sie auf der integrierten Informationsplattform in einer einheitlichen Form und Genidentifikation gespeichert werden. Bisher existiert aber kein Dienst, der automatisch die Formate und Genidentifikationen aufeinander abbilden kann. Die integrierte Informationsplattform des ‚SFB/TRR 77' soll als eine Service-Orientierte Architektur (SOA) auf Basis des cancer Biomedical Informatics Grid (caBIG) realisiert werden. Um die Nachhaltigkeit der wissenschaftlichen Daten zu gewährleisten, muss ein Dienst erstellt werden, der die Eindeutigkeit der Genidentifikation garantiert. Damit können die Daten aus den verschiedenen Projekten aufeinander abgebildet und projektübergreifende Analysen und Auswertungen ermöglicht werden.

1.2 Problemstellung der Arbeit

Problem 1: Die wissenschaftlichen Daten aus den 22 Projekten des ‚SFB/TRR 77' weisen unterschiedliche Genidentifikationen auf.

Problem 2: Es existiert bisher kein Werkzeug für die integrierte Informationsplattform des ‚SFB/TRR 77', in dem unterschiedliche Genidentifikationen aus unterschiedlichen Datendiensten eindeutig aufeinander abgebildet werden können.

1.3 Zielsetzung

Ziel 1: Analyse von Standards zur eindeutigen Identifikation von Genen

Ziel 2: Analyse von caBIG auf Methoden und Werkzeuge zur Unterstützung der eindeutigen Genidentifikation

Ziel 3: Entwicklung eines Mapping-Dienstes für Gen-IDs zur Einbindung in caBIG

1.4 Frage- und Aufgabenstellung

Fragensammlung der Arbeit:

Fragen zu Ziel 1:

- Welche Methoden der Identifikation von Genen werden in Gendatenbanken genutzt?
- Gibt es eine Referenz-Gendatenbank?
- Welche Methoden der Identifikation von Genen werden in biomedizinischen Standards genutzt?

Fragen zu Ziel 2:

- Welche Möglichkeiten der eindeutigen Genidentifikation bietet caBIG?
- Welche Anforderungen müssen erfüllt sein, dass caBIG auf externe Dienste zugreifen kann?

Fragen und Aufgaben zu Ziel 3:

- Wie kann ein Gen für die IT-Plattform des ‚SFB/TRR 77' eindeutig identifiziert werden?
- Welche Anforderungen muss die Informationsplattform diesbezüglich erfüllen?
- Entwicklung des Dienstes im Rahmen der SOA des Z2-Projektes.
- Implementierung, Dokumentation, Integration und Validierung des Dienstes.

2 Grundlagen

2.1 Die Krankheit Krebs

Als Krebs wird in der Medizin ein bösartiger Tumor bezeichnet. Er ist nach Herz-Kreislauferkrankungen die häufigste Todesursache in Deutschland (siehe Abbildung 1). Allein in Deutschland erkranken jährlich rund 450.000 Menschen neu an Krebs, 216.000 Menschen im Jahr sterben daran.[1] Es wird geschätzt, dass die Inzidenz aufgrund der demographischen Entwicklung bis zum Jahr 2050 um 30 % steigt, da Krebs eine Erkrankung ist, die besonders ältere Menschen trifft. Krebs entsteht dadurch, dass sich bestimmte Abschnitte der Gene in einer Zelle verändern, diese Veränderungen nicht mehr repariert werden und sich damit die Erbinformation verfälscht. Solche Krebszellen unterliegen nicht mehr der Kontrolle des Zellzyklus und wachsen deshalb unkontrolliert zum Nachteil des umliegenden Gewebes. Darüber hinaus besiedeln sie auch andere, fremde Gewebszellen.

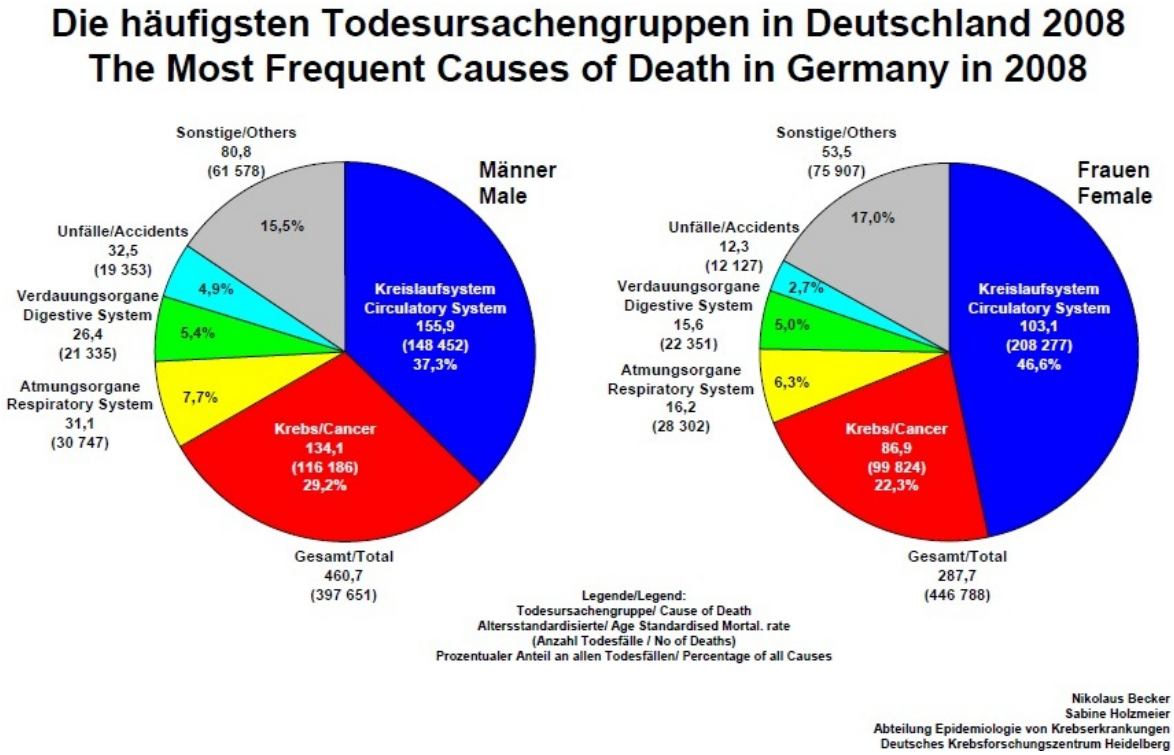


Abbildung 1:
Eine Übersicht der häufigsten Todesursachen in Deutschland 2008
Quelle: http://www.dkfz.de/de/krebsatlas

Es gibt mehrere Risikofaktoren, die die Krebsentstehung fördern können. Hierzu zählen unter anderem UV-Strahlen, Tabakrauch, Chemikalien, erhöhter Alkoholgenuss und eine ungesunde Lebensweise. Prinzipiell kann jedes Organ im menschlichen Körper von Krebs betroffen sein. Die häufigsten Krebsarten in Deutschland sind Lungen-, Darm-, Brust-, Prostata-, Haut- und Gebärmutterhalskrebs (Huch 2003).
Diese Arbeit entsteht in einem Projekt, das sich mit der Erforschung des hepatozellulären Karzinoms, auch als Leberkrebs oder HCC bekannt, beschäftigt. Leberkrebs kommt in Deutschland im Vergleich zu Brust- oder Prostatakrebs seltener vor, ist jedoch weltweit eine der häufigsten Krebsarten. Zudem ist es eine der Krebsarten mit der stärksten Inzidenzrate. Ursachen, die zu Leberkrebs führen können, sind unter anderem die Infektionskrankheiten Hepatitis B und C. Leberkrebs verur-

[1] Statistiken der Gesundheitsberichterstattung des Bundes (http://www.gbe-bund.de)

sacht anfangs nur sehr wenige Beschwerden, wodurch er meist erst in Spätstadien erkannt wird, wenn sich in andere Organen bereits Metastasen gebildet haben. Heilungschancen sind zu diesem Zeitpunkt nur noch mit einer sehr geringen Wahrscheinlichkeit gegeben (Hussain 2010).

2.2 Das Projekt ‚SFB/TRR 77'

Das hepatozelluläre Karzinom ist eine der häufigsten und bösartigsten Krebsarten weltweit. Gleichzeitig sind die Therapiemöglichkeiten von HCC bisher nur sehr eingeschränkt. Aber HCC ist ein relevantes Beispiel für Krebs, der durch Viren oder entzündungsvermittelt verursacht wird. Damit ist er ein ideales Modell für die Tumorforschung. Die vier Kooperationspartner, die Ruprecht-Karls-Universität Heidelberg, das Deutsche Krebsforschungs-Zentrum (DKFZ) in Heidelberg, die Medizinische Hochschule Hannover und das Helmholtz-Zentrum für Infektionsforschung in Braunschweig haben das ‚SFB/TRR 77' Projekt eingerichtet mit dem Hauptziel ein fundiertes Verständnis von der molekularen Basis von Leberkrebs zu erlangen und damit die Therapiemöglichkeiten und Früherkennung von Leberkrebs zu verbessern.
Um dieses Ziel zu erreichen, hat der ‚SFB/TRR 77' das Hauptziel in drei Forschungsschwerpunkte unterteilt, die zusammen 22 Teilprojekte umfassen.
Forschungsschwerpunkt A beschäftigt sich mit dem Verständnis von generellen und spezifischen Mechanismen chronischer Lebererkrankungen, die die Entwicklung von HCC auslösen oder die Leber dafür anfällig machen. Die Erkenntnisse über diese Mechanismen bieten Aussicht für präventive Maßnahmen und Interventionen an. Dabei werden vor allem Lebererkrankungen betrachtet, die durch Virusinfektionen, z.B. chronische Hepatitis, oder chronische Entzündungen verursacht wurden.
Ziel von Forschungsschwerpunkt B ist die Definition neuartiger protumorigener Mechanismen durch zwei verschiedene Wege. Der erste Weg identifiziert Schlüsselmoleküle, die Leberkrebs begünstigen. Sie dienen als Marker für tumor-relevante Funktionen. Der zweite Weg thematisiert Grundmechanismen wie zum Beispiel die Zellteilung oder die Genregulation.
Der dritte Forschungsschwerpunkt C erforscht Mechanismen für zukünftige Therapiemöglichkeiten. Ansätze dazu sind: frühzeitige Prognose des Krankheitsverlaufs, Tumormarker, Chaperone- und Tumorzellen-Targeting.
Neben den drei Forschungsschwerpunkten gibt es noch einen zentralen Schwerpunkt Z. Dieser ist in die zwei Bereiche Z1 und Z2 unterteilt. Z1 umfasst alle Aspekte der Verwaltung des ‚SFB/TRR 77'. Darüber hinaus kümmert er sich um alle Interaktionen zwischen den Projekten. Z2 entwickelt eine integrierte Informationsplattform (Z2-IT) für die biomedizinischen Daten aus den verschiedenen Forschungsschwerpunkten. Außerdem hat Z2 die Aufgabe, die Projektgruppen mit biostatistischen Tools und Beratung sowie bei der Durchführung von umfassenden Datenanalysen zu unterstützen (Z2-Biostatistics)[1].

Diese Arbeit befindet sich im Schwerpunkt Z2-IT. Der Schwerpunkt stellt eine integrierte Informationsplattform namens ‚pelican' (**p**latform **e**nhancing **li**ver**ca**ncer **n**etworked research) zur Verfügung. Sie erlaubt Forschern ihre Daten zentral zu speichern und Daten von anderen Projekten einzusehen. Jedes Projekt hat die Möglichkeit, den Zugriff auf seine Daten zu verwalten und zu bestimmen, wer Zugang zu den Daten hat. Der Zugriff auf Daten wird protokolliert. Die Plattform soll Projektversuchsdaten wie Gewebe-, Molekular-, genetische und klinische Daten speichern, um sie langfristig auch in anderen Projekten und für übergreifende Analysen verwenden zu können.
Die Aufgabe dieser Arbeit ist es, ein einheitliches Identifikationsschema im Bereich der genetischen Daten zu erarbeiten, das eine projekt-übergreifende Analyse und langfristige Nutzung der Daten erlaubt. Darüber hinaus bietet die Informationsplattform an, Datenquellen zu verknüpfen oder Informationen zu Genen zu bekommen.
Die Forscher aus den SFB-Projekten können über das Portal ‚pelican' mit einem Standard-Webbrowser auf die Daten und Dienste zugreifen. Der Vorteil des Webbrowser ist, dass keine In-

[1] http://www.livercancer.de (letzter Zugriff 19.Januar 2011)

stallation von Nöten ist. Alle Dienste liegen im caGrid, einem Tool von caBIG (siehe Kapitel 2.3). caGrid unterstützt den Aufbau einer Service-orientierten Architektur. Neben dem aCGH- (array comparative genomic hybridization) und dem Methylierungsdienst gibt es den Genkonvertierungs-Dienst. Diesen Konvertierungs-Dienst im Bereich der genetischen Daten bereitzustellen ist Aufgabe dieser Arbeit.

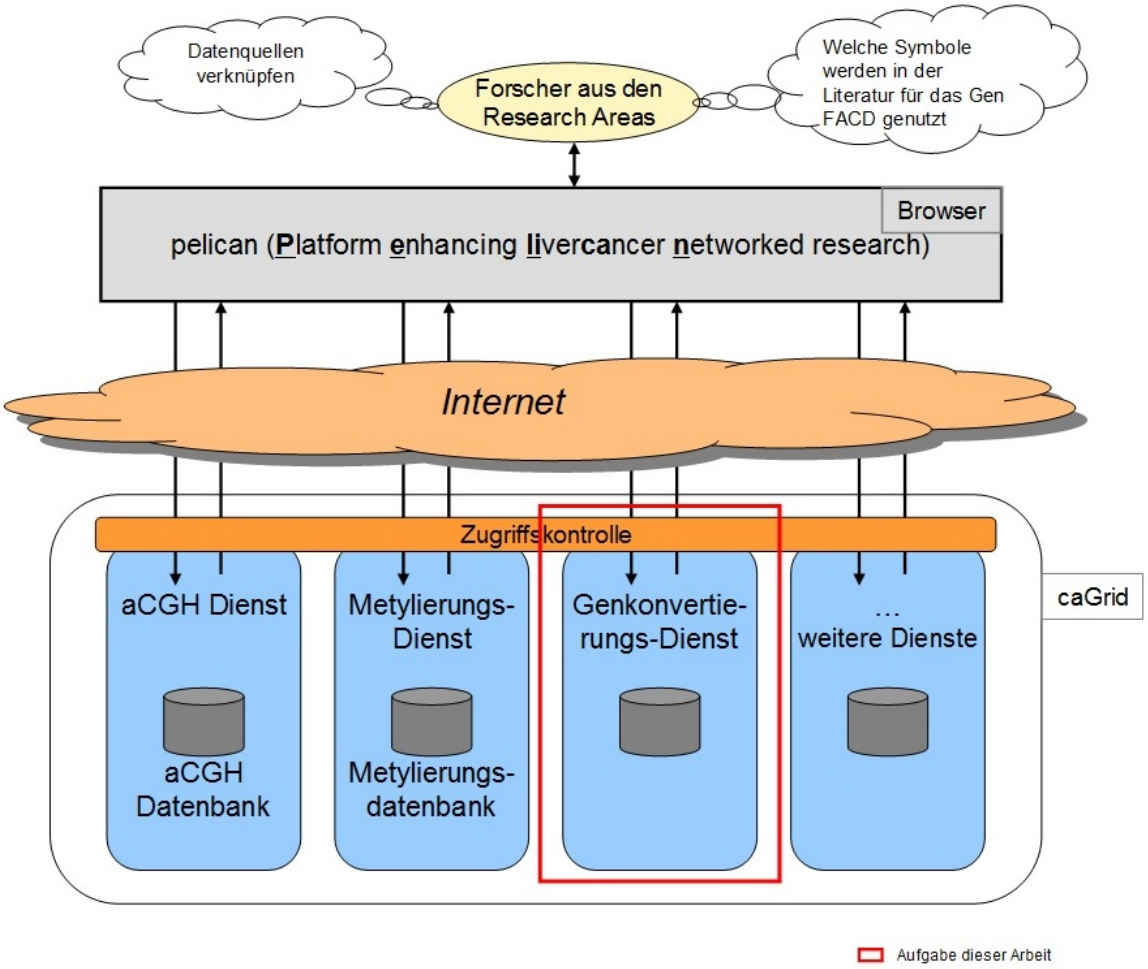


Abbildung 2:
Schematische Darstellung der Informationsplattform ‚pelican'

In der Forschung unterliegen die Namen und Identifikationsverfahren eines Gens einem häufigen Wechsel, zum Beispiel durch den Fortschritt der Forschung. Zweck des Dienstes ist es, über eine Bezeichnung eines Gens, zum Beispiel über ein Gensymbol, die aktuelle und frühere Standardbezeichnung für dieses Gen zu bekommen.
Beispiel (siehe Abbildung 3): Ein Forscher möchte aktuelle Daten und frühere Namen für seine Literaturrecherche zu einem Gen finden. Er hat aber nur eine alte Bezeichnung des Gens. Er gibt über den Webbrowser sein Gensymbol ein: ‚FACD' (Fanconi anemia, complementation group D1). Das Symbol wird an den Genkonvertierungs-Dienst weitergeleitet. Dieser gibt dann das aktuelle Symbol BRCA2 (breast cancer 2, early onset) zurück, sowie weitere Informationen zu dem Gen. Zu diesen Informationen gehören die ID-Bezeichnungen von den verschiedenen Gendatenbanken (siehe Kapitel 2.5), Vorgänger-Symbole und andere Bezeichnungen für das Gen.

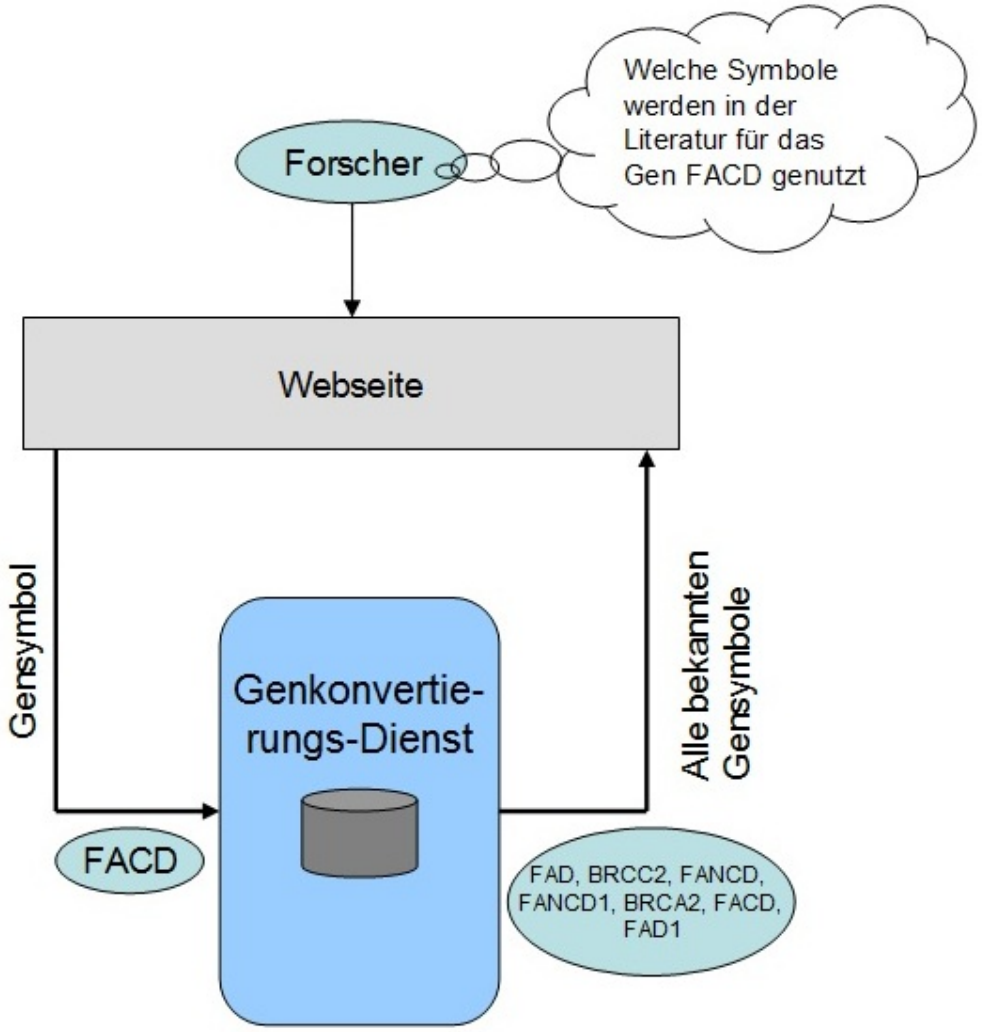


Abbildung 3:
Schematische Abbildung des Genkonvertierungs-Diensts, der in dieser Arbeit erstellt werden soll.

2.3 caBIG

Im Jahr 2004 hat das amerikanische National Cancer Institute (NCI) die Cancer Biomedical Informatics Grid (caBIG)-Initiative gegründet. Das Ziel von caBIG ist es, durch die Verbindung von Forschern und Ärzten mit ihren Patienten über ein Informationsnetzwerk Forschungsergebnisse zu beschleunigen und die Diagnosen zu verbessern. Über dieses gemeinsame Netzwerk haben die Forscher und Ärzte die Möglichkeit, Daten und Wissen zu teilen und die Zusammenarbeit zu vereinfachen. Dadurch wird die Entdeckung von neuen Methoden für die Erkennung, Diagnose, Therapie und Prävention von Krebs beschleunigt. caBIG wurde vor allem für die Krebsforschung implementiert. Die Dienste von caBIG sind durch die service-orientierte Architektur auch über die Krebsforschung hinaus verwendbar.
Ein Problem beim Austausch von Forschungsdaten zwischen verschiedenen Institutionen ist die unterschiedliche Darstellung der Daten. Außerdem erschweren verschiedene Terminologien es Forschern, Daten von anderen Einrichtungen zu verstehen, auch wenn beide Einrichtungen dieselbe Krankheit untersuchen. Aus diesem Problem entstand ein weiteres Ziel: Die Entwicklung von Standardregeln und einer gemeinsamen Sprache, um Informationen leichter zu teilen.

Aus diesem Grund hat es sich caBIG zur Aufgabe gemacht Tools für die Sammlung, Analyse, Integration und Verteilung von Informationen für die Krebsforschung und der Krebsbehandlung bereitzustellen und anzupassen. caBIG verpflichtet sich folgenden Prinzipien[1]:

- **Open Access**: Damit ein umfassender Datenaustausch und eine umfassende Datensammlung gewährleistet werden, sind die Ressourcen von caBIG frei zugänglich.
- **Open Development**: Tools von caBIG werden über einen offenen und mitbestimmenden Prozess entwickelt. Wenn die Möglichkeit besteht, setzt caBIG eher bereits vorhandene Ressourcen ein, als neue Tools aufzubauen.
- **Open Source**: Open Source bedeutet, dass der Quellcode frei verfügbar ist und für eigene Zwecke verändert oder erweitert werden darf. caBIG stellt die eigene Software als Open-Source-Lizenz zur Verfügung, um die Wiederverwendung von existierendem Code zu unterstützen.
- **Föderation**: Die Software von caBIG ermöglicht Organisationen Daten mit anderen Organisationen zu teilen, sowie Daten, die andere beitragen, zu verwenden. Die Daten von mehreren Standorten können verbunden werden und im Grid als ein integrierter Datensatz erscheinen, während jede Institution die Kontrolle über ihre eigenen Ressourcen und Daten behält (Buetow & Eschenbach 2006).

2.4 Service-orientierte Architektur

caBIG basiert auf dem Konzept einer service-orientierten Architektur (SOA). Eine service-orientierte Architektur ist ein Architekturmuster, das mehrere Dienste miteinander verbindet. Diese sind jeweils für die Lösung einzelner Aufgaben zuständig und stellen über Schnittstellen ihre Funktionen den anderen Diensten zur Verfügung. Die Lösung einer Fragestellung ergibt sich aus einer Reihung von Diensten.
Service-orientierte Architekturen haben folgende wesentliche Merkmale:

- **Lose Kopplung**: Die Dienste einer SOA sind voneinander unabhängig. Jeder Dienst wird nur bei Bedarf gesucht und in die Aufgabenbewältigung eingebunden. Damit ein Dienst leichter gefunden werden kann, gibt es ein Dienstverzeichnis, in dem alle zur Verfügung stehenden Dienste registriert sind. Neue Dienste können jederzeit hinzugefügt und registriert werden.
- **Leichte Wartbarkeit**: Jeder Dienst kann für jede anderen Aufgabenstellungen wiederverwendet werden. Neue Dienste können leicht in die SOA integriert werden. Auch Software, die vor einer SOA erstellt wurde, kann über eine Schnittstelle verfügbar gemacht werden**.**
- **Offenheit**: Die SOA basiert nicht auf einer festgelegten Technologie. Die Implementierung der Dienste ist nach außen hin verborgen, sie können an unterschiedlichen Orten liegen, von verschiedenen Organisationen entwickelt und in verschiedenen Programmiersprachen geschrieben sein.

Jeder Dienst publiziert seine Schnittstelle. Sie beschreibt, welche Eingaben in welchem Format der Dienst benötigt und in welcher Form er das Ergebnis liefert.
In einer SOA spielen drei Rollen zusammen: Dienstanbieter, Dienstnutzer und Verzeichnisdienst. Abbildung 4 zeigt das Zusammenspiel dieser Rollen: Ein Dienstanbieter veröffentlicht seinen Dienst. Dieser wird dann mit einer Beschreibung der Funktionalität im Verzeichnisdienst registriert. Ein Dienstnutzer kann im Verzeichnisdienst nach Diensten und ihrer Funktionalität suchen. Damit er den Dienst einbinden kann, benötigt er die Schnittstellenbeschreibung des Dienstes. Zusätzlich können Vorraussetzungen zur Nutzung des Dienstes, wie zum Beispiel eine Authentifizierung, gefordert sein. Erst wenn alle Voraussetzungen erfüllt sind, kann der Dienst vom Nutzer verwendet werden.

[1] http://cabig.cancer.gov/caBIGstory/principles/ (letzter Zugriff 19.Januar 2011)

Ein Dienstnutzer kann hier sowohl ein Nutzer sein, der einen Dienst zum Beispiel über eine Webseite aufruft, als auch ein Dienst, der auf einen anderen Dienst zugreift (Melzer 2008) (Finger 2009).

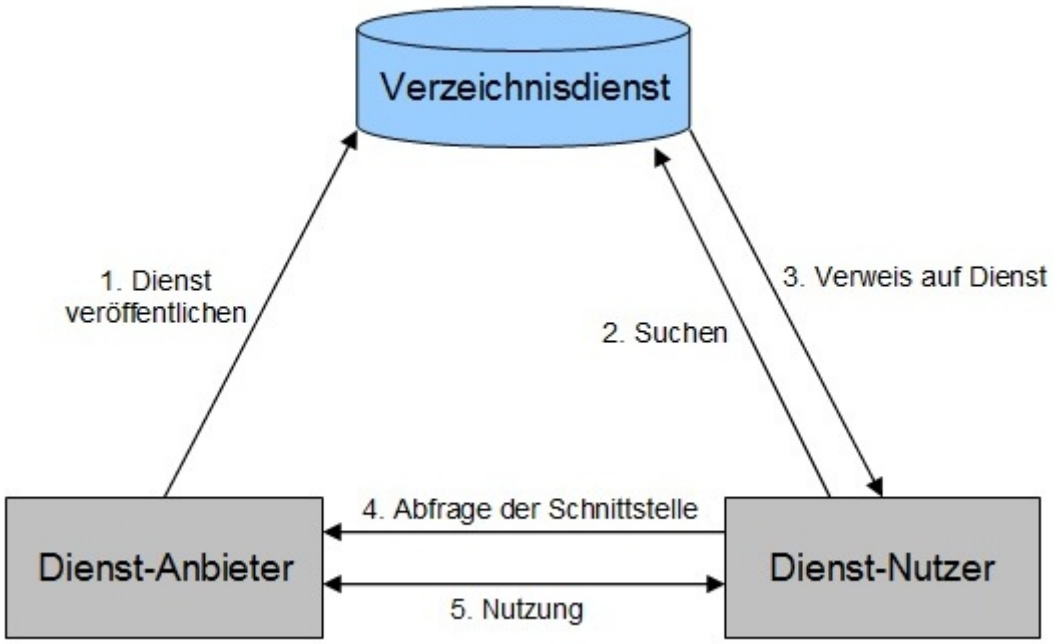


Abbildung 4:
Diese Abbildung zeigt das Zusammenspiel zwischen dem Dienstanbieter, dem Dienstnutzer und dem Verzeichnisdienst in einer SOA.
In Anlehnung an: (Melzer 2008)

2.5 Genetische Grundlagen

Die Desoxyribonukleinsäure (DNA) ist der Träger der genetischen Erbinformationen eines Lebewesens. Sie liegt in Form von Chromosomen vor. Ein Mensch besitzt zwei Kopien von jedem Chromosom: eine von der Mutter, eine vom Vater. Insgesamt hat ein Mensch 46 Chromosomen, auf denen ungefähr 27.500 Gene[1] verteilt sind.
Die DNA ist aus den organischen Basen Adenin(A), Guanin (G), Cytosin (C) und Thymin(T) aufgebaut. Diese Basen bilden zusammen mit dem Zucker Desoxyribose und einem Phosphatrest ein Nukleotid. Zwei Polynukleotid-Einzelstränge sind durch Basenpaare zu einem Doppelstrang verknüpft: sie bilden die Doppelhelixstruktur der DNA. Dabei bilden Adenin und Thymin, sowie Cytosin und Guanin immer ein Basenpaar.

Ein Gen ist ein Abschnitt auf der DNA. Jedes Gen hat eine definierte Position auf dem Chromosom, den sogenannten Genlocus. Wie in Abbildung 5 zu erkennen, sind Gene durch Exons und Introns aufgebaut. Exons sind Abschnitte der DNA, die Informationen enthalten, um ein Protein zu kodieren. Introns sind nicht kodierende Abschnitte eines Gens, die die kodierenden Abschnitte voneinander trennen. Proteine sind für unterschiedliche Funktionen zuständig, wie zum Beispiel Stoffwechsel oder den Schutz des Körpers durch Antikörper (Goodsell 2010).

[1] http://www.ensembl.org/Homo_sapiens/Info/Index (letzter Zugriff 16.März 2011)

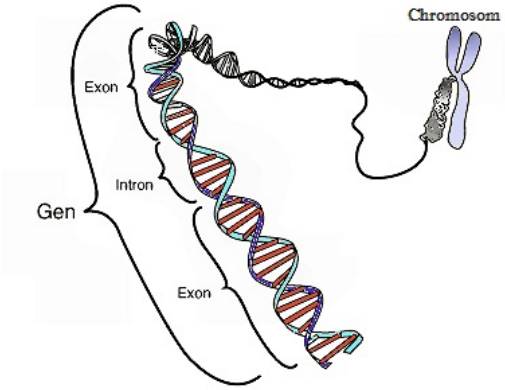


Abbildung 5:
Der Aufbau eines Gens
Quelle: http://de.wikipedia.org/wiki/Gen
(letzter Zugriff 10.März 2011)

Der Weg eines Gens zum Protein führt über sein Transkript, der mRNA (messenger Ribonukleinsäure). Diese besteht nur noch aus Exons und wird mittels der so genannten Translation in die Aminosäuren übersetzt, die das Protein bilden (Siehe Abbildung 6). Über die mRNA kann durch Reverse Transkiption komplementäre DNA (cDNA) hergestellt werden. cDNA wird nur für wissenschaftliche Zwecke in der Molekularbiologie oder zur medizinischen Diagnostik hergestellt und kann durch eine Polymerase-Ketten-Reaktion (PCR) vervielfältigt werden.

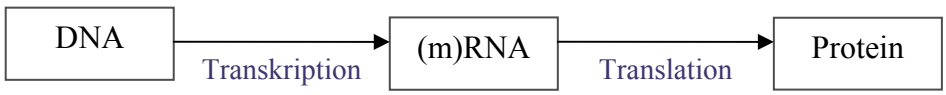


Abbildung 6:
Der Weg von der DNA über die RNA zum Protein wird als ‚Zentrales Dogma der Molekularmedizin' bezeichnet.

Die Umsetzung von genetischen Informationen nennt man Genexpression. (Neis-Beeckmann 2009) Um ein Gen in der Biomedizin eindeutig zu identifizieren und klassifizieren, sind Informationen über Lage und Funktion notwendig:

Zur Lokalisation:
- chromosomale Position und Struktur (Anzahl und Größe von Exons und Introns)
- Größe sowie Sequenzinformation des Transkripts oder der genomischen DNA (teilweise oder komplett)

Zur Funktion:
- Expressionsmuster in verschiedenen Geweben
- Rückschlüsse auf die normale Funktion durch Vergleich mit Genen ähnlicher Struktur und Sequenz
- Ausfallerscheinungen bei Mutationen (Passarge 2008)

An diese Informationen gelangt man durch DNA-Sequenzierung, der Bestimmung der Nukleotid-Abfolge der DNA, sowie durch die Genexpressionsanalyse.
Eine tiefer gehende Beschreibung über Gene und der Sequenzierungsverfahren findet sich unter anderem bei (Goodsell 2010) und (Neis-Beeckmann 2009). Da die Details dieses Themas für diese Arbeit nicht relevant sind und den inhaltlichen Rahmen sprengen würden, sei auf diese Quellen als Empfehlungen hingewiesen.

2.6 Gendatenbanken

Es gibt zwei Arten von Gendatenbanken. Das verbreitete Verständnis unter dem Begriff Gendatenbank ist eine Datenbank, die zum Beispiel Informationen über menschliches Erbgut beinhaltet, welches einem Individuum zugeordnet werden kann. Diese dienen zum Beispiel:

- der Strafverfolgung bei gefundener DNA am Tatort,
- in der Medizin, um beispielsweise einen passenden Organspender zu finden und
- in der Biologie, um Verwandtschaftsgrade zu ermitteln.[1]

Der andere Typ der Datenbankart, die in dieser Arbeit betrachtet wird, beinhaltet sequenzierte Genome ohne Bezug auf ein einzelnes Individuum. Sequenzierte Genome dienen der Klassifizierung und Beschreibung der Funktionen und Eigenschaften von Genen. Im Folgenden werden bereits vorhandene und für den zu entwickelnden Dienst relevante Gendatenbanken dieser Variante vorgestellt. Alle diese Datenbanken sind online zugänglich. Neben der allgemeinen Vorstellung der Datenbank, wird die Darstellung der Gen-ID für sequenzierte Gene, die Zugriffsmöglichkeiten auf die Datenbank sowie die Verlinkungen zu anderen Datenbanken zusammengestellt.

2.6.1 Ensembl-Datenbank

Die Ensembl Datenbank ist ein Gemeinschaftsprojekt zwischen EMBL-EBI und dem englischen Wellcome Trust Sanger Institute (WTSI) und setzt ihren Schwerpunkt in den Genomen der Wirbeltiere. Die Datenbank, die seit 1999 existiert, hat das Ziel, Genome automatisch zu annotieren, diese Annotationen mit anderen biologischen Daten zu verknüpfen und diese Informationen über das Web verfügbar zu machen.
Die Identifikation von Genen bei Ensembl fängt mit ‚ENSG' als Schlüsselwert an. Danach folgt eine elfstellige Zahl. Beispielsweise hat das Gen ‚BRCA2' die Ensembl ID: ENSG00000139618. ENSG ist das Präfix für das menschliche Genom. Für Mausgene zum Beispiel beginnt die ID mit ENSMUSG.
Es gibt verschiedene Möglichkeiten, einen Zugriff auf die Daten der Ensembl Datenbank zu bekommen:

- Zum einen ist der Zugang zur Ensembl Datenbank über einen Webbrowser möglich. Dort kann man ein Gen über die ID oder über einen Namen suchen und Anzeigen lassen.
- Für den Abruf einer größeren Menge von Daten wird empfohlen, den Zugang über einen öffentlich-verfügbaren MySQL-Server zu nutzen. Auf den Server kann über die von Ensembl zur Verfügung gestellte Schnittstelle (Perl-API) oder über ein MySQL-Clientprogramm zugegriffen werden.
- Als weitere Zugangsmöglichkeit zu den Daten aus der Ensembl-Datenbank bietet Ensembl das Data Mining Tool ‚BioMart' an. Dieses Tool bietet einen Filter an, mit dem explizit gewünschte Informationen, wie zum Beispiel zum menschlichen Genom, heruntergeladen werden können. Die erhaltenen Informationen werden als komma- bzw. tabulatorseparierte Datei, als Excel-Datei oder als HTML-Datei gespeichert.
- Die letzte Möglichkeit Zugang zu den Daten von der Ensembl Datenbank zu bekommen, ist eine Kopie der kompletten Datenbank über die FTP-Seite (ftp://ftp.ensembl.org/pub/) herunterzuladen.

Jeder Geneintrag der Ensembl-Webseite hat mehrere verschiedene Querverweise auf anderen Seiten. Diese Verweise führen unter anderem zur HGNC-, zur VEGA-, zur CCDS- und zur ‚Entrez Gene'-Webseite.[2]

[1] http://de.wikipedia.org/wiki/Gendatenbank (letzter Zugriff 10.November.2010)
[2] http://www.ensembl.org (letzter Zugriff 16.Februar 2011)

2.6.2 Entrez-Datenbank

Entrez Gene ist eine Datenbank des amerikanischen National Center for Biotechnology Information (NCBI), die genspezifische Informationen speichert. Der Schwerpunkt dieser Datenbank liegt dabei auf Genomen,

- die komplett sequenziert wurden,
- die eine aktive Forschungsgemeinschaft haben, die genspezifischen Informationen einbringt,
- für die intensive Sequenzanalysen vorgesehen sind.

Die ID von Genen wird meistens mit ‚GeneID: ' gekennzeichnet, besteht aus ganzen Zahlen und ist artspezifisch. Artspezifisch heißt, dass es abhängig von dem Lebewesen ist, von dem das Gen kommt. Anders als bei Ensembl wird die ID nicht mit Nullen aufgefüllt um eine feste Länge zu erzielen. Ein Beispiel für eine ID der Entrez Gene-Datenbank ist: GeneID: 675. Dieses ist die Gen ID für das Gen ‚BRCA2'
Der Zugang zu den Informationen, die in der Entrez Gene-Datenbank gespeichert sind, ist wie bei der Ensembl-Datenbank auf mehrere Arten möglich. Die einfachste Möglichkeit ist dabei, ein Gen über den Webbrowser zu suchen. Eine weitere ist der Zugang über die FTP-Seite ‚Gene ftp' (ftp://ftp.ncbi.nih.gov/gene/). Hier kann man zum Beispiel alle Geninformationen von der Klasse der Säugetiere herunterladen oder nur die Geninformationen einer bestimmten Spezies.
Die Einträge der Entrez Datenbank sind in allen Datenbanken der NCBI verlinkt. Damit kann der Zugriff auf die Datenbank auch über den Link einer anderen Datenbank erfolgen. Als Beispiel: In einem Artikel der PubMed-Datenbank werden mehrere Gene erwähnt. Über die Option "Find related data" findet der Benutzer die Einträge zu diesen Genen.
Umgekehrt gibt es auch von den Einträgen Verknüpfungen nach außen. Diese sind unter anderem zu: zugehörigen Artikeln in PubMed, zur entsprechenden Genseite in der Ensembl-Datenbank und zum Eintrag auf der HGNC-Webseite. (Maglott u. a. 2010)

2.6.3 VEGA Datenbank

Die Havanna Gruppe im WTSI hat 2004 die ‚Vertebrate Genome Annotation'-Datenbank, kurz VEGA, entwickelt. Das Ziel von VEGA ist es, manuelle Annotationen von Mensch-, Maus- und Zebrafisch-Genomen zu sammeln und zu veröffentlichen. Diese drei sind die einzigen Genome von Wirbeltieren, die komplett sequenziert wurden. Mittlerweile wurden von einigen weiteren Wirbeltieren kleinere Bereiche der Genome annotiert, die von speziellem biologischem Interesse sind.
Die Datenbankstruktur ist technisch wie die Ensembl-Datenbank aufgebaut. Der wichtigste Unterschied zu Ensembl und zu anderen Datenbanken besteht darin, dass die VEGA-Datenbank nur Annotationen enthält, die durch manuelle Sichtung und Korrektur bestätigt wurden. Bei den meisten anderen Datenbanken werden hingegen fehlende Informationen durch automatisch abgeleitete Genvorhersagen ergänzt.
Eine Gen ID aus der VEGA Datenbank wird durch den Präfix ‚OTTHUMG' zusammen mit einer elfstelligen Zahl dargestellt. Die ID für das Symbol ‚BRCA2' ist in VEGA zum Beispiel: OTTHUMG00000017411
Die Vega-Datenbank bietet Zugang:

- für die Suche nach einem Gen über einen Webbrowser
- für das Herunterladen von größeren Datenmengen über den FTP-Server (ftp://ftp.sanger.ac.uk/pub/vega/)

VEGA verlinkt auf seinen Genseiten auf den jeweils zugehörigen HGNC-, CCDS-, sowie Ensembl-Eintrag. (Ashurst 2004)

2.6.4 Übersicht Gendatenbanken

Im Folgenden werden noch einmal die wichtigsten Eigenschaften der genannten Gendatenbanken tabellarisch aufgeführt. Die ID-Beispiele beziehen sich alle auf das Gen ‚BRCA2'

	Entrez Gene	**Ensembl**	**VEGA**
Homepage	http://www.ncbi.nlm.nih.gov/gene	www.ensembl.org	http://vega.sanger.ac.uk
ID-Darstellung	‚GeneID: ' + kurze Zahl GeneID: 675	ENSG + elfstellige Zahl ENSG00000139618	OTTHUMG + elfstellige Zahl OTTHUMG00000017411
Verweise	Interne Verlinkungen der NCBI	-	-
	CCDS	CCDS	CCDS
	Ensembl	-	Ensembl
	-	Entrez Gene	-
	HGNC	HGNC	HGNC
	PubMed	-	-
	-	VEGA	-
Zugang zu Daten	Webbrowser	Webbrowser	Webbrowser
	FTP ftp://ftp.ncbi.nih.gov/gene/	FTP ftp://ftp.ensembl.org/pub/	FTP ftp://ftp.sanger.ac.uk/pub/vega/
	-	BioMart	-
	-	MySQL-Server	-

Tabelle 1
Eine Übersicht über Eigenschaften von Gendatenbanken.

3 Methodik

Die einzelnen Projekte des ‚SFB/TRR 77' führen experimentelle Untersuchungen durch. Im Rahmen ihrer Versuche sammeln sie Dateien, die Geninformationen zu den einzelnen Proben in ihren Versuchsreihen enthalten. Diese werden für ihre Fragestellung ausgewertet. Die Formate der Versuchsergebnisse und auch die verwendete Genidentifikation hängen von den in den Projekten eingesetzten Messinstrumenten (wie zum Beispiel Microarrays) und deren Software ab.
Die Ausgangssituation für die Forscher ist also, dass jedes Projekt nur über die eigenen Daten verfügt.

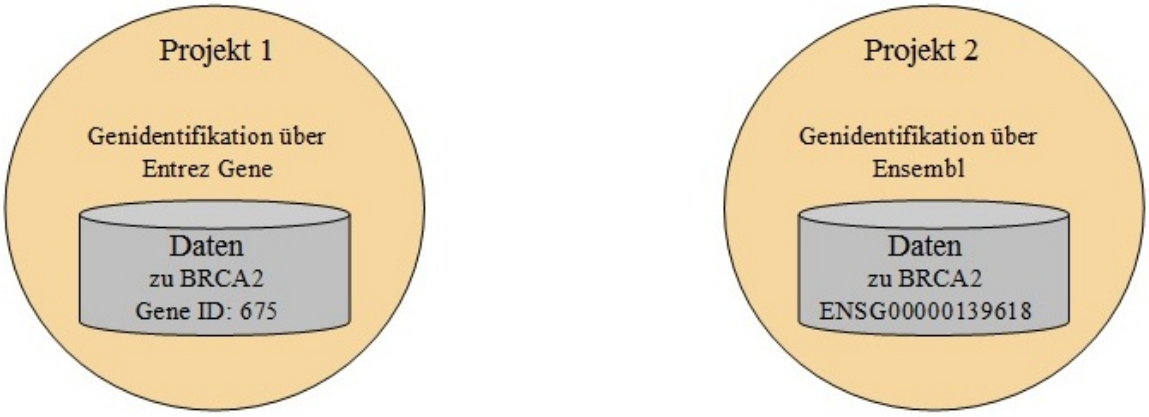


Abbildung 7:
Zwei verschiedene Projekte, die Forschung zu dem gleichen Gen betreiben. Bei Projekt 1 werden die Gene über die Entrez Gene-Datenbank identifziert, bei Projekt 2 über die Ensembl-Datenbank

Mit dem Zentralprojekt Z2-IT sollen die Versuchsergebnisse in einer Daten-Cloud projektübergreifend zur Verfügung gestellt werden. Das Zugreifen über die eigene ID liefert jedoch keine Ergebnisse und ungezieltes Zugreifen (oder die Suche im Heuhaufen) auf verwendbare Daten ist nicht effizient.

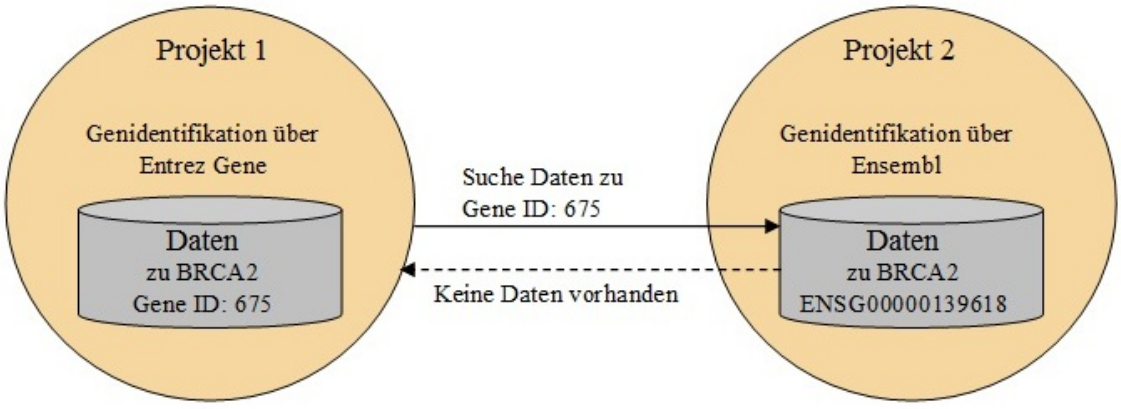


Abbildung 8:
Projekt 1 möchte nun, um weitere Analysen durchzuführen, Daten zu dem Gen BRCA2 aus den Daten von Projekt 2 anfordern. Es übermittelt die Gen ID für die Anfrage. Die Suche nach der Entrez Gene ID in den Daten von Projekt 2 liefert jedoch kein Ergebnis, obwohl Daten zum Gen BRCA2 vorhanden sind.

Eine Lösung des Problems ist, die vorhandenen Primärdaten mit weiteren Metadaten (Standard Genidentifikation) zu ergänzen oder bei der Datensuche Genidentifikationen automatisch durch einen Service um die äquivalenten Genidentifikationen (gemappte IDs) zu erweitern.

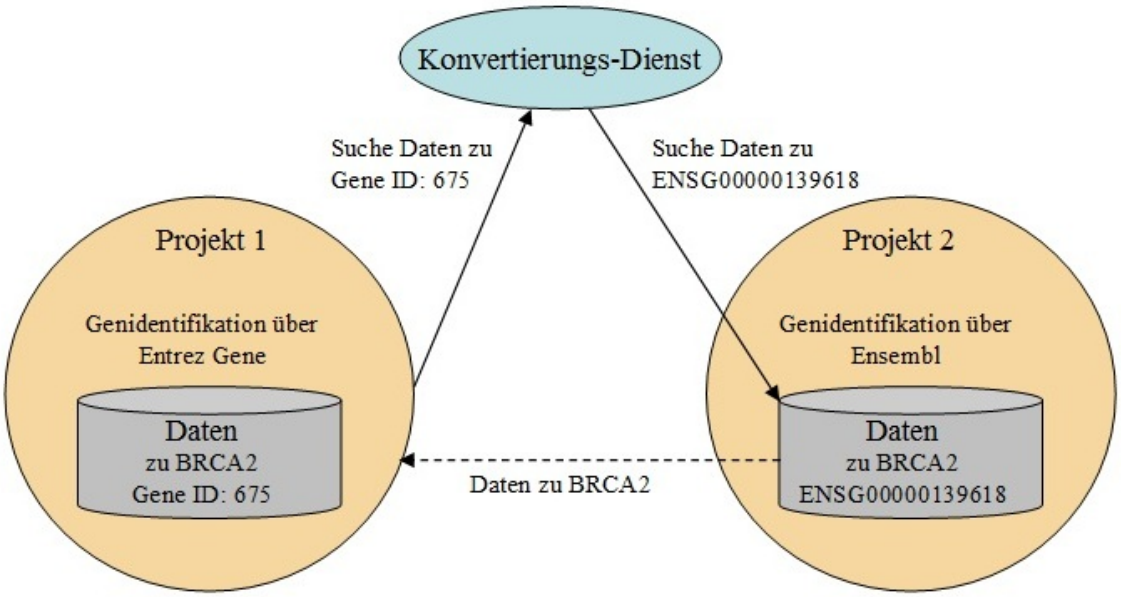


Abbildung 9:
Lösung des Genidentifikationproblems: Die ID von Projekt 1 wird an einen Konvertierungsdienst geschickt. Dieser mappt die Entrez Gene-ID auf die Ensembl ID und gibt die Suche an Projekt 2 weiter. Jetzt findet Projekt 2 Daten zu dem Gen und kann diese an Projekt 1 liefern.

Diese Arbeit stellt einen Genkonvertierungs-Dienst zur Verfügung, der prinzipiell in beiden Fällen verwendet werden kann.
Es waren also drei Schritte notwendig, um dies zu erreichen:

- Analyse von Methoden und Standards zur eindeutigen Identifikation von Genen
- Analyse von caBIG auf Methoden und Werkzeuge zur Unterstützung der eindeutigen Genidentifikation
- Entwicklung eines Genkonvertierungs-Dienstes für Gen-IDs zur Einbindung in caBIG

3.1 Analyse von Standards zur eindeutigen Identifikation von Genen

Ein erster Schritt, um Standards für eine eindeutige Identifikation zu finden, ist es verschiedene Gendatenbanken zu untersuchen mit dem Fokus, wie diese die eindeutige Genidentifikation gelöst haben. Dabei wird betrachtet, welche Identifikationsmöglichkeiten bei mehreren Datenbanken auftreten, ob diese eindeutig auch über die Datenbank hinaus sind und ob sie für die eindeutige Identifikation von Genen brauchbar sind. (siehe Kapitel 4.1)
Neben den Gendatenbanken sollen auch verschiedene biomedizinische Standards analysiert werden. Dafür wird erst der Standard im Allgemeinen betrachtet und anschließend wird untersucht, wie und wo der Standard mit Gendaten umgeht, sowie ob und wie eine eindeutige Genidentifikation sichergestellt wird. Dazu werden neben der Spezifikation des Standards gegebenenfalls Beispieldaten, die standardkonform sind, durchgesehen.

3.2 Analyse von caBIG auf Methoden und Werkzeuge zur Unterstützung der eindeutigen Genidentifikation

Diese Arbeit ist Bestandteil einer integrierten Informationsplattform, die auf Tools von caBIG aufbaut. caBIG stellt bereits verschiedene Dienste zur Verfügung, die auf der caBIG-Webseite, sowie teilweise in verschiedenen Artikel, dokumentiert sind. Diese Dienste wurden auf Funktionalität und Umfang untersucht. Dabei war der Aspekt wichtig, ob ein Tool das Problem der eindeutigen Genidentifikation lösen kann oder wie es in den verschiedenen Tools gelöst wurde. (siehe Kapitel 4.3)

3.3 Entwicklung eines Mapping-Dienstes für Gen-IDs zur Einbindung in caBIG

Um ein Konzept für einen Dienst, der Genbezeichnung aufeinander abbilden kann, zu entwickeln, werden Stichpunktartig Genbezeichnungen aus zwei verschiedenen Datensätzen genommen. Über diese wird versucht, die äquivalente Genbezeichnung für den anderen Datensatz und damit einen Eintrag zu dem gleichen Gen zu finden. Mit diesem Vorgehen soll eine Methode gefunden werden, die zur Genkonvertierung verwendet werden kann.
Auf diesem Ergebnis aufbauend wird ein Datendienst erstellt, der Genbezeichnungen konvertieren kann. Dieser wird mit einer MySQL-Datenbank (My Structured Query Language-Datenbank) aufgebaut.
Das Einlesen von Daten in die Datenbank soll über ein Javaprogramm erfolgen. Java ist eine objektorientierte und plattformunabhängige Programmiersprache, die von Sun Microsystems entwickelt wurde. Java eignet sich sehr gut, um Daten in eine Datenbank einzulesen. Die Objektorientierung ermöglicht es, Daten vor dem Einleseprozess in objektspezifischen Vektoren zu sortieren. Darüber hinaus unterstützt Java Implementierungen durch eine umfangreiche Klassenbibliothek. Die Datenbankschnittstelle der Java-Plattform ‚Java Database Connectivity' (JDBC) setzt die Voraussetzungen, um relationale Datenbanken mit Java ansprechen zu können (Ullenboom 2009).
Laut Aufgabenstellung wurde keine Anforderung an die Wahl der Programmiersprache gestellt, so dass Java auch aufgrund individueller Fähigkeiten und guter Erfahrungen in anderen Datenbankprojekten ausgesucht wurde.
Damit die Datenbank in ‚pelican' eingebunden werden kann, muss letztlich noch ein Webservice auf Basis der Datenbank erzeugt werden. Dies soll mit dem caBIG Tool caCore Software Development Kit (SDK) realisiert werden. Es basiert auf den Prinzipien der modellgetriebenen Architektur (MDA, engl. Model-Driven Architectur). Die MDA ist eine neue Form der Softwareentwicklung. Sie beruht auf der Idee, dass wiederkehrende Aufgaben im Entwicklungsprozess durch automatisierte Prozesse abgelöst werden können. Durch die Abbildung der Domäne, ihrer Anforderungen, Objekte und Interaktionen in einer einheitlichen Form (Modellen) sind Code-Generatoren in der Lage, diese Informationen zu verarbeiten und zu lauffähigem Code zu übertragen. Somit stehen schnell die benötigten Datenbank-Strukturen und Schnittstellen zur Verfügung. In den meisten Fällen ist nur noch die manuelle Implementierung der Applikationslogik notwendig. Ein großer Vorteil der MDA ist, dass bei Änderung der Anforderung nur ein geringer Aufwand betrieben werden muss, um die eigene Applikation an die neuen Umstände anzupassen.
Die Validierung des Genkonvertierungs-Diensts erfolgt im Vergleich zu GeneConnect. Da die Validierung manuell, mit Daten aus einem aCGH- und einem Methylierungsdatensatz, durchgeführt werden soll, wird nur eine Stichprobe von insgesamt zehn Datensätzen genommen. Es werden jeweils fünf Stichproben aus den zwei verschiedenen Datensätzen entnommen und per Genkonvertierungs-Dienst und GeneConnect konvertiert. Dabei sollen die folgenden Aspekte verglichen werden:

- Welches Ergebnis liefert der jeweilige Dienst und stimmen beide Ergebnisse überein?
- Welche Informationen werden zu dem Ergebnisse angezeigt?
- Welche Möglichkeiten gibt es, um das Gen zu konvertieren?
- Sind die Daten korrekt, wenn sie mit der jeweiligen Gendatenbank überprüft werden?

4 Ergebnisse

4.1 Eindeutige Identifikation von Genen

Die Beschreibung eines Gens besteht in den verschiedenen betrachteten Gendatenbanken mindestens aus einem Gennamen, einem Gensymbol und aus einer Gen-ID. Der Genname ist eine kurze Beschreibung der Eigenschaft oder der Funktion des Gens. Das dazugehörige Symbol ist eine Abkürzung für den Gennamen und besteht aus lateinischen Großbuchstaben oder aus einer Kombination aus lateinischen Großbuchstaben und arabischen Zahlen. So ist zum Beispiel das Symbol ‚STIP1' die Abkürzung für den Gennamen ‚**st**ress-**i**nduced-**p**hosphoprotein **1**'.
Die ID ist von Gendatenbank zu Gendatenbank verschieden. Die Darstellung der ID lässt auf die dazugehörige Gendatenbank schließen. Das Präfix ‚ENSG' steht zum Beispiel für die Ensembl Datenbank.
In der Forschung wird ein Gen auch über seinen Locus, der Ort auf dem Chromosom, auf dem das Gen liegt, identifiziert. Der Genlocus kann sich bei Genen, die sich auf einem noch nicht komplett sequenzierten DNA Abschnitt befinden durch weitere DNA Analysen ändern. Aus diesem Grund geben die Forscher neben dem Genlocus häufig auch eine Versionsnummer an.
In wissenschaftlichen Veröffentlichungen wird neben dem Locus oft das Symbol des Genes genannt. Aber auch bei der Verwendung des Gen-Symbols in der Forschung treten Probleme bei der eindeutigen Identifikation auf:

- Für ein Gen sind mehrere Symbole in Verwendung.
  Als Beispiel: Das Gen ‚HOPX' ist auch unter dem Symbol ‚LAGY' bekannt
- Ein Symbol wird für mehrere Gene verwendet.
  Als Beispiel: Das Symbol ‚HOP' wird für das Gen ‚HOPX' aber auch für ‚STIP1' verwendet.

Um das Problem der Mehrdeutigkeit zu lösen, wurde das HUGO Gene Nomenclature Committee, kurz HGNC, gegründet. HUGO steht hierbei für die Human Genome Organisation. HGNC versucht die Identifikationen zu standardisieren und lässt für jedes menschliche Gen nur einen einzigen Gennamen und ein einziges anerkanntes Gensymbol zu. Jedes Symbol ist dabei eindeutig. Alle anerkannten Gensymbole werden in einer öffentlichen Datenbank gespeichert, die unter anderem öffentlich online zugänglich ist.
Neben dem anerkannten Symbol werden in der HGNC-Datenbank bekannte ‚Aliases' gespeichert, das heißt Symbole, die auch in der Forschung verwendet wurden und werden, um auf dieses Gen zu verweisen. Darüber hinaus werden weitere Informationen, wie zum Beispiel der Genname, der Genlocus und die IDs von den verschiedenen Gendatenbanken, bereitgestellt. Die Datenbank enthält jedoch keine zusätzlichen Details, wie zum Beispiel die DNA-Sequenz an sich.
Die HGNC-Datenbank bietet keine Schnittstelle für den Zugriff mit Hilfe eines Programms und kann deshalb nicht direkt in das caGrid eingebunden werden. [1]

Tabelle 2 zeigt einen Überblick über die Vor- und Nachteile der verschiedenen Möglichkeiten, ein Gen zu identifizieren. Für den Genkonvertierungs-Dienst ist das Symbol als Identifikationsmerkmal am besten geeignet. Dieses wird in wissenschaftlichen Veröffentlichungen häufig verwendet und es ist vor allem sprechend ist, das heißt, dass ein Forscher allein durch das Symbol auf die Funktion des Gens schließen kann.

[1] http://www.genenames.org/ (letzter Zugriff 17.Februar 2011)

Identifikation über	Vorteil	Nachteil
ID (Präfix Datenbank + Zahl)	-	• Datenbank-spezifisch • Nicht sprechend
Genlocus	Häufig von Forschern verwendet	Unterliegt Veränderungen / Versionsabhängig
Symbol (Buchstabenabkürzung)	• In wissenschaftlichen Veröffentlichungen verwendet • HGNC, die versucht die Symbole zu vereinheitlichen • sprechend	• Kein verpflichtender Standard • Teilweise mehrdeutig
Name (Beschreibung)	-	Sehr lang

Tabelle 2
Zusammenstellung der Vor- und Nachteile der jeweiligen Identifikationsmöglichkeiten.

4.2 Analyse von Biomedizinschen Standards

Im Folgenden werden biomedizinische Standards auf den Aspekt hin untersucht, mit welchen Methoden sie Gene identifizieren. Dies soll weitere Möglichkeiten aufzeigen, wie Gene eindeutig identifiziert werden können. Die folgenden Standards werden dabei betrachtet:

- Die Clinical Data Interchange Standard Consortium (CDISC), dass sich unter anderem mit der Standardisierung von klinisches Versuchsdaten auseinandersetzt
- Der Kommunikationsstandard Health Level 7 (HL7)
- Minimum Information About A Microarray Experiment (MIAME), ein Standard für Microarraydaten

4.2.1 CDISC

CDISC ist eine offene, multidisziplinäre Non-Profit-Organisation. Sie hat Standards eingeführt, die die Erfassung, den Austausch, die Eingabe und die Archivierung von klinischen Versuchsdaten unterstützen. Das Ziel von CDISC ist es, globale, plattform-unabhängigen Datenstandards für die klinisches Forschung zu entwickeln, um die Interoperabilität zwischen Informationssystemen zu ermöglichen und damit die medizinische Forschung und die zugehörigen Bereiche im Gesundheitswesen zu verbessern.[1] Zu diesen Standards gehören unter anderem (CDISC o. J.):

- Operational Data Model (ODM): Ein technisches Standarddatenmodell, das die Erfassung, den Austausch, und die Archivierung von klinischen Versuchsdaten unterstützt.
- Laboratory Data Model (LAB): LAB ist ein Standard für den Datenaustausch zwischen klinischen Laboratorien und Forschungsinstituten.
- Study Data Tabulation Model (SDTM): Beschreibt inhaltliche Struktur von Tabellen, in denen Case Report Forms (CRF, Erhebungsbogen) von klinischen Studien zusammengefasst werden können.
- Protocol Representation (PR): Dieser Standard unterstützt den Austausch von klinischen Studienprotokollinformationen
- Terminology: Das kontrollierte Vokabular für alle CDISC Standards.

[1] http://www.cdisc.org/mission-and-principles (letzter Zugriff 28.Februar 2011)

In ODM ist eine Struktur für alle Informationen einer Studie und ihrer Daten vorgesehen. Dies schließt alle Metadaten samt Versionierung ein. Jedoch gibt es in ODM derzeit keine Semantik zu einzelnen Studiendaten. Das heißt, dass nicht festgelegt ist, in welche Form eine Information gespeichert werden muss. Ein Gen kann also sowohl durch das Symbol, als auch durch den Genlocus angegeben werden (Drepper & Semler o. J.).

4.2.2 HL7

HL7 beinhaltet eine Reihe von Standards, die die Kommunikationsinhalte und Austauschformate auf der Anwendungsebene spezifizieren und damit die Kommunikation zwischen Institutionen im Gesundheitswesen ermöglichen. Standards, die zu HL7 gehören, sind zum Beispiel:

- HL7 Version 2: Dieser wird vorwiegend zwischen Systemen von Krankenhäusern eingesetzt.
- HL7 Version 3: Ein Standard für den Nachrichtenaustausch im gesamten Gesundheitswesen
- Clinical Document Architecture (CDA): CDA ist ein Teil von HL7 Version 3, der die Struktur und den Inhalt von medizinischen Dokumenten standardisiert.

Für die semantische Interoperabilität verwendet HL7 verschiedene Terminologien, die von anderen Organisationen verwaltet werden. Zu diesen Terminologien gehören unter anderem:

- LOINC (Logical Observation Identifier Names and Codes): Nomenklatur zur eindeutigen Kennzeichnung von Laborwerten, klinischen Untersuchungen, medizinischen Dokumententypen und anderen medizinischen Parametern.[1]
- Snomed-CT (Systematized Nomenclature of Medicine – Clinical Terms): Nomenklatur zur Beschreibung medizinischer Sachverhalte und zusätzlicher Informationen. Wird genutzt, um möglichst detailliert Informationen über Patienten austauschen zu können
- ICD (Internationale Klassifikation der Krankheiten): Klassifikation, mit der Krankheiten und Symptome eindeutig codiert werden. Wird unter anderem zu Abrechnungszwecken genutzt und ist deshalb weit verbreitet.
- UCUM (Unified Code for Units of Measure): Kodiersystem, das Maßeinheiten aus Medizin und Pharmazie in standardisierter Form abbildet (Blobel u. a. 2009).[2]

HL7 verwendet als Terminologie unter anderem LOINC. LOINC identifiziert ein Gen über das Gensymbol. Jedoch schreibt LOINC nicht vor, dass die Benennung zum Beispiel nach HUGO erfolgen muss (McDonald u. a. 2009).

4.2.3 MIAME

Microarrays, oder auch Genchips, sind Werkzeuge zur Analyse der Genexpression. Sie bestehen aus einer dünnen Membran oder aus Glasobjektträgern, die Proben von einer Vielzahl von Genen enthalten, die in einem regelmäßigen Muster angeordnet sind. [3]
MIAME ist ein Standard für Microarraydaten. Dieser beschreibt die minimalen Informationen, die eine Microarrayanalyse beinhaltet muss, um eine eindeutige Interpretation des Ergebnisses und gegebenenfalls eine Reproduktion des Experiments zu ermöglichen. Diese Informationen lassen sich in sechs Bereiche aufteilen (Brazma u. a. 2001), (Steinmetz 2005) :

[1] http://www.dimdi.de/static/de/ehealth/loinc/loincbasis.htm (letzter Zugriff 28.Februar 2011)
[2] http://www.dimdi.de/static/de/ehealth/ucum/index.htm (letzter Zugriff 28.Februar 2011)
[3] http://www.ncbi.nlm.nih.gov/About/primer/microarrays.html (letzter Zugriff 27.Februar 2011)

- **Experimentelles Design**: Informationen zum Experiment als Ganzes. Hierunter zählen unter anderem die Beschreibung des Experiments (auch Titel und Ziel), der Experimenttyp oder die verwendeten Variablen und Versuchsbedingungen.
- **Array Design**: Daten, wie zum Beispiel die Beschreibung des Arrays, die Arraytypdefinition oder die Darstellung, welche Gen wo auf dem Array liegen.
- **Proben**: Die Quelle (z.B. Name des Organismus) und Charakteristika (z.B. Geschlecht oder Alter) der biologischen Probe, sowie experimentelle Faktoren (z.B. gesundes / krankes Gewebe).
- **Hybridisierung**: Protokoll von Laborbedingungen, wie zum Beispiel die Temperatur, unter der die Hybridisierung stattgefunden hat oder die Hybridisierzeit.
- **Messergebnisse**: Rohdaten, also die unbearbeiteten experimentellen Ergebnisse, wie zum Beispiel Original Scans.
- **Normalisierung**: Relevante Parameter zur Normalisierung, sowie Kontrollelemente sollen spezifiziert werden. Statistischen Verfahren und Kontrollproben werden hier aufgeführt.

Abbildung 10 zeigt den Ablauf eines Microarrayexperiments mit den Daten, die bei den verschiedenen Schritten anfallen und die MIAME für die eindeutige Interpretation des Ergebnisses verlangt. Der erste Schritt ist die Extraktion von mRNA-Einzelsträngen aus den Gewebezellen (Probe). Diese werden dann in cDNA umgeschrieben und durch Hybridisierung an das Microarray gebunden (Hybridisation). Die (Roh-) Daten werden anschließend normalisiert, um die Ergebnisse repräsentativ zu machen (Denz u. a. 2010).

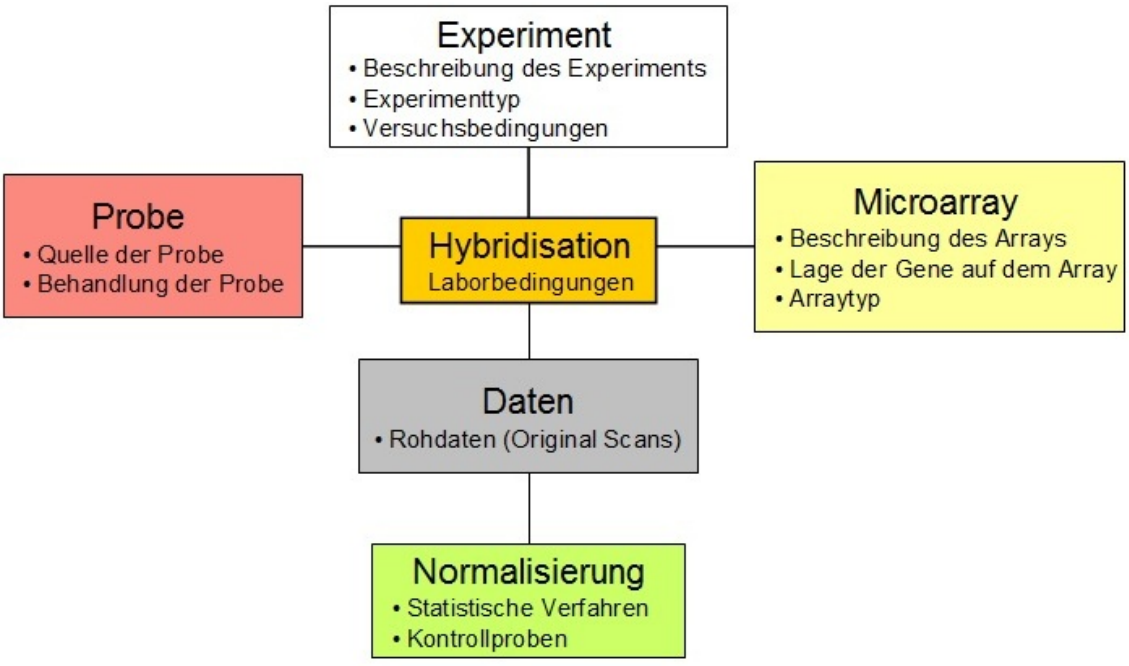


Abbildung 10:
Grafische Darstellung der verlangten Information von MIAME.
In Anlehnung an: (Rahmann 2009)

Der MIAME Standard legt zwar fest, welche minimalen Informationen nötig sind, jedoch schreibt er nicht vor, in welchem Format oder mit welcher Terminologie die Daten gespeichert werden müssen.[1] Dementsprechend werden genidentifizierende Daten höchst unterschiedlich von den verschiedenen Herstellern abgebildet. Abbildung 11 und Abbildung 12 sind MIAME-konforme Darstellungen unterschiedlicher Microarrayexperimente, die dies deutlich machen.

[1] http://www.mged.org/Workgroups/MIAME/miame.html (letzter Zugriff 28.Februar 2011)

Gendaten kommen bei MIAME vor allem im Array-Design vor. Dieses beschreibt unter anderem die Lage der Gene auf dem Microarray. Bei den beiden Abbildungen stellt eine Zeile ein bestimmtes Gen auf dem Microarray dar.
Im Array-Design der Microarrayanalyse von Abbildung 11 werden, um das Gen zu beschreiben, Gen-IDs verschiedener Gendatenbanken, wie zum Beispiel Ensembl (Spalte 2), verwendet.
Die Microarrayanalyse von Abbildung 12 hingegen verwendet keinerlei Gen-IDs, sondern nur den Genlocus als Genidentifikation.

embl	ensembl	genbank	gpcrdb	interpro	kegg	locus	omim	pfam
http://www	http://www.ensembl.o	http://www.nc	http://www.gp	http://www.eb	http://www.ge	http://www.nc	http://www3.r	http://www.sa
Composite	Composite Element D	Composite El	Composite El	Composite El	Composite El	Composite El	Composite El	Composite El
6772	ENSG00000115415	AAH02704	Hs.21486	21536301	600555	STAT	NM_007315	d1bg1a2
6772	ENSG00000115415	P42224	Hs.21486	21536301	600555	STAT	NM_007315	d1bgf__
6772	ENSG00000115415	AAH02704	Hs.21486	21536301	600555	SH2	NM_007315	d1bgf__
6772	ENSG00000115415	P42224	Hs.21486	21536301	600555	STAT	NM_007315	d1bg1a1

Abbildung 11:
Beispiel 1: Ausschnitt eines Array-Designs, eines MIAME-konformen Microarrayexperiments. Zur Genidentifikation werden mehrere Gen-IDs genannt.

Reporter Nam	Reporter Data	Reporter Data	Comment[Chr_ID]	Comment[Fish]	Comment[Block]
1	RP11-12M14	B75909	21	21q21-21q22.1	3
2	RP11-12M14	B75909	21	21q21-21q22.1	4
3	RP11-12M14	B75909	21	21q21-21q22.1	9
4	RP11-12M14	B75909	21	21q21-21q22.1	10
5	RP11-13J15	AC011275	21	21q21	1

Abbildung 12:
Beispiel 2: Ausschnitt eines Array-Designs, eines (anderen) MIAME-konformen Microarrayexperiments. Hier wird zur Genidentifikation der Genlocus verwendet (siehe Spalte 5).

4.3 Analyse von caBIG auf Methoden und Werkzeuge

caBIG bietet verschiedene Analyse- und Datendienste an. Jeder Dienst kann für sich alleine laufen. Meistens wird jedoch der Zusammenschluss mehrerer Dienste benötigt um eine größere Fragestellung zu beantworten. Die Dienste werden in das caGrid eingebunden, welches die Infrastruktur zur Verfügung stellt, die die Kommunikation der Dienste miteinander ermöglicht. In das Grid können beliebig viel caBIG- oder selbst entwickelte Dienste eingebunden werden. Nachfolgend gibt es einen Einblick in bereits vorhandene Dienste von caBIG.

4.3.1 caArray

caArray ist ein Arraydaten-Managementsystem, das über das Web oder programmtechnisch zugänglich ist. Es ermöglicht die Annotation und den Austausch von Arraydaten. In erster Linie ist caArray ein Datendienst, der durch den Erwerb, die Verbreitung und die Aggregation von semantisch interoperablen Arraydaten die translationale Krebsforschung fördert, um spätere Analysen durch Tools zu unterstützen.
caArray bietet folgende Dienste an:

- Experimente durchsuchen
- Erstellen und Verwalten von Arrayexperimenten
- Verwalten von Arraydesign, Protokollen und Vokabeltermen
- Experimente annotieren

- Hochladen, validieren und importieren von Arraydaten
- Daten von caArray extrahieren

caArray wurde betrachtet, da die meisten Daten aus den ‚SFB/TRR 77' Projekten über Mircoarrays erzeugt werden. Jedoch ist das Tool nur drauf ausgelegt Microarraydaten zu verwalten und nicht darauf Genidentifikationen zu konvertieren oder zu standardisieren.

4.3.2 caDSR

Common Data Elements (CDEs) sind Metadaten, die Datenelemente definieren, die in klinischen Studien verwendet werden. Sie wurden entwickelt, um die von Forschern und Ärzten verwendete Terminologie in klinischen Studien zu standardisieren und eine einheitliche Erfassung von Studiendaten zu unterstützen. CDEs stellen durch klare Definitionen und definierte Werte sicher, dass die Datenerhebung oder ein Vergleich zwischen verschiedenen Studien zu einer Krankheit möglich ist (Meadows u. a. 2001).
Das Cancer Data Standards Registry and Repository (caDSR) unterstützt die Entwicklung und die Bereitstellung der CDEs in der Krebsforschung. caDSR basiert auf dem ISO (International Organisation for Standardization) / IEC (International Electrotechnical Commission) 11179 Standard für Metadaten-Register. Dieser Standard enthält eine Sammlung von sorgfältig definierten Komponenten, die die Art und die Qualität von den Metadaten festsetzen, die notwendig für die Beschreibung der Daten sind.

4.3.3 GeneConnect

GeneConnect ist ein Mapping Dienst, der die Zusammenarbeit verschiedener Dienste durch Verlinkung verschiedener Gen IDs ermöglicht. Verlinkung heißt, dass er zu der Gen ID einer Datenbank die IDs anderer Datenbanken zu dem gleichen Gen findet. Die GeneConnect Ergebnissuche umfasst neben der Ensembl- und der Entrez Gene-Datenbank eine Vielzahl Datenbanken, die für diese Arbeit nicht relevant sind, während die VEGA-Datenbank nicht durchsucht wird. [1]
Abbildung 13 zeigt die Verbindung der verschiedenen Datenbanken untereinander in GeneConnect. Dabei bedeutet eine Verbindung über ‚Direct Annotation', dass die Gen ID von der Ziel-Datenbank explizit in der Quell-Datenbank genannt und verlinkt wird. Man nennt dies auch eine bidirektionale Verbindung. Existiert eine ‚Inferred Annotation'-Verbindung zwischen zwei Datenbanken, so verweist die Ziel-Datenbank auf die Quell-Datenbank, jedoch gibt es umgekehrt keinen Verweis. Die ‚Inferred Annotation' ist äquivalent zur unidirektionalen Verbindung. Waren zwischen den Datenbanken weder eine ‚Direct Annotation' noch eine ‚Inferred Annotation' vorhanden, wurden die Sequenzen der Datenbanken verglichen um Ähnlichkeiten oder sogar Übereinstimmungen zu finden. Den Vergleich zweier Sequenzen nennt man ‚Sequence Alignment'. [2]

Die Idee, Gene über ihre Gen-ID zu konvertieren, ist ein guter Ansatz für das Genkonvertierungskonzept des Dienstes. Um jedoch weitere Informationen zu einer Gen-ID, wie zum Beispiel das Gensymbol, weitere Gen-IDs oder den Genlocus, zu bekommen, müsste mindestens ein weiterer Dienst eingebunden werden. Dieser würde über eine Gen-ID weitere Informationen wie das Gensymbol oder den Genlocus liefern.
Die Möglichkeit, GeneConnect in den Dienst einzubauen, sollte nicht außer Acht gelassen werden.

[1] https://cabig.nci.nih.gov/tools/GeneConnect (letzter Zugriff 16.Februar 2011)
[2] http://cbmi.wustl.edu/html/GeneConnect.html (letzter Zugriff 17.Februar 2011)

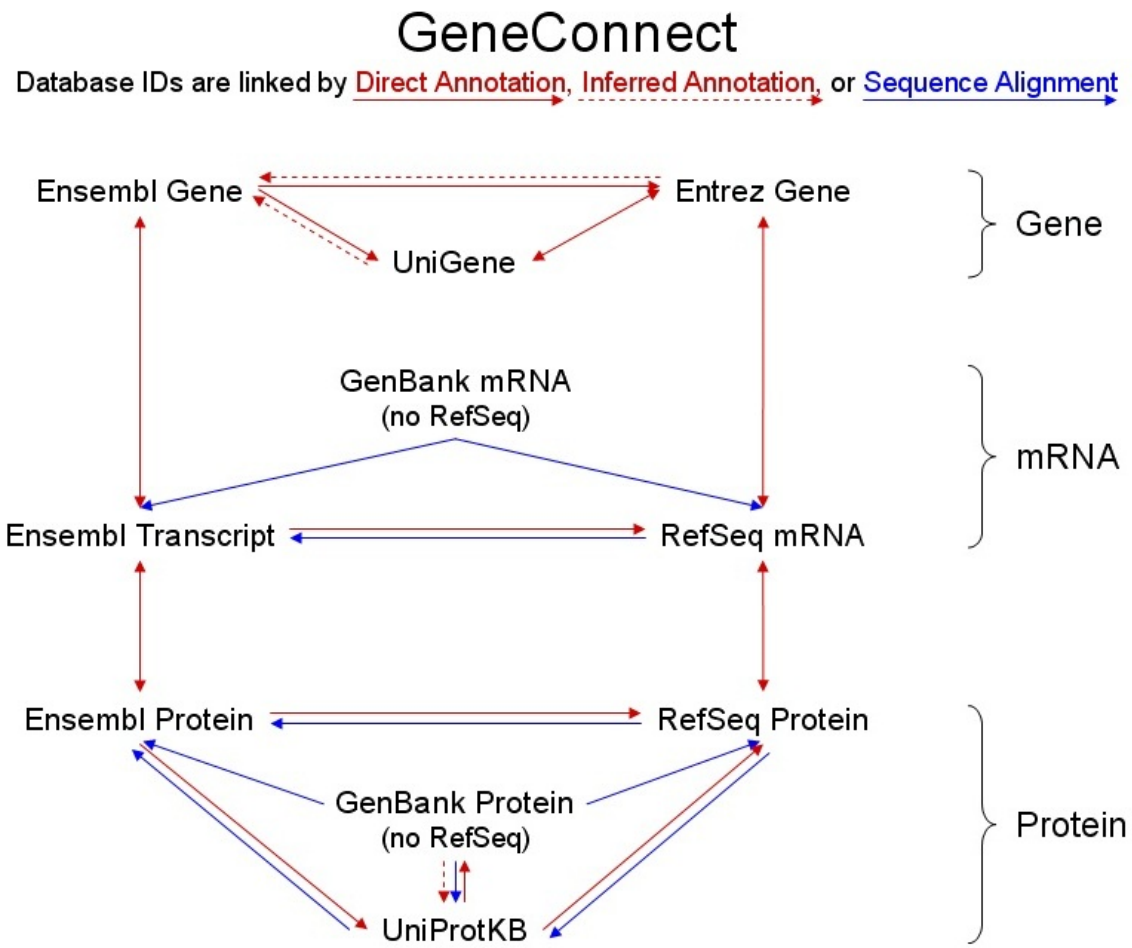


Abbildung 13:
GeneConnect verlinkt mehrere Datenbanken untereinander. Diese Abbildung zeigt, wie die Datenbanken miteinander verbunden sind.
Quelle: https://cabig.nci.nih.gov/tools/GeneConnect

4.4 Entwicklung einer eigenständigen IT-Lösung für das Mapping von Gen IDs

4.4.1 Methode zur eindeutigen Genidentifikation im ‚SFB/TRR77'

Die Forscher aus den Projekten des ‚SFB/TRR 77' speichern derzeit ihre Forschungsdaten überwiegend als Excel-Tabellen. Dabei werden Gen-IDs unterschiedlicher Gendatenbanken verwendet. Um das Problem der unterschiedlichen Genidentifikationen zu lösen, wurde die folgende Möglichkeit ausgearbeitet (Abbildung 14):
Um eine Gen-ID in eine andere Genbezeichung zu konvertieren wird zunächst die Quell-Datenbank ermittelt. Jede Gendatenbank stellt ihre Gen-IDs anders dar, was die Zuordnung einer bestimmt ID zu der zugehörigen Datenbank ermöglicht.
Als Beispiel wird die Gen ID ENSG000000139618 verwendet, die von der Ensembl-Datenbank stammt. Über die Gensuche auf der Ensembl-Homepage findet man heraus, dass diese ID zum Gen ‚BRCA2' gehört. Auf der Seite des Eintrags von BRCA2 befindet sich ein Link, der zur HGNC-Seite von ‚BRCA2' führt. (Abbildung 15, A) Die HGNC-Seite liefert weitere Informationen über das Gen, wie zum Beispiel Aliases, also Symbole, die auch für dieses Gen verwendet werden, oder

die Gen-IDs von anderen Gendatenbanken (Abbildung 15, B). In dem Beispiel hat man nun also die Information, dass das Gen unter anderem auch mit den Symbolen FAD, FAD1 und BRCC2 bezeichnet wird, und, dass man das Gen in der Entrez-Gene-Datenbank unter der Gen-ID 675 findet.
Dieses Beispiel zeigt, dass die Konvertierung durch vorhandenes strukturiertes Wissen und Verknüpfungen bereits möglich ist, jedoch auf manuellem Wege.

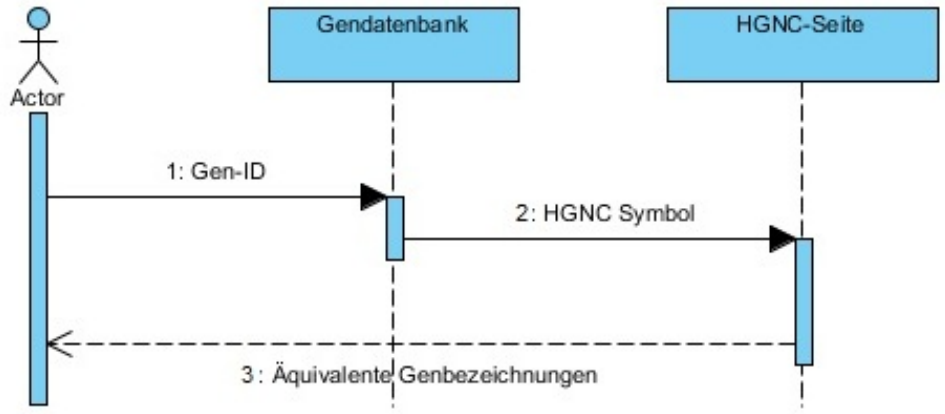


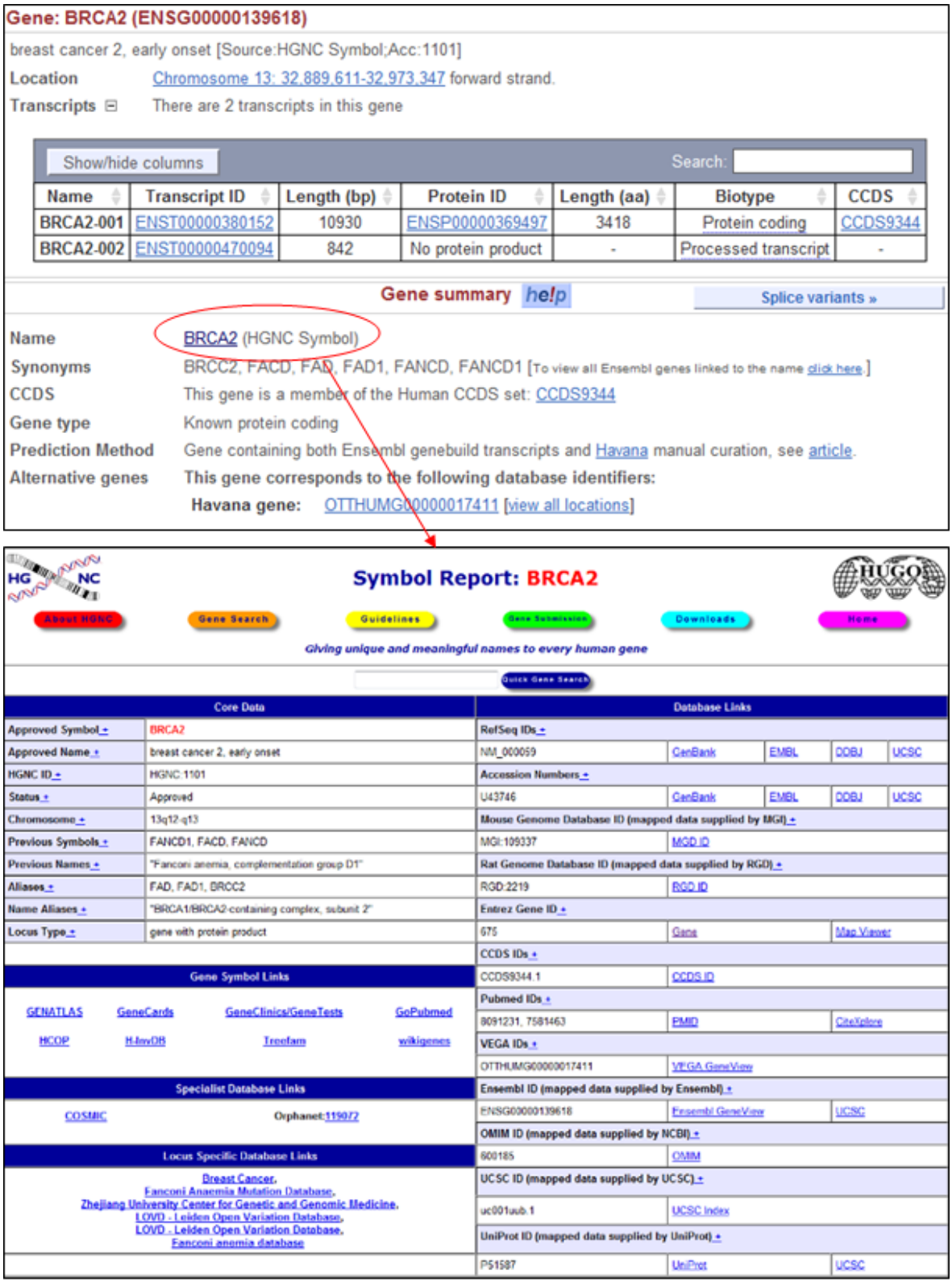
Abbildung 14:
Konzept zur Lösung des Problems der unterschiedlichen Genidentifikationen.

Die Möglichkeit, Genbezeichnungen zu konvertieren soll als Service in ‚pelican' eingebaut werden. Das Konzept muss demnach automatisiert ablaufen können. Bisher bietet nur Ensembl eine Schnittstelle für den programmatischen Zugriff auf seine Daten. Mit dem jetzigen Konzept wäre ein automatisierter Ablauf also nur möglich, wenn alle Gendatenbanken, sowie die HGNC-Datenbank, eine Schnittestelle zur Verfügung stellen würden, die einen programmatischen Zugriff ermöglichen.
Die HGNC-Webseite speichert Gen-Daten, die als Bezeichner oder als Identifikation für das Gen dienen. Das bedeutet, dass sie alle Daten beinhaltet, die zur Konvertierung einer Genbezeichnung nötig sind. Ein Zugriff auf eine Datenbank, die die HGNC-Daten beinhaltet, reicht folglich, um Genbezeichnungen automatisiert zu konvertieren. Damit diese Datenbank in ‚pelican' eingebunden werden kann, wird sie als Webservice zur Verfügung gestellt.

4.4.2 Daten-Analyse der HGNC-Daten

Die HGNC-Datenbank bietet zu jedem Gen insgesamt 40 verschiedene Informationen an. Dabei gibt es Informationen die das Gen direkt, als auch Informationen die den Eintrag in der Datenbank betreffen. Informationen die das Gen direkt betreffen sind zum Beispiel Aliase, das Gensymbol, der Genname oder der Genlocus. Bei den Informationen zum Eintrag in der Datenbank handelt es sich beispielsweise um das Datum, wann das Symbol von HUGO offiziell anerkannt wurde, oder die interne ID des Eintrags: die HGNC ID.
Im Anhang dieser Arbeit befindet sich eine Tabelle, die alle Spaltennamen erklärt. Die wichtigsten werden hier kurz aufgelistet:

- **Approved Symbol**: Das von der HGNC offiziell anerkannte Symbol des Gens.
- **Approved Name**: Der Standardname des Gens.



Gene: BRCA2 (ENSG00000139618)

breast cancer 2, early onset [Source:HGNC Symbol;Acc:1101]

Location Chromosome 13: 32,889,611-32,973,347 forward strand.

Transcripts There are 2 transcripts in this gene

Show/hide columns Search:

Name	Transcript ID	Length (bp)	Protein ID	Length (aa)	Biotype	CCDS
BRCA2-001	ENST00000380152	10930	ENSP00000369497	3418	Protein coding	CCDS9344
BRCA2-002	ENST00000470094	842	No protein product	-	Processed transcript	-

Gene summary help Splice variants »

Name BRCA2 (HGNC Symbol)

Synonyms BRCC2, FACD, FAD, FAD1, FANCD, FANCD1 [To view all Ensembl genes linked to the name click here.]

CCDS This gene is a member of the Human CCDS set: CCDS9344

Gene type Known protein coding

Prediction Method Gene containing both Ensembl genebuild transcripts and Havana manual curation, see article.

Alternative genes This gene corresponds to the following database identifiers:

Havana gene: OTTHUMG00000017411 [view all locations]

Symbol Report: BRCA2

About HGNC | Gene Search | Guidelines | Gene Submission | Downloads | Home

Giving unique and meaningful names to every human gene

Quick Gene Search

Core Data	
Approved Symbol	BRCA2
Approved Name	breast cancer 2, early onset
HGNC ID	HGNC:1101
Status	Approved
Chromosome	13q12.q13
Previous Symbols	FANCD1, FACD, FANCD
Previous Names	"Fanconi anemia, complementation group D1"
Aliases	FAD, FAD1, BRCC2
Name Aliases	"BRCA1/BRCA2-containing complex, subunit 2"
Locus Type	gene with protein product

Gene Symbol Links

GENATLAS | GeneCards | GeneClinics/GeneTests | GoPubmed | HCOP | H-InvDB | Treefam | wikigenes

Specialist Database Links

COSMIC | Orphanet:1190/2

Locus Specific Database Links

Breast Cancer.
Fanconi Anaemia Mutation Database.
Zhejiang University Center for Genetic and Genomic Medicine.
LOVD - Leiden Open Variation Database.
LOVD - Leiden Open Variation Database.
Fanconi anemia database

Database Links				
RefSeq IDs				
NM_000059	GenBank	EMBL	DDBJ	UCSC
Accession Numbers				
U43746	GenBank	EMBL	DDBJ	UCSC
Mouse Genome Database ID (mapped data supplied by MGI)				
MGI:109337	MGD ID			
Rat Genome Database ID (mapped data supplied by RGD)				
RGD:2219	RGD ID			
Entrez Gene ID				
675	Gene		Map Viewer	
CCDS IDs				
CCDS9344.1	CCDS ID			
Pubmed IDs				
8091231, 7581463	PMID		CiteXplore	
VEGA IDs				
OTTHUMG00000017411	VEGA GeneView			
Ensembl ID (mapped data supplied by Ensembl)				
ENSG00000139618	Ensembl GeneView		UCSC	
OMIM ID (mapped data supplied by NCBI)				
600185	OMIM			
UCSC ID (mapped data supplied by UCSC)				
uc001uub.1	UCSC Index			
UniProt ID (mapped data supplied by UniProt)				
P51587	UniProt		UCSC	

Abbildung 15:
Screenshot A zeigt einen Ausschnitt von der Ensembl-Datenbank (www.ensembl.org). Die Seite der Gen ID ENSG000000139618 beinhaltet unter anderem das verlinkte HGNC Symbol ‚BRCA2'. Der Link führt auf den HGNC-Eintrag (www.genenames.org) des Gensymbols, welcher in Screenshot B zu sehen ist. Unter diesem Eintrag sind neben den Aliases auch die Gen IDs der anderen Gendatenbanken zu finden.

- **Status**: Der Status teilt die Einträge in drei verschiedene Zustände ein:
  - ‚*Approved*' wenn es ein von HUGO anerkanntes Symbol ist
  - ‚*Entry withdrawn*' wenn ein früher anerkanntes Gen als nicht mehr existent angesehen wird
  - ‚*Symbol withdrawn*' wenn ein anerkannter Datensatz mit einem anderen Datensatz zusammengefügt wurde
- **Previous Symbol / Name**: Symbole / Namen, die vorher für das Gen anerkannt waren
- **Aliases**: Symbole, die benutzt werden, um auf dieses Gen zu verweisen. In dieser Spalte können beliebig viele ‚Aliase' stehen.
- **Chromosome**: Der Genlocus.

Auch Symbole, die ‚withdrawn' sind, haben in der Regel einen Eintrag in der Spalte ‚Approved symbol'.
Es gibt in der HGNC-Datenbank für fast jede Gendatenbank zwei ID-Einträge. Diese werden dadurch unterschieden, dass einer der Einträge den Zusatz ‚*mapped data*' besitzt. Der Zusatz weist darauf hin, dass die ID von einer externen Quelle kommt und von der HGNC nicht manuell geprüft wurde. Die ID-Spalten ohne diesen Zusatz beinhalten manuell überprüfte Gen-IDs.
Aktuell (März 2011) hat die HGNC Datenbank knapp 34000 Einträge. Jeder Eintrag besitzt eine eindeutige HGNC ID, ein ‚Approved Symbol' und einen ‚Approved Name'. Über 30000 der Einträge haben den Status ‚Approved' und einen datierten Genlocus. Ca. 8000 der Einträge haben ein ‚Previous Symbol', wurden also schon einmal überarbeitet.

4.4.3 Anforderung an den Service

Der Genkonvertierungs-Dienst soll verschiedene Genbezeichnungen konvertieren können. Abbildung 16 zeigt einen typischen Anwendungsfall, den der Genkonvertierungs-Dienst lösen soll. Als Eingabe bekommt der Dienst eine Gen-ID oder eine Gensymbol. Zu dieser Eingabe soll er als Ausgabe Informationen zu dem dazugehörigen Gen liefern. Solche Informationen sind zum Beispiel die Gen-IDs der verschiedenen Gendatenbanken, das offiziellem Symbol oder der Genlocus.
Ein anderer Anwendungsfall ist in Abbildung 17 dargestellt. Wenn Forscher ihre Daten zentral auf der integrierten Informationsplattform speichern wollen, so werden die Gendaten konvertiert und das offizielle HGNC-Symbol wird mitgespeichert. Damit wird sichergestellt, dass spätere Analysen alle passenden Daten zu einem Gen finden.

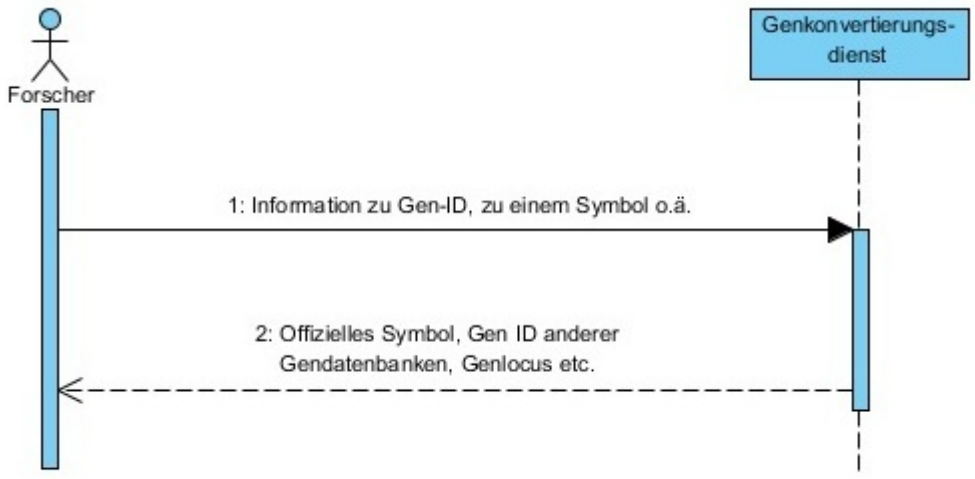


Abbildung 16:
Anwendungsfall 1: Ein Forscher möchte Informationen zu einem bestimmten Gen haben. Dabei gibt er die Gen-ID oder ein Gensymbol ein und erhält Informationen wie zum Beispiel die Gen-ID von anderen Gendatenbanken, das offizielle Symbol oder den Genlocus.

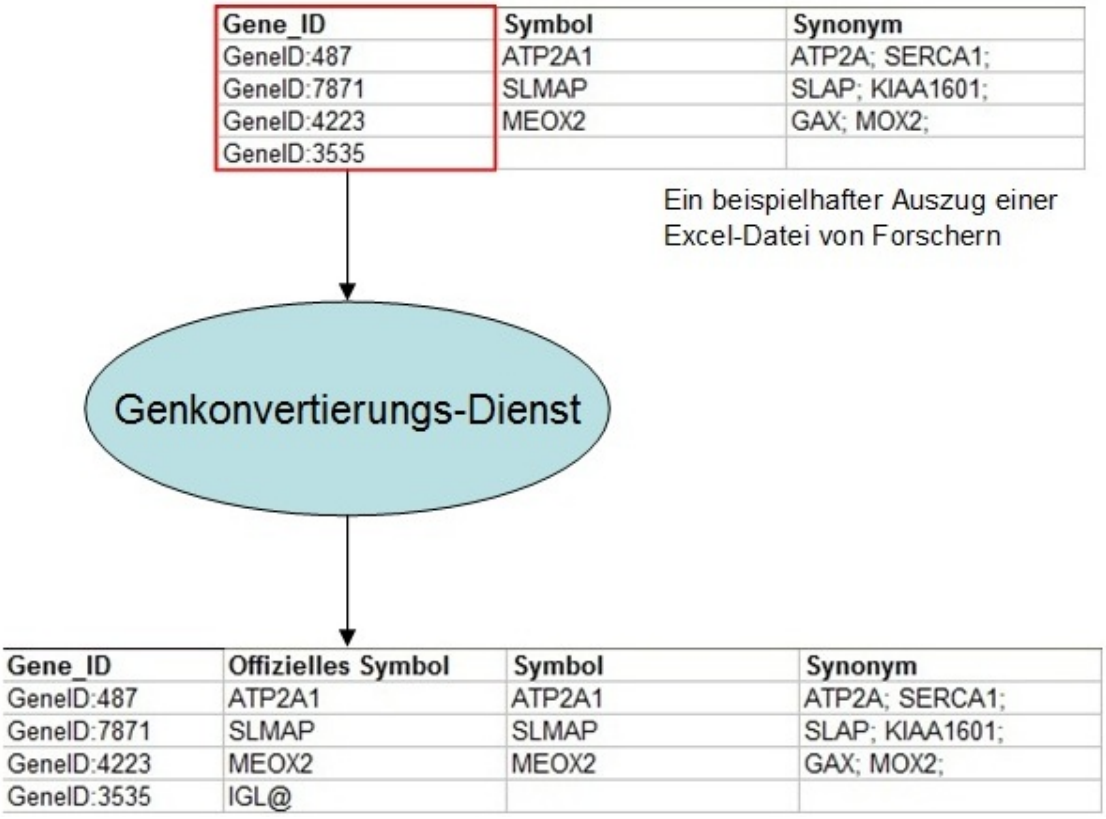


Abbildung 17:
Anwendungsfall 2: Die Daten sollen zentral gespeichert werden. Damit umfassende Analysen später möglich sind, wird zu jedem Datensatz das offizielle HGNC-Symbol mitgespeichert.

4.4.4 Vorgehen nach ETL-Prozess

Die HGNC-Daten werden von der HGNC-Download-Seite extrahiert und als Text-Datei lokal gespeichert. Sie sollen über ein Javaprogramm in eine eigene Datenbank geladen und durch einen Webservice für ‚pelican' zur Verfügung gestellt werden.
Von der HGNC-Webseite bis zum fertigen Webservice soll nach dem ETL-Prozess verfahren werden. Als ETL-Prozess bezeichnet man einen Prozess, der eine oder mehrere Datenbanken in eine Zieldatenbank transferiert. ETL steht für Extract, Transform, Load:

- **Extract** bedeutet in diesem Fall, dass die HGNC-Daten aus der HGNC-Datenbank extrahiert werden. Dies geschieht durch den Download der HGNC-Daten. Der Download wird in Form einer Text-Datei gespeichert.
- **Transform**: Die Text-Datei wird dann in das Javaprogramm geladen. Dort werden die Daten zerlegt und in einer anderen Form gespeichert. Sie werden transformiert.
- **Load**: Der letzte Prozess ist dann das Laden der Daten in eine lokale Datenbank (Rainardi 2008).

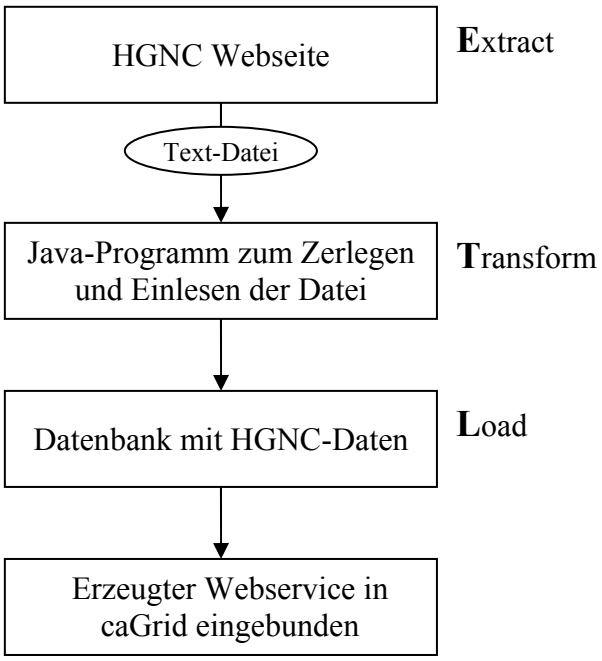


Abbildung 18:
Der ETL Prozess auf das Projekt dieser Arbeit abgebildet

4.4.5 Erstellung eines Datenbankschemas

Ein Datenbankschema legt fest, welche Daten in welcher Form und mit welchen Beziehungen zu anderen Daten gespeichert werden. Die HGNC-Datenbank stellt ihr Datenbankschema nicht zur Verfügung und es ist damit nicht bekannt. Für die lokale Datenbank, die als Grundlage für den Genkonvertierungs-Dienst dienen soll, gibt es damit zwei Möglichkeiten für ein eigenes Datenbankschema:

Erste Möglichkeit:
Die Datenbank bekommt ein einfaches Datenbankschema, welches direkt der Struktur der heruntergeladenen HGNC-Daten entspricht und somit aus nur einer Tabelle besteht (Abbildung 19). Die daraus folgende Datenbank würde für jede Zeile des HGNC-Downloads ein Tupel enthalten. Daraus würden dann Redundanzen in den Spalten mit dem anerkannten Symbol und dem Vorgängersymbol folgen, da ein Vorgängersymbol noch als anerkanntes Symbol mit dem Status ‚*withdrawn*' auftreten kann. Ein Beispiel: Symbol *A* ist Nachfolger von Symbol *B*. *B* ist also als Vorgängersymbol im Tupel von *A* enthalten. Aber *B* ist auch immer noch in der Spalte ‚Approved Symbol' zu fin-

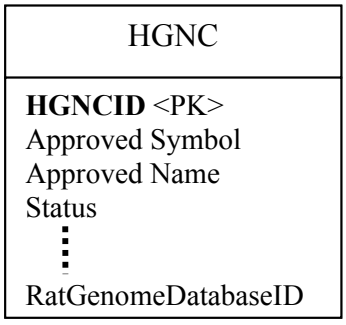


Abbildung 19:
Erste Möglichkeit für das Datenbankschema

den, das den Status ‚*withdrawn*' hat und gegebenenfalls als ‚B~withdrawn' dargestellt wird.
Redundanzen bedeuten, dass die Suche nach einem Symbol gegebenenfalls über zwei Spalten geschehen muss, um das aktuell anerkannte Symbol zu bekommen. Um es am eben genannten Beispiel zu zeigen: Ein Forscher sucht nach dem Gensymbol *B* in der Spalte ‚Approved Symbol'. Er bekommt als Ergebnis das offizielles Symbol *B* mit dem Status ‚withdrawn' und weiß dadurch, dass es eine aktuellere Bezeichnung für das Symbol gibt. Die HGNC-Daten besitzen nur einen Verweis auf das Vorgängersymbol, keinen der auf das Nachfolger-Symbol zeigt. Folglich muss im nächsten Schritt nach einem Tupel gesucht werden, welches das Symbol *B* als ‚Previous Symbol' besitzt. Dieses liefert Symbol *A*, das Nachfolger- und nun offizielle Symbol für das Gen.

Zweite Möglichkeit:
Die zweite Möglichkeit ist in Abbildung 20 dargestellt. Die Spalten ‚Approved Symbol', ‚Approved Name', ‚Status', ‚Previous Symbol' und ‚Previous Symbol Name' werden in die Tabelle ‚Symbols' gespeichert. Die Zusätze ‚Previous' und ‚Approved' werden gekürzt. Ein anerkanntes Symbol ist nun über den Status ‚*approved*' und ein Vorgängersymbol über den für den Genkonvertierungs-Dienst hinzugefügten Status ‚*previousSymbol*' gekennzeichnet. Alle anderen Daten werden in die Tabelle ‚HGNC' übernommen.
Über die Verbindungstabelle ‚HGNC_SYMBOL' kann weiterhin ein Symbol einer HGNC-ID zugeordnet werden. Diese Tabelle ist nötig, da es sich zwischen HGNC-ID und Symbol um eine so genannte ‚m zu n'- Beziehung handeln kann. ‚m zu n' bedeutet, dass zu einem HGNC-Datensatz mehrere Symbole zugeordnet werden können und umgekehrt, dass ein Symbol-Datensatz mehrere HGNC-Datensatz-Zuordnungen hat.
Das Datenbankschema entspricht eher der objektorientierten Programmierung als das Datenbankschema aus der ersten Möglichkeit. Darüber hinaus löst es die Redundanzen in den Symbolspalten auf. Aus diesen Gründen wurde dieses Datenbankschema für die HGNC-Datenbank dieser Arbeit ausgewählt.

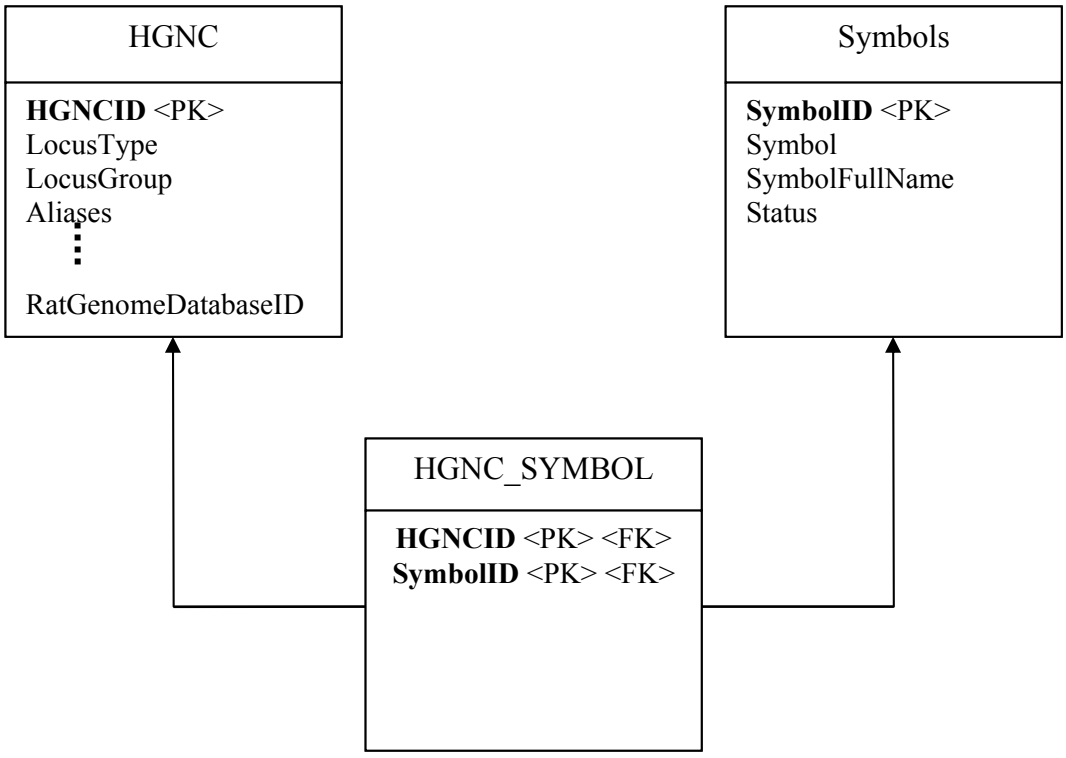


Abbildung 20:
Das Datenbankschema für die relationale HGNC-Datenbank dieser Arbeit. Es löst die Redundanzen im Symbol-Bereich auf.

4.4.6 Importieren der HGNC-Daten in die Datenbank

Das Fundament der Datenbank, das Datenbankschema, steht nun. Der Vorgang, ein Datenbankschema zu erzeugen, ist ein einmaliger Vorgang, solange die Attribute der Tabellen nicht geändert, erweitert oder gelöscht werden. In der Annahme, dass die HGNC die zur Verfügung gestellten Informationen vorerst nicht ändert, wurden die zugehörigen Tabellen mit ihren Attributen über ein MySQL-Verwaltungsprogramm erzeugt.

Für das Importieren der HGNC-Daten in die lokale Datenbank gibt es zwei Möglichkeiten:
Die eine Möglichkeit ist das Einlesen der HGNC-Daten über ein MySQL-Verwaltungsprogramm. Dieses bietet an, über eine selbsterstellte Importvorlage, Daten immer nach dem gleichen Schema einzulesen. Das Verwaltungsprogramm und die Importvorlage gehen dabei davon aus, dass die einzulesenden Daten einem strikten Schema folgen. Leider weichen einzelne Einträge der HGNC-Daten von einem erkennbaren Schema ab (Einsatz von Sonderzeichen, Auslassung bestimmter Informationen), sodass sie beim Importvorgang bearbeitet werden müssen. Jedoch ist eine Bearbeitung von Dateneinträgen vor dem Einlesen nur sehr schwer bzw. in manchen Fällen überhaupt nicht möglich.
Die andere Möglichkeit ist das Einlesen in die Datenbank über Schnittstellen, die von der Datenbank angeboten werden. Diese können von beliebigen anderen Programmen aufgerufen und genutzt werden. Aufgrund der vorliegenden Programmierkenntnisse würde der Aufruf vorhandener Schnittstellen durch ein Java-Programm erfolgen. Dies bringt unter anderem den Vorteil, dass die Daten vor dem Einlesen geändert und sortiert werden könnten. So können anerkannte Symbole, die den Anhang , ~withdrawn' haben, vor dem Einlesen geändert und in gekürzter Version gespeichert werden. Ein Beispiel: Ein Beispielsymbol ‚ABC~withdrawn' soll eingelesen werden. Der Anhang wird vor dem Einlesen gekürzt und das Symbol wird als ‚ABC' gespeichert.

Die Text-Datei, die von der HGNC zur Verfügung gestellt wird, ist eine Tabulator-Separated Values (TSV) Datei. In einer TSV sind einzelne Datenwerte bzw. Spalten durch einen Tabulator getrennt. Abbildung 21 zeigt beispielhaft einen Auszug aus der Text-Datei. Die Pfeile markieren die Tabulatoren.

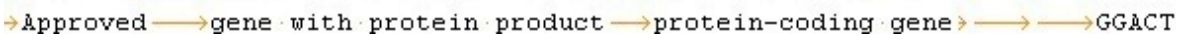


Abbildung 21:
Ein beispielhafter Auszug aus der Text-Datei von der HGNC Seite. Die Pfeile markieren einen Tabulator. Mehrere Pfeile hintereinander deuten auf leere Spalten hin.

Wie in Abbildung 22 zu sehen, ist der erste Schritt des Data Importers, die Textdatei in das Programm zu laden. Dort wird sie zeilenweise ausgelesen und verarbeitet. Jede Zeile wird zunächst komplett in einen String kopiert. Im Folgenden wird dieser String als Line-String bezeichnet. Die erste Zeile der Textdatei wird vom Data Importer ignoriert, da sie nur die Spaltennamen enthält.
Um den Inhalt der ersten Spalte eine Zeile zu bekommen, wird in dem Line-String nach dem ersten Tabulator mit Hilfe der indexOf-Methode gesucht. Die indexOf-Methode sucht das ihr übergebene Zeichen und gibt die Stelle des ersten Treffers zurück. Dadurch ist die Stelle bekannt, an der der erste Wert zu Ende ist. Der erste Wert, die HGNC-ID, wird nun über die substring-Methode auf einen anderen String namens *hgncID* kopiert. Die substring-Methode hat als Übergabeparameter zwei Werte: eine Anfangsstelle und eine Endstelle. Die Anfangsstelle ist beim Line-String der Wert 0, da der Wert am Anfang liegt. Die Endstelle ist der erste Tabulator, der durch die indexOf-Methode bekannt ist.
Nach dem Setzen des Wertes wird der Line-String um genau diesen Wert gekürzt. Das heißt, dass erneut die substring-Methode angewandt wird, diesmal jedoch von der Stelle des ersten Tabulators bis zum Ende des Line-Strings (siehe Abbildung 23).

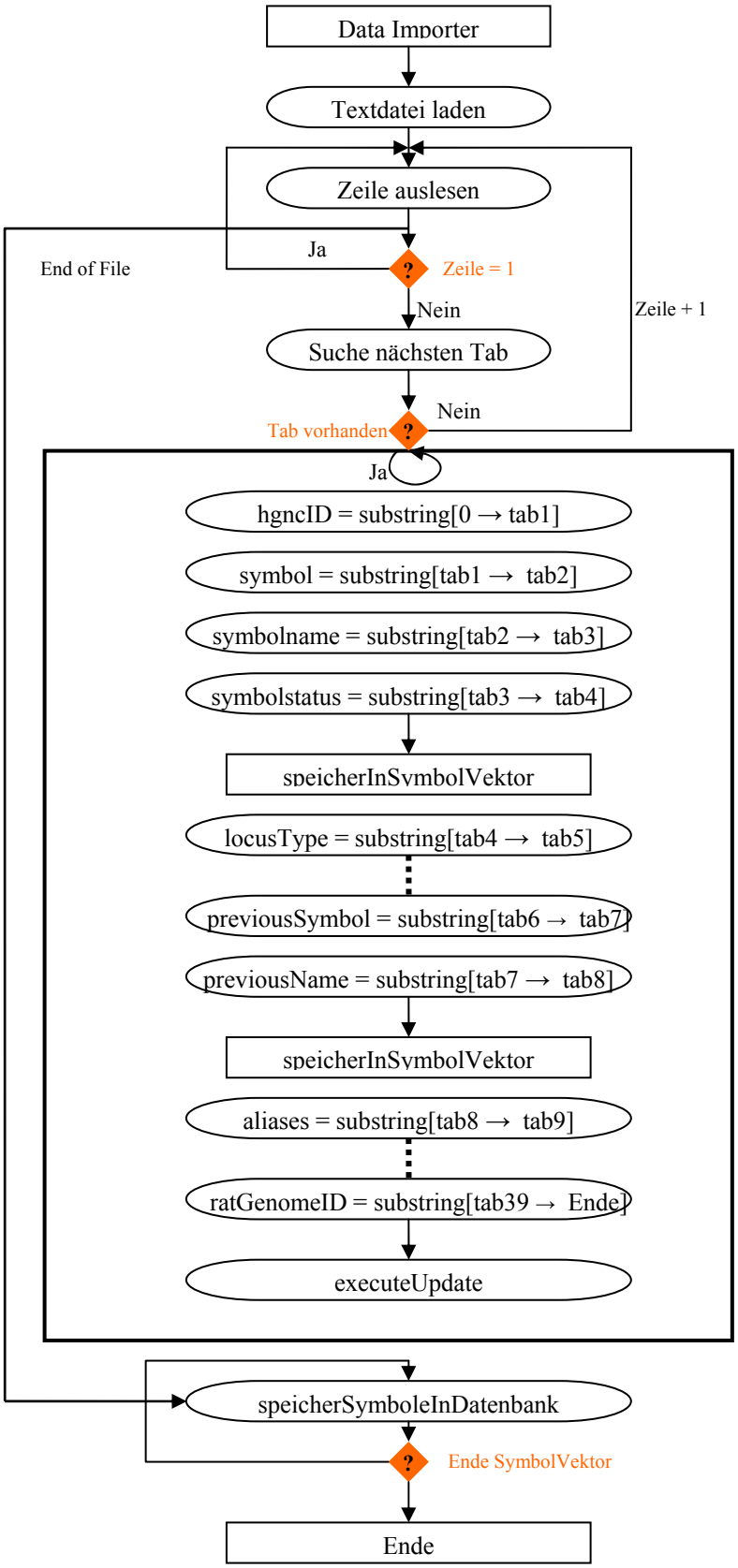


Abbildung 22:
Die Abbildung zeigt den Programmablaufplan des Java-Import Programms. Für das bessere Verständnis bekommen die Substring-Methoden die Tab Nummer übergeben anstatt immer wieder von Null anzufangen, so wie es im Text beschrieben ist.

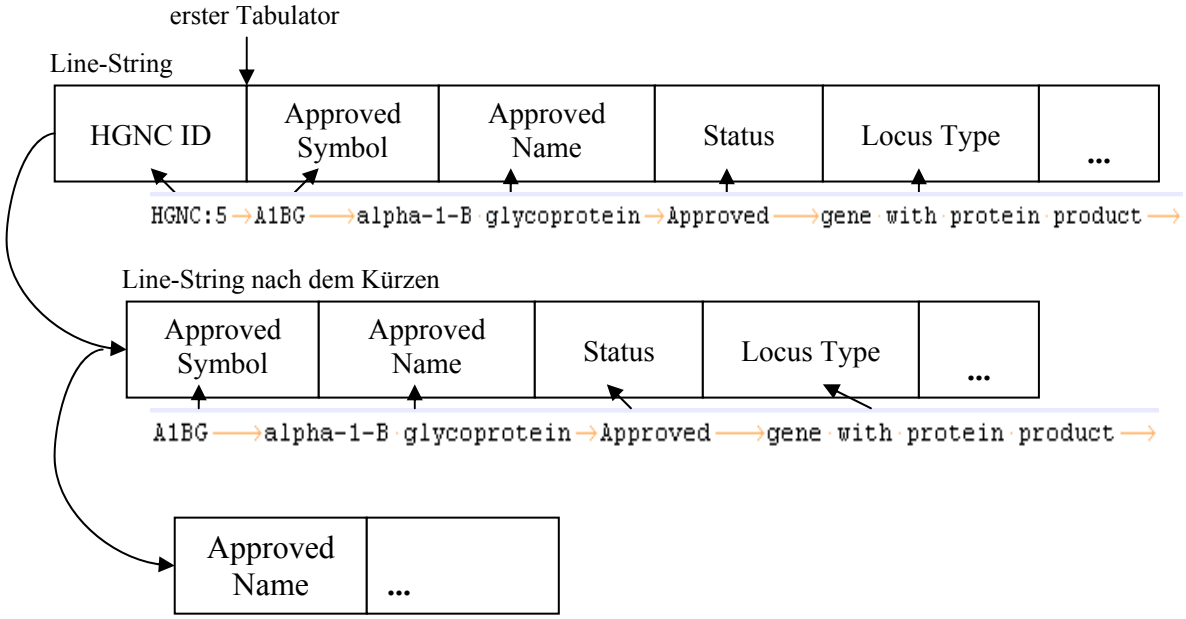


Abbildung 23:
Schematische Darstellung der Zerlegung desr Line-String

Danach wird erneut nach dem nächsten Tabulator gesucht und damit der Wert für die zweite Spalte gesetzt. Dies wird solange wiederholt, bis die ersten vier Spalten einen Wert haben. Nach der vierten Spalte, die Spalte für den Symbolstatus, wird ein Symbol-Objekt erzeugt. Ein Symbol-Objekt enthält eine Symbol ID, die durch einen Java-Generator erzeugt wird, das Gensymbol, den Gennamen, den Symbolstatus und einen Vector, der alle HGNC-IDs enthält, die auf dieses Symbol verweisen (siehe Abbildung 24).
Nach der Erzeugung des Objektes wird die Methode ‚*speicherInSymbolVector*' (siehe Abbildung 25) aufgerufen, die als Übergabeparameter das Symbol-Objekt enthält.
Da die Symbol-Tabelle der Datenbank keine doppelten Symbole enthalten soll und auch erst dann befüllt werden kann, wenn alle Beziehungen zwischen Symbol und HGNC-ID bekannt sind, werden die Symbol vorerst in einen Vector gespeichert, anstatt direkt in die Datenbank gespeichert zu werden. Die ‚*speicherInSymbolVector*'-Methode überprüft, ob ein Symbol bereits im Vector vorhanden ist und entscheidet dann, ob das Symbol gespeichert oder nur die HGNC-ID zu dem schon vorhanden Symbol hinzugefügt werden soll.
Die ‚*Approved Symbol*'-Spalte hat immer einen Eintrag, jedoch besitzt nicht jedes Tupel ein Vorgängersymbol. Dadurch können sogenannte ‚null'- Symbol-Attribute entstehen. Ein ‚null'-Symbol-Attribut bedeutet, dass die Spalte ‚*Previous Symbol*' des Tupels keinen Eintrag hat und damit das Attribut ‚*symbol*' eines Symbol-Objekts den Wert ‚*null*' hat. Auch wenn es kein Vorgängersymbol gibt, kann ein Vorgängersymbolname vorhanden sein. Ist dies der Fall, überprüft die ‚*speicherInSymbolVector*'-Methode, ob die Kombination aus dem Symbolattribut ‚*null*' und dem Symbolnamen schon vorhanden ist. Liegt diese Kombination das erste Mal vor, so wird das komplette Symbol-Objekt in den Vector aufgenommen. Ist sie schon einmal vorhanden, so wird zu dem HGNC Vector des entsprechenden ‚null'-Symbol-Attributs die HGNC-ID des neuen Symbols hinzugefügt.
Die gleiche Überprüfung wird durchgeführt, wenn das Symbol einen normalen Wert hat, dass heißt nicht ‚*null*' ist. Hier wird aber nur das Symbolattribut überprüft, nicht der Symbolname, da dieser, bei einem bereits vorhandenen Symbol, der gleiche sein sollte. Ist das Symbolattribut schon vorhanden, wird geprüft, ob das neue und das bereits vorhandene Symbol den gleichen Status haben.

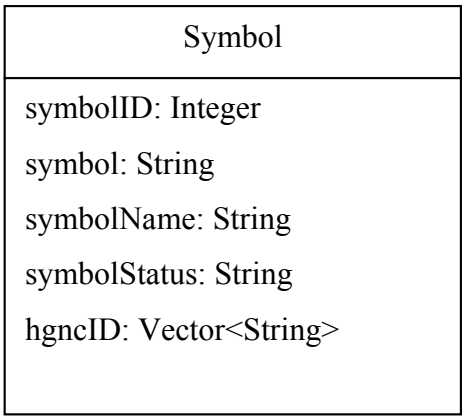


Abbildung 24:
Die Klasse Symbol mit ihren Attributen.

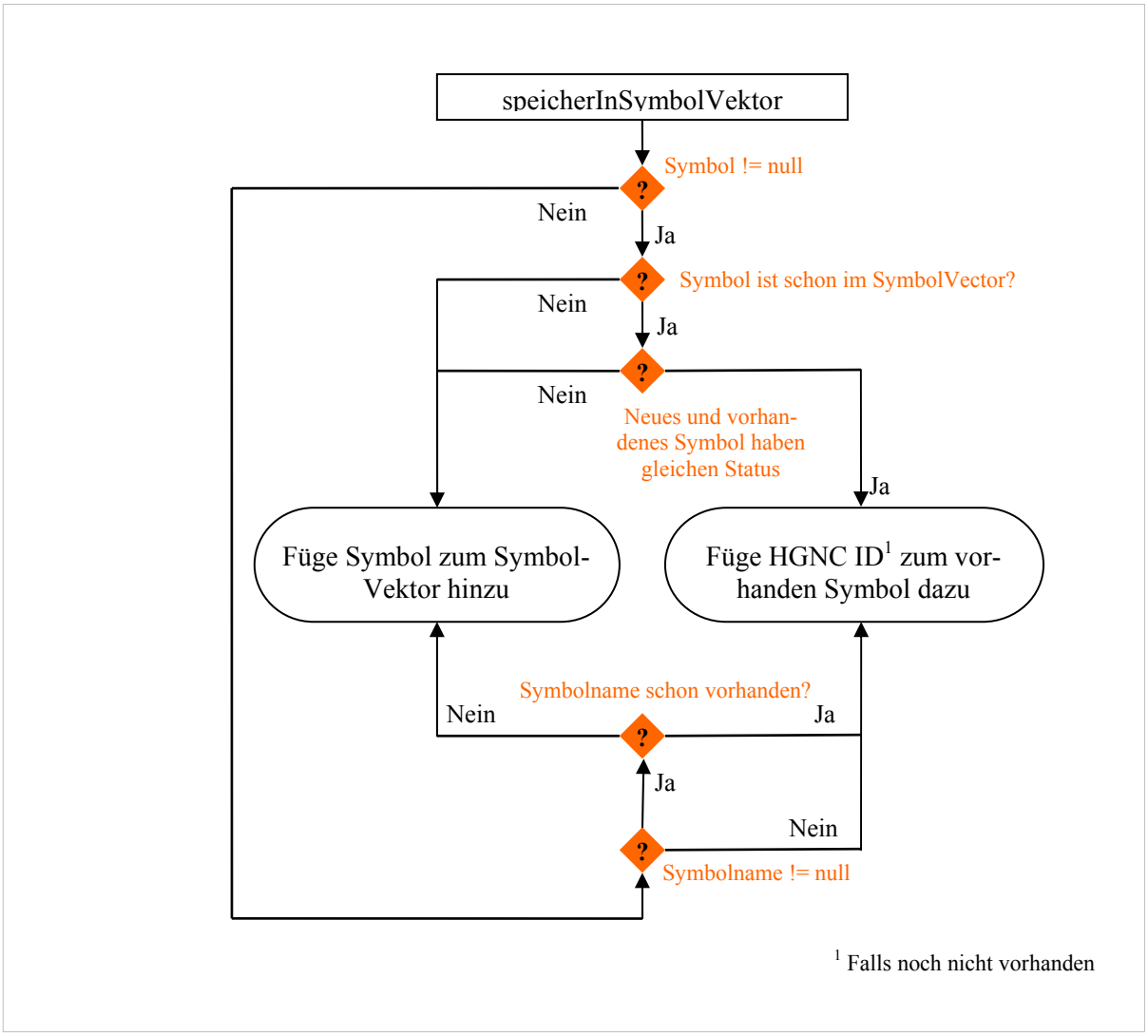


Abbildung 25:
Programmablaufplan der ‚*speicherInSymbolVector*' -Methode

Für die Datenbank wurde der zusätzliche Status ‚*previousSymbol*' eingeführt. Dieser soll bei späterer Abfrage eines Symbols darauf hinweisen, dass das angefragte Symbol ein Vorgängersymbol ist und damit ein neueres Symbol zur Verfügung steht.
Nach dem Durchlaufen der ‚*speicherInSymbolVector*'-Methode wird mit dem Zerlegen des Line-Strings fortgefahren und die Attribute für die HGNC-Tabelle gesetzt. Nach dem achten Tabulator (siehe Abbildung 22) sind alle Attribute für das Vorgängersymbol bekannt. Die ‚*speicherInSymbolVector*'-Methode wird erneut aufgerufen und durchgeführt. Diesmal wird als Status, wie eben erwähnt, der Wert ‚*previousSymbol*' übergeben. Nach dem Speichern des Vorgängersymbols bzw. dem Hinzufügen der HGNC-ID zu einem bereits vorhanden Symbol wird der Line-String weiter zerlegt. Alle Werte, die in die Tabelle HGNC gespeichert werden sollen (siehe Abbildung 20), werden, nachdem der Line-String komplett zerlegt und damit alle Werte der HGNC-Tabelle gesetzt wurden, in die Datenbank gespeichert.
Nun wird abgefragt, ob eine weitere Zeile im Textdokument vorhanden ist. Ist dies der Fall, so wird auch die Zeile wie eben beschrieben zerlegt. Das wird solange wiederholt, bist keine Zeile mehr im Textdokument vorhanden ist.

Ist die Textdatei komplett zerlegt, so sind alle vorhandenen Symbole in dem Symbol-Vector gespeichert. Diese werden dann einzeln ausgelesen und in die Symbol Tabelle gespeichert. Gleichzeitig werden die Verbindungen zwischen einem Symbol und den dazugehörigen HGNC IDs in die Verbindungstabelle gespeichert.

4.5 Erzeugen eines Webservice mit caBIG

Die erstellte Datenbank steht lokal zur Verfügung. Anfragen können unter anderem über ein MySQL-Verwaltungsprogramm oder über die Konsole gestellt werden. Diese Möglichkeiten sind jedoch nicht sehr elegant, da ein Benutzer SQL-Kenntnisse braucht, um eine Anfrage formulieren zu können. Darüber hinaus können noch nicht alle Forscher aus den verschiedenen Projekten des ‚SFB/TRR 77' auf die Datenbank zugreifen. Damit die Datenbank in ‚pelican' integriert werden kann und damit allen Forschern zur Verfügung steht, wird ein Webservice auf Basis der Datenbank erzeugt. Als Webservice kann dieser Dienst in eine SOA eingebunden und damit vollautomatisch in verschiedenen Prozessen genutzt werden. In ‚pelican' wird für diesen Dienst dann eine Weboberfläche bereitgestellt, die einfache Anfragen an die Datenbank ermöglicht.
Die Schritte einen Webservice zu erzeugen sind für jeden neuen Dienst gleich. Um die Entwicklung eines Webservices zu unterstützen und Fehler zu reduzieren, stellt caBIG das Tool caCore SDK, im Folgenden als SDK Code Generator bezeichnet, zur Verfügung.

Abbildung 26 zeigt den Prozess der Codegenerierung mit dem Tool caCore SDK. Da nur der erste Schritt direkt auf den Dienst dieser Arbeit angepasst werden muss, wird lediglich auf diesen intensiver eingegangen. Ziel dieses Arbeitsschritts ist es, ein UML (Unified Modeling Language) Modell des Dienstes zu erstellen. Dieses muss alle Objekte mit seinen Attributen, sowie die Beziehungen zwischen den Objekten detailliert darstellen. Die restlichen Schritte laufen halbwegs automatisch durch den SDK Code Generator ab. Der zweite Schritt des Code-Generierungsprozesses ist der Export des UML-Modells. Das exportierte Modell kann dann vom SDK Code Generator verwendet werden, um den Code zu generieren. Die Schritte drei bis sechs beschreiben den Ablauf des Code-Generierungsprozesses.

Da Besonderheiten beim Erstellen des UML Modells existieren und Fehler beim Umgang mit dem SDK möglich sind wird im Folgenden der Entwicklungs-Prozess genauer dargestellt.
Das Modell muss mit ArgoUML oder mit Enterprise Architect (EA) gebaut werden, damit der SDK Code Generator es richtig einlesen kann. Bei der Verwendung von ArgoUML sollte zum Zeitpunkt dieser Arbeit mit einer Version zwischen 0.24 bis 0.26 gearbeitet werden. Andere Versionen führen im Verlauf zu Fehlern. Der Aufbau mit EA wird hier nicht weiter betrachtet.

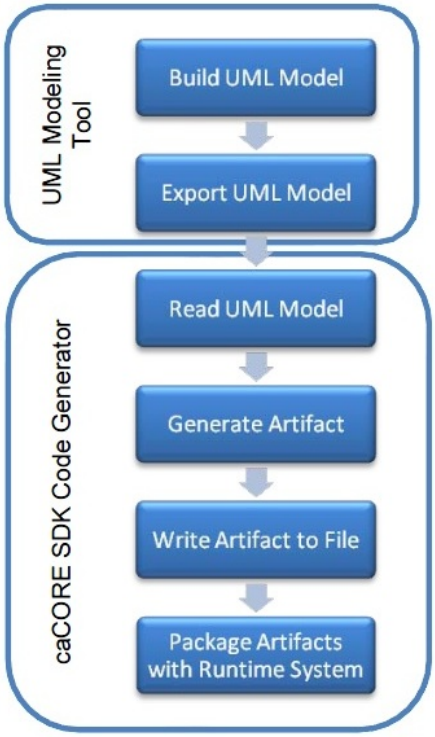


Abbildung 26:
Prozess der Codegenerierung[1]

Der erste Schritt zur Kompatibilität mit caBIG ist ein Informations-Modell von dem zu adaptierenden Dienst. Das Informations-Modell ist hier im Sinne eines UML-Klassenmodells, dass ein logisches (Objekt) Modell und ein Datenmodell (Tabelle) enthält
In dem Packet von caCore SDK 4.3[2] ist ein Template namens ‚*SDKArgoTemplate.uml*', das schon die wichtigsten Grundzüge für einen caBIG Service abbildet. Dies kann für die Erstellung des UML-Modells genutzt werden. Das Template enthält unter anderem das Paket ‚*Logical View*' mit den Unterpaketen ‚*Data Model*'(Datenmodell) und ‚*Logical Model*'(logisches Modell).
Das logische Modell definiert die Objekte, die von dem SDK Code Generator erstellt werden, sowie nach welchen Regeln mit diesen Objekten gearbeitet wird.[3] Das Datenmodell stellt die Datenbankstruktur in Form eines Diagramms dar.
Abbildung 28 zeigt das UML-Modell, das die Grundlage für die Code-Generierung des Webservices auf die Datenbank dieser Arbeit bildet. Die grünlich markierten Klassen stellen das logische Modell dar, die gelblichen das Datenmodell.
Das Datenmodell ist eine Abbildung des Datenbankschemas. Die Grundstruktur des UML-Modells für die Code-Generierung ist also die gleiche wie die des Datenbankschemas (siehe Abbildung 20). Die Klassen im Datenmodell müssen mit dem Stereotyp <table> und ihre Attribute mit <column> markiert werden. Über den Stereotyp kann der SDK Code Generator unterscheiden, ob es sich bei der Klasse um eine Klasse im Sinn von Java und damit vom logischen Modell handelt oder ob es eine Tabelle ist, die dem Datenmodell zuzuordnen ist. Der SDK Code Generator wird später mit den Attributnamen aus der Datenmodell-Klasse versuchen, auf die Datenbank zuzugreifen. Deshalb ist es wichtig, dass diese exakt die gleichen sind, wie sie in der Datenbank verwendet werden. Auch die Groß-/Kleinschreibung kann zu Problemen führen und muss deswegen beachtet werden. Auf die Verwendung von Leerzeichen in der Datenbank sollte generell verzichtet werden. Stattdessen können Wörter zum Beispiel mit einem Unterstrich getrennt werden.

[1] Quelle: https://wiki.nci.nih.gov/display/caCORE/2+-+caCORE+SDK+4.4+Code+Generation+Technical+Overview
[2] Download unter https://gforge.nci.nih.gov/frs/download.php/8616/SDK-4.3-bin.zip
[3] http://msdn.microsoft.com/de-de/library/aa266920%28v=vs.60%29.aspx

Der Primärschlüssel der Tabelle wird im ‚Abschnitt mit Operationen' einer Klasse definiert (siehe Abbildung 27). Dazu wird eine Operation namens *PK_Name* geschrieben. *Name* kann hier eine beliebige Bezeichnung sein. Damit es eindeutig ist, sollte der Name der Tabelle oder des Primärschlüssels verwendet werden, also zum Beispiel *PK_HGNCID*. Als Übergabeparameter der Operation wird der Primärschlüssel der Tabelle mit seinem Datentyp übergeben. Der Primärschlüssel der HGNC Tabelle ist hier als Beispiel mit der Methode *PK_HGNCID(HGNCID : VARCHAR2)* definiert (siehe auch Abbildung 28). Die Methode bekommt dann den Sterotyp <PK> zugewiesen.

Abbildung 27: Klassendiagramm. Darstellung einer Klasse eines UML-

Im Gegensatz zum Datenmodell, wird das logische Modell nicht mit Stereotypen annotiert. Da aber eine Umsetzung des Modells in Java-Programmcode vorgesehen ist, muss schon an dieser Stelle auf die ‚Java Naming Convention' geachtet werden. Wie in Abbildung 28 zu sehen, hat jede Tabelle außer der *HGNC_SYM BOL* Tabelle ein Abbild im logischen Modell. Die *HGNC_SYMBOL* Tabelle ist eine Hilfstabelle, um die ‚m zu n'-Verbindungen zwischen der HGNC- und der Symbol-Tabelle zu ermöglichen.

Der zweite Schritt ist die Semantische Integration. Jedes System hat seine eigene Struktur und Eigenschaften. Bei einer service-orientierten Architektur muss jedoch jedes System die gleiche Sprache sprechen, da sonst die Kommunikation unter den Systemen nicht funktioniert. Die eigene Systemsprache muss also auf die Standardsprache der SOA abgebildet werden. Die Standardsprache ist im Metadatenverzeichnis einer SOA gespeichert. Für die Überführung in die Standardsprache wird die Semantische Annotation genutzt. Im UML-Modell geschieht dies durch die Verwendung von so genannten ‚Tag Values' (Schlagworten).
Ein Beispiel: Wird bestimmt, dass unter dem Begriff ‚Bank' ein Geldinstitut verstanden werden soll, so wird dieser Begriff mit dem Schlagwort <Geldinstitut> versehen. So ist klar, dass damit ein Geldinstitut gemeint ist und nicht eine Parkbank zum Sitzen.
In dieser Arbeit wurde dieser Schritt nicht beachtet, da die Informationsplattform ‚pelican' bisher keine Verbindung zum caDSR, dem Metadatenverzeichnis der Plattform, hat.
Im dritten Schritt, werden das logische Modell und das Datenmodell aufeinander abgebildet, das so genannte Model Mapping. Dieser ist nicht in Abbildung 28 zu sehen. Model Mapping bedeutet, dass das logische Modell und das Datenmodell miteinander verbunden werden[1]. Auch hier wird mit Annotation durch Schlagworte gearbeitet. In ArgoUML werden diese Schlagworte in die *‚Eigenschaftswerte'* eingetragen. Dazu wird im Datenmodell ein Attribut ausgewählt, der Tab *‚Eigenschaftswerte'* geöffnet und als Kennzeichen *‚mapped-attributes [Model]'* bestimmt. Als Wert wird der äquivalente Wert aus dem logischen Modell angegeben. Dabei muss drauf geachtet werden, dass die gesamte Ordnerstruktur als Pfad angegeben wird.
Ein Beispiel: Die Spalte ‚*HGNCID*' aus der Tabelle *HGNC* hat mit dem Attribut ‚*id*' in der Klasse ‚*HGNCData*' ein äquivalent. Die Klasse ist in dem Paket ‚*start*' eingeordnet. Sollen diese zwei Werte nun miteinander Verbunden werden, so muss als ‚*mapped-attributes*' Wert in der Tabelle

[1] Zusätzliche Tools für das Mapping über eine GUI, wie caAdapter, werden hier nicht weiter betrachtet. Diese stehen nicht im Fokus und die Datenbank ist klein genug für manuellen Aufwand.

unter der Spalte *HGNCID* der Wert ‚*start.HGNCData.id*' eingetragen werden. Wichtig ist auch hier, auf die genaue Schreibweise zu achten, da schon ein Kleinbuchstabe statt eines Großbuchstabens zu Mapping - Fehlern führt.
Schlagworte sind sehr wichtig für den SDK Code Generator. Dieser sucht explizit nach Schlagworten um Artifakte zu generieren. Sind diese nicht oder fehlerhaft vorhanden, so kommt es zu Fehlern und der Generator bricht ab.
Sind alle Attribute zweier äquivalenter Klassen aufeinander abgebildet, so muss noch eine unidirektionale Abhängigkeits-Assoziation von der Datenmodell-Klasse zur logischen Modell-Klasse eingefügt werden. Ein Beispiel für so eine Assoziation ist zwischen der Klasse *HGNC* und *HGNCData* (siehe Abbildung 28). Diese Assoziation wird mit dem Stereotyp <DataSource> gekennzeichnet. Der Name der Assoziation kann beliebig gewählt werden, für eine bessere Übersicht empfiehlt es sich aber den Namen beider Klassen sowie die Richtung der Assoziation einzubringen, also zum Beispiel ‚*HGNC -> HGNCData*'. [1]
Weitere Informationen und Tipps zur Erstellung des UML-Modells oder zu Schlagwörtern sind als ‚Model Creation Guide' auf dem Wiki des NCI [2] zu finden.

Sind diese drei Schritte getan, kann das Informations- oder UML-Modell für die Generierung des Application Programming Interface (API) verwendet werden. Die API wird zum Austausch von Daten zwischen den Tools verwendet.
Der weitere Prozess der Code-Generierung ist sehr ausführlich im Wiki [3] beschreiben. Er wird hier nicht näher beschrieben, da er im Gegensatz zur UML-Modell-Erstellung keine wichtigen projektbetreffende Anpassungen hat. Wichtig ist, diesen Prozess immer mit einer bestehenden Verbindung zum Internet auszuführen, da einige Vorgänge sonst fehlschlagen.

Das Ergebnis der Code-Generierung ist ein System, das als Webservice aufgerufen und als solcher in das caGrid integriert werden kann. Die Generierung des Webservices muss nur einmal durchgeführt werden. Dieser hängt nur von dem Datenschema der Datenbank ab, nicht von den Dateninhalten an sich. Sollte jedoch die HGNC ein weiteres Attribut hinzufügen, ändern oder löschen, so muss ein neues Datenbankschema überlegt bzw. das alte Datenbankschema angepasst werden. Dies würde dann wiederum den Webservice beeinflussen, der deswegen neu erzeugt werden müsste.

[1] https://wiki.nci.nih.gov/display/caCORE/6.2+Creating+Classes+and+Tables+in+Argo+UML
[2] https://wiki.nci.nih.gov/display/caCORE/UML+Model+Creation+Guide+v1
[3] https://wiki.nci.nih.gov/display/caCORE/7+-+Create+and+Modify+Application+(Workflow+Step+4)
(letzter Zugriff jeweils 16.Februar 2011)

Abbildung 28:
Das UML-Modell, das für die Generierung des Webservice in dieser Arbeit verwendet wurde. Die Grün markierten Klassen stellen das Logical Model, die Orange markierten das Data Model dar.

4.6 Validierung des Dienstes im Vergleich mit GeneConnect

Für die Validierung des Dienstes wurden fünf beliebige Genidentifikationsdaten aus dem Methylierungsdatensatz, sowie fünf beliebige Genidentifikationsdaten aus dem aCGH-Datensatz genommen. Diese wurden in den Genkonvertierungs-Dienst und in GeneConnect eingegeben und konvertiert. Des Weiteren wurde die Richtigkeit des Ergebnisses durch eine Online-Suche auf der entsprechenden Gendatenbank-Webseite überprüft. Tabelle 3 zeigt die Ergebnisse der Konvertierungen.
Die Ergebnisse können vier Informationen enthalten: Die konvertierte Gen-ID, das anerkannte Symbol, den Genlocus und weitere Informationen zur Suchanfrage (siehe Abbildung 29). Die Abkürzung GKD, in der Spalte Korrektheit, steht für Genkonvertierungs-Dienst, GC für GeneConnect.
Die Suchanfragen 3, 4 und 5 der Methylierungsdaten, sowie 6, 9 und 10 der aCGH-Daten waren im jeweils im anderen Datensatz zu finden.
Leider war GeneConnect bis zum Ende dieser Arbeit online nicht mehr erreichbar, so dass keine Ergebnisse von GeneConnect eingetragen werden konnten.

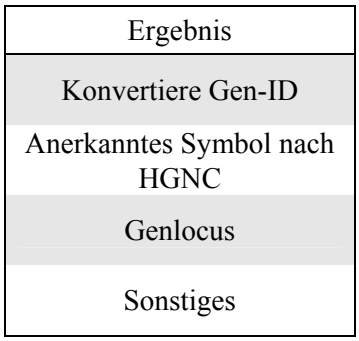


Abbildung 29:
Darstellung der Informationen in der jeweiligen Ergebnisspalte

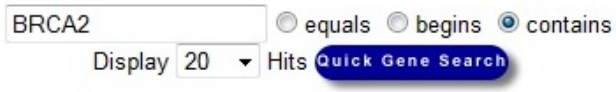
		Ergebnis des Genkonvertierungs-Diensts	Ergebnis von GeneConnect	Korrektheit	
Methylierungsdaten:				GKD	GC
Gene ID: 3754 Symbol: KCNF1	**1**	ENSG00000162975 KCNF1 2p25		✓	
Gene ID: 60468 Symbol: BACH2	**2**	ENSG00000112182 BACH2 6q15		✓	
Gene ID: 10395 Symbol: DLC1	**3**	ENSG00000164741 DLC1 8p22		✓	
Gene ID: 1776 DNASE1L3	**4**	ENSG00000163687 DNASE1L3 3p14.3		✓	
Gene ID: 64175 Symbol: LEPRE1	**5**	ENSG00000117385 LEPRE1 1p34.1		✓	
aCGH-Daten:					
ENSG00000145242 Symbol: EPHA5	**6**	Gene ID: 2044 EPHA5 4q13.1		✓	
ENSG00000104777 Symbol: ZNF90	**7**	Gene ID: 7643 ZNF90 19p12 ID war nur über Symbol auffindbar. Korrekte Ensembl ID: ENSG00000213988		✓	
ENSG00000183753 Symbol: BPY2C	**8**	Gene ID: 9083 BPY2 Yq11		✓	
ENSG00000152061 Symbol: RABGAP1L	**9**	Gene ID: 9910 RABGAP1L 1q24		✓	
ENSG00000108018 Symbol: SORCS1	**10**	Gene ID: 114815 SORCS1 10q23-q25		✓	

Tabelle 3
Ergebnistabelle der Validierung

5 Diskussion und Ausblick

Während in den Fachpublikationen parallel Gensymbole und Genlocus zur Identifikation und Kommunikation zwischen Forschern und Medizinern verwendet werden, wird in den Microarrays von klinischen Untersuchungen auch eine interne Gen-ID basierend auf der jeweils verwendeten Gendatenbank genutzt. Derzeit sind sowohl Gensymbole als auch Genlocus Änderungen durch neue Erkenntnisse weiterer DNA-Analysen unterworfen.
Im Verlauf dieser Diplomarbeit wurde ein Dienst entwickelt, der Genbezeichnungen aufeinander abbilden kann und mit dem Abbilden auf das offizielle Gensymbol des HGNC Standards der eindeutigen Genidentifikation dient. Für den Entwurf des Dienstes wurden verschiedene Gendatenbanken und Microarrays von klinischen Versuchen untersucht, und festgestellt, dass in diesen die Genidentifikation vor allem über eine datenbankspezifische Gen-ID läuft (Kapitel 2.6). Weitere Identifikationsmöglichkeiten sind über das Gensymbol oder den Genlocus möglich, wobei nach Abwägung der Vor- und Nachteile der Identifikationsmöglichkeiten das Gensymbol die beste Identifikationsmöglichkeit ist (Kapitel 4.1). Im Laufe der Arbeit stellte sich der Genlocus als häufiger unter Forschern verwendete Genidentifikation heraus.

Bei der Untersuchung der verschiedenen Gendatenbanken fiel auf, dass alle Datenbanken auf die HGNC-Datenbank verwiesen (Kapitel 2.6). Die HGNC Organisation bestrebt eine Standardisierung der Genidentifikation und ist daher zu unterstützen. Derzeit ist jedoch die HGNC kein verpflichtender Standard.
Die HGNC versucht die Gensymbole zu vereinheitlichen und jedem Gen ein offizielles Gensymbol zu zuweisen. Diese Arbeit ermöglicht eine eindeutige Genidentifikation über die Verwendung des offiziellen Symbols der HGNC.
Wie gerade erwähnt, stellte sich heraus, dass der Genlocus die gängigere Methode unter Forschern ist, Gene zu identifizieren. Die HGNC-Datenbank ist dennoch eine geeignete Referenzdatenbank, da sie die Flexibilität bietet, neben dem Gensymbol auch den Genlocus bzw. gegebenenfalls andere Genidentifikationen anzugeben. Die HGNC hat mittlerweile sogar ihre Datenbankabfrage dahingehend optimiert, dass der Genlocus beim Suchergebnis angezeigt wird (siehe Abbildung 30).
Bei der Analyse der HGNC-Daten wurde festgestellt, dass nur zwei anerkannte Symbole keinen Genlocus angegeben haben. Diese Zahl ist im Hinblick auf ungefähr 30.000 Einträge mit dem Status ‚anerkannt' vernachlässigbar. Da der Webservice eine Abfrage auf jede Spalte der Datenbank erlaubt, kann die Identifikation jederzeit vom Gensymbol auf den Genlocus umgestellt werden.



Advanced Search

Approved Symbol	Approved Name	Location	Best Match
BRCA2	breast cancer 2, early onset	13q12-q13	**Approved Symbol**: BRCA2
BCCIP	BRCA2 and CDKN1A interacting protein	10q26.2	**Approved Name**: BRCA2 and CDKN1A interacting protein
BRCC3	BRCA1/BRCA2-containing complex, subunit 3	Xq28	**Approved Name**: BRCA1/BRCA2-containing complex, subunit 3
FACD~withdrawn	symbol withdrawn, see BRCA2		**Approved Name**: symbol withdrawn, see BRCA2

Abbildung 30:
Ausschnitt der Ergebnisanzeige einer Gensuche auf der HGNC-Webseite

Um weitere Möglichkeiten zur eindeutigen Identifikation von Genen zu erhalten, wurden die biomedizinische Standards HL7, CDISC und MIAME betrachtet. Auch in den biomedizinischen Standards gibt es keine einheitliche Methode der Genidentifikation. Die vom Standard HL7 verwendete Terminologie LOINC identifiziert Gene über den Gennamen. CDISC gibt keine Semantik für einzelne Variablen vor, empfiehlt jedoch eine Terminologie, wie zum Beispiel LOINC, zu verwenden. MIAME schreibt weder eine bestimmte Genidentifikationsmethode noch eine Terminologie vor (Kapitel 4.2). Die betrachteten Standards beschreiben nur den Inhalt und die Struktur, wie eine (Gen-) Datenübertragung von einer Institution zur einer anderen ohne Komplikationen durchgeführt werden kann. Mit welcher Semantik der Inhalt beschreiben wird, legen sie nicht fest.

Diese Diplomarbeit ist Teil einer Informationsplattform, die mit den Tools von caBIG arbeitet. caBIG bietet unter anderem das Tool GeneConnect an, das Gen IDs verschiedener Gendatenbanken aufeinander abbilden kann. Eine eindeutige Genidentifikation ist dadurch nicht gegeben, sondern auf eine Konvertierungsmöglichkeit beschränkt. Darüber hinaus ist es bei GeneConnect nicht möglich weiteren Informationen zu erhalten, wie zum Beispiel das Gensymbol oder den Genlocus, die die Forscher vorzugsweise zur Genidentifikation nutzen. Das bedeutet, bei einer Konvertierung der Gen-IDs durch GeneConnect, müsste ein weiterer Services folgen, der die zusätzliche Informationen, wie Genname, Gensymbol oder Genlocus, zu einer Gen-ID liefert. Diesen Service stellt das caGrid von ‚pelican' derzeit nicht zur Verfügung (siehe Kapitel 4.3.3).
Eine Wiederverwendung von GeneConnect für den Dienst dieser Arbeit wäre möglich gewesen. Da die Daten der HGNC aber alle benötigten Gen-IDs schon beinhalten, ist eine direkte Nutzung der HGNC-Daten effizienter als die Einbindung von GeneConnect.

caBIG basiert auf dem Konzept einer service-orientierten Architektur. Damit das caGrid auf einen externen Dienst zugreifen kann, muss der Dienst online verfügbar und im Verzeichnisdienst registriert sein. Außerdem muss der Dienstnutzer eine Berechtigung besitzen, diesen Dienst zu verwenden (Kapitel 2.4). In dieser Arbeit wurde über das caBIG-Tool caCore SDK, ein Webservice auf Basis einer lokalen Datenbank erzeugt. Die dazu benötigten Schritte zur UML-Erstellung und zur Erzeugung des Services sind sehr detailliert im caBIG-Wiki beschrieben.

Eine eindeutige Genidentifikation für die Informationsplattform des ‚SFB/TRR 77' kann durch die Konvertierung einer Gen-ID auf das anerkannte Symbol der HGNC erfolgen oder auf eine andere Gen-ID (Kapitel 4.4.1). Diese Methode wird von einem Genkonvertierungs-Dienst realisiert, der in eine SOA eingebaut werden kann.

Damit der Genkonvertierungs-Dienst in die Informationsplattform pelican eingebunden werden kann, muss ihr eine service-orientierte Architektur zu Grunde liegen. Dies ist durch die Aufgabenstellung gegeben.

Der Genkonvertierungs-Dienst ist auf Basis einer MySQL-Datenbank aufgebaut, die mit den Daten aus der HGNC-Datenbank befüllt ist. Die HGNC-Daten werden durch ein Java-Programm eingelesen. Die Programmiersprache Java eignet sich sehr gut aufgrund ihrer Objektorientierung, ihrer umfangreichen Klassenbibliothek und der Unterstützung mit JDBC.
Das für die Datenbank dieser Arbeit erstellte Datenbankschema (siehe Abbildung 20) für die HGNC-Daten löst Redundanzen auf, die innerhalb der verschiedenen Symbol-Spalten auftreten können. Die Überprüfung, ob jedes Symbol nur einmal in der Symboltabelle vorkommt, wird durch Java vorgenommen. Ist ein Symbol bereits vorhanden, so wird nur die HGNC ID zu dem bereits vorhanden Symbol gespeichert. So eine Überprüfung wäre auch durch MySQL möglich gewesen, in dem man die Symbols Tabelle als ‚*Unique*' deklariert. Jedoch können in dieser Arbeit mehrere ‚*null*'-Symbole vorkommen, die unterschiedliche Gennamen besitzen. Mit der Deklaration ‚*Unique*' könnte diese Unterscheidung nicht vorgenommen werden. Zudem wäre der Aufwand, zu einem bereits vorhandenen Symbol eine HGNC ID hinzuzufügen, etwas komplizierter als in Java, da das Symbol erst über eine Datenbankabfrage gesucht werden müsste.

Die Aliases-Spalte wurde nicht mit in die Symboltabelle aufgenommen, obwohl sie eigentlich auch Symbole enthält. Jedoch fiel bei der Betrachtung der Daten auf, dass nicht nur Gen-Symbol in dieser Spalte vorhanden sind, sondern auch Einträge wie zum Beispiel: ein Datum, eine Gen-ID, der Genlocus oder Buchstabenfolgen, die nicht der Gen-Symbol-Konvention folgen (lateinische Großbuchstaben mit arabischen Zahlen).

Die HGNC-Daten werden regelmäßig aktualisiert. Das bedeutet, dass auch die Datenbank dieser Arbeit regelmäßig auf den neusten Stand gebracht werden muss. Die HGNC bietet ihre Aktualisierung nur als kompletten Datensatz an, es gibt keine Option nur die HGNC-Datensätze zu bekommen, die geänderten oder aktualisierten wurden.
Bei den HGNC-Daten handelt es sich um standardisierte Inhalte. Deswegen hat es keinen Sinn, die Daten in der Datenbank zu ändern oder zusätzliche Anmerkungen zu den Daten hinzuzufügen. Aus diesem Grund wurde entschieden, dass bei einer Aktualisierung der HGNC-Daten alle Daten aus der lokalen Datenbank gelöscht und neu eingelesen werden. Dabei musst lediglich das Java-Programm zum Einlesen der Daten aufgerufen werden. Die Änderungen bzw. Aktualisierung der Daten haben keinen Einfluss auf das Datenbankschema oder auf den Webservice, diese müssen also nicht neu erzeugt werden.
Die Implementierung einer Update-Methode für die Datenbank wäre ebenso möglich gewesen. Diese würde sich jedoch nur lohnen, wenn die HGNC einen Datensatz zur Verfügung stellen würde, der nur die aktualisierten Daten enthält. Dann wäre der Abgleich der Daten deutlich schneller als das Neu-Einlesen aller HGNC-Daten.
Sollte die HGNC ihr Datenbankschema ändern, also zum Beispiel ein neues Attribut einfügen oder ein bereits vorhandenes entfernen, wäre das Neu-Einlesen der HGNC-Daten nicht mehr möglich. Dazu müsste zunächst das neue Attribut in das Datenbankschema dieser Arbeit eingefügt bzw. ein neues Datenbankschema erstellt werden.

Die Validierung des Dienstes wurde im Vergleich zu GeneConnect durchgeführt. Durch jeweils fünf zufällig ausgewählte Datensätze aus den Methylierungs- und aCGH-Daten wurde nachgewiesen, dass der Genkonvertierungs-Dienst korrekt arbeitet.
Der Genkonvertierungs-Dienst konnte alle Gen-IDs korrekt auf die jeweils andere Gen-ID und auf das offizielle Gensymbol abbilden. Suchanfrage 7 (siehe Tabelle 3) enthielt als Eingabe eine veraltete Ensembl-ID. Der Genkonvertierungs-Dienst konnte diese Suchanfrage dennoch über das Gensymbol auflösen. Alle Ensembl IDs des aCGH-Datensatzes waren nur als ‚*mapped data*' in den HGNC-Daten vorhanden, also als nicht manuell geprüfte IDs. Dies zeigt, dass Spalten, die als ‚*mapped data*' gekennzeichnet sind, durchaus verwendbar sind.
Bedauerlicherweise war GeneConnect bis zum Ende dieser Arbeit nicht mehr verfügbar, so dass ein direkter Vergleich nicht möglich war. Als Ergebnis der Gen-ID-Konvertierung von GeneConnect wäre eine korrekte Konvertierung erwartet worden. Das Ergebnis wäre allein durch eine Gen-ID dargestellt worden, ohne weitere Informationen zu Gensymbol oder Genlocus. Wahrscheinlich hätte GeneConnect die Suchanfrage 7 nicht auflösen können, da es keine Abbildung in der Entrez Gene Datenbank gibt.
Als Stichprobe wurden zehn zufällig ausgewählte Datensätze gewählt. Sie ist zwar im Vergleich zur Gesamtmenge an HGNC-Daten (ca. 34.000) klein, genügt aber, um eine Aussage über die Software-Qualität zu machen. Die Validität der HGNC-Daten selbst sollte durch diesen Test nicht bestätigt werden.

6 Literaturverzeichnis

Ashurst, J.L., 2004. The Vertebrate Genome Annotation (Vega) database. *Nucleic Acids Research*, 33(Database issue), D459-D465.

Blobel, B. u. a., 2009. HL7 Kommunikationsstandards für das Gesundheitswesen - Ein Überblick. Verfügbar unter: http://www.hl7.de/download/InfobroschuereHL7.pdf.

Brazma, A. u. a., 2001. Minimum information about a microarray experiment (MIAME)-toward standards for microarray data. *Nature Genetics*, 29(4), 365-371.

Kenneth Buetow & Andrew C. von Eschenbach, Cancer Informatics Vision: caBIG™. *Cancer Informatics*, 2, 22-24.

CDISC, CDISC (Clinical Data Interchange Standards Consortium) Brochure. Verfügbar unter: http://www.cdisc.org/stuff/contentmgr/files/0/8e8cbe6e3af177832ca3a86a44ac5aa0/misc/new_cdisc_bro.pdf.

Denz, T., Meier, R. & Rempfler, M., 2010. *DNA-Microarrays*, Muttenz. Verfügbar unter: http://www.dusseiller.ch/mis_wiki/images/b/b4/Fallstudie_DNA-Microarrays.pdf.

Drepper, J. & Semler, S.C., Standardisierung klinischer Forschungsdaten auf Basis von CDISC als Voraussetzung für eine bessere Integration von Forschung und Versorgung. *Telemedizinführer Deutschland*, (2006). Verfügbar unter: http://www.telemedizinführer.de/free/2006/drepper_362_366.pdf.

Finger, P., 2009. *SOA und WebServices*, Berlin ; Heidelberg: Springer.

Goodsell, D., 2010. *Wie Zellen funktionieren : Wirtschaft und Produktion in der molekularen Welt* 2. Aufl., Heidelberg: Spektrum Akad. Verl.

Huch, R., 2003. *Mensch, Körper, Krankheit.* 4. Aufl., München ; Jena: Urban und Fischer.

Hussain, H., 2010. *Primary carcinomas of the liver*, Cambridge UK ; New York: Cambridge University Press.

Maglott, D. u. a., 2010. Entrez Gene: gene-centered information at NCBI. *Nucleic Acids Research*, 39(Database), D52-D57.

McDonald, C. u. a. hrsg., 2009. Logical Observation Identifiers Names and Codes (LOINC®). Verfügbar unter: http://loinc.org/international/german/german_loinc_user_guide.pdf.

Meadows, B. u. a., 2001. The Common Data Element Dictionary-A Standard Nomenclature for the Reporting of Phase 3 Cancer Clinical Trial Data. In 4th IEEE Symposium on Computer-Based Medical Systems (CMBS'01).

Melzer, I., 2008. *Service-orientierte Architekturen mit Web Services : Konzepte - Standards - Praxis* 3. Aufl., Heidelberg: Spektrum Akad. Verl.

Neis-Beeckmann, P., 2009. *Molekularbiologie für Dummies : [der Stoff, aus dem das Leben ist]* 1. Aufl., Weinheim: Wiley-VCH-Verl.

Passarge, E., 2008. *Taschenatlas Humangenetik* 3. Aufl., Stuttgart: Thieme.

Rahmann, P.D.S., 2009. Einführung in die Angewandte Bioinformatik: Analyse und Design von DNA Microarrays. Verfügbar unter: http://ls11-www.cs.uni-dortmund.de/people/rahmann/teaching/ss2009/angebio/Microarrays.pdf.

Rainardi, V., 2008. *Building a data warehouse with examples in SQL Server*, Berkeley, CA :: Apress,

Steinmetz, A., 2005. MIAME - Standard für Microarraydaten. Verfügbar unter: http://ibios.dkfz.de/ibios_old/lectures/seminar_ss05/kabbe/MIAME.pdf.

Ullenboom, C., 2009. *Java ist auch eine Insel : Programmieren mit der Java Platform, Standard Edition 6 [das umfassende Handbuch DVD-ROM inkl. Openbook-Bibliothek mit über 4000 Seiten 300 Aufgaben und Lösungen* 8. Aufl., Bonn: Galileo Press.

7 Abkürzungsverzeichnis

aCGH ..... *Array comparative genomic hybridization*
API ..... *Application Programming Interface*
BRCA2 ..... *breast cancer 2, early onset*
caBIG ..... *cancer Biomedical Informatics Grid*
caDSR ..... *Cancer Data Standards Registry and Repository*
CCDS ..... *Consensus coding Sequences*
CDA ..... *Clinical Document Architecture*
CDE ..... *Common Data Element*
CDISC ..... *Clincal Data Interchange Standard Consortium*
cDNA ..... *complementary DNA*
CRF ..... *Case Report Form*
DFG ..... *Deutsche Forschungsgemeinschaft*
DKFZ ..... *Deutsches Krebsforschungszentrum*
DNA ..... *Desoxyribonukleinsäure*
EBI ..... *European Bioinformatics Institute*
EC ..... *Enzyme Commission*
EMBL ..... *European Molecular Biology Laborator*
FACD ..... *Fanconi anemia, complementation group D1*
GUI ..... *Graphical User Interface*
HCC ..... *Hepatozelluläres Karzinom*
HGNC ..... *HUGO Nomenclature Committee*
HL7 ..... *Health Level 7*
HUGO ..... *Human Genome Organisation*
ICD ..... *International classification of diseases*
IEC ..... *International Electrotechnical Commission*
ISO ..... *International Organisation for Standardization*
JDBC ..... *Java Database Connectivity*
LAB ..... *Laboratory Data Model*
LOINC ..... *Logical Observation Identifier Names and Codes*
MDA ..... *Model-Driven Architectur*
MGI ..... *Mouse Genome Informatics*
MIAME ..... *Minimum Information About A Microarray Experiment*
mRNA ..... *messenger Ribonukleinsäure*
MySQL ..... *My Structured Query Language*
NCBI ..... *National Center for Biotechnology Information*
NCI ..... *National Cancer Institute*
ODM ..... *Operational Data Model*
OMIM ..... *Online Mendelian Inheritance*
PCR ..... *Polymerase-Ketten-Reaktion*
pelican ..... *platform enhancing livercancer networked research*
PR ..... *Protocol Representation*
RefSeq ..... *Reference Sequence*
RGD ..... *Rat Genome Database*

SDK ..... *Software Development Kit*
SDTM ..... *Study Data Tabulation Model*
SFB/TRR ..... *Sonderforschungsbreich (Transregio)*
Snomed-CT ..... *Systematized Nomenclature of Medicine - Clinical Terms*
SOA ..... *Service-orientierte Architektur*
STIP1 ..... *stress-induced-phosphoprotein 1*
TSV ..... *Tabulator-Separated Values*
UCUM ..... *Unified Code for Units of Measure*
UML ..... *Unified Modeling Language*
VEGA ..... *Vertebrate Genome Annotation*
WTSI ..... *Wellcome Trust Sanger Institute*

8 Abbildungsverzeichnis

9 Tabellenverzeichnis

Anhang

Die folgende Tabelle beschreibt die Datenfelder der Datenbank.

Datenbankfeld	Bedeutung
HGNC ID	Eindeutige interne ID, die von dem HGNC zur Verfügung gestellt wird.
Approved Symbol	Das Standard Symbol des Genes, welches von dem HGNC anerkannt wurde.
Approved Name	Der Standard Name des Gens.
Status	Der Status zeigt an, wie ein Gen klassifiziert ist: • Approved: diese Gene haben ein anerkanntes HGNC-Symbol • Entry withdrawn: diese Gene waren einmal anerkannt und werden jetzt nicht mehr als existent angesehen • Symbol withdrawn: ein früher anerkannter Datensatz, welcher nun mit einem anderen Datensatz zusammengefügt wurde
Locus Type	Spezifiziert den Typ des Ortes durch verschiedene Einträge. Z.B. g*ene with protein product* , was auf protein-codierende Gene hinweist. Weitere Typen findet man auf: http://www.genenames.org/data/gdlw_columndef.html
Previous Symbols	Symbole für dieses Gen, die vorher anerkannt durch das HGNC waren.
Previous Names	Gennamen für dieses Gen, die vorher anerkannt waren.
Aliases	Andere Symbole, die benutzt werden, um auf dieses Gen zu verweisen.
Name Aliases	Andere Gennamen, die benutzt werden, um auf dieses Gen zu verweisen
Chromosome	Beschreibt den Genlocus
Date Approved	Das Datum, an dem das Symbol und der Name von dem HGNC anerkannt wurden.
Date Modified	Sollte etwas bei dem Eintrag für dieses Gen geändert worden sein, so steht hier das Datum der Änderung.
Date Symbol Changed	Das Datum, an dem das Genymbol das letzte Mal von einem vorher anerkannten Symbol durch das HGNC geändert wurde.
Date Name Changed	Das Datum, an dem der Gennamen das letzte Mal von einem vorher anerkannten Namen durch das HGNC geändert wurde
Accession Number	Von dem HGNC ausgewählte Zugangsnummern für jeden Eintrag.

Enzyme ID	Enzymeinträge sind mit Enzyme Commission (EC) Nummern verbunden, die auf die hierarchisch funktionale Klassen weisen, zu denen sie gehören
Entrez Gene ID	Die ID, mit der das Gen in der Entrez Datenbank gefunden werden kann.
CCDS ID	Die ID, mit der das Gen in dem Consensus CDS (CCDS) Projekt gefunden werden kann.
VEGA ID	Die Gen ID von VEGA.
Locus Specific Database	Eine Liste von Links zu Datenbanken oder Datenbankeinträgen, die relevant für das Gen sind.
Mouse Genome Database ID	Die Gen ID von der Mouse Genome Informatics (MGI) Datenbank.
Specific Database Links	Diese Spalte enthält Links zu speziellen Datenbanken, die ein besonderes Interesse an diesem Gen bzw. Symbol haben.
Ensembl Gene ID	Die Gen ID von der Ensembl Datenbank. Diese wurde manuell eingetragen.
Specialist Database IDs	Links zu betreffenden Datenbanken.
Pubmed IDs	IDs, mit denen man in der PubMed Datenbank von NCBI veröffentlichte Artikel findet, die relevant für den Eintrag sind.
RefSeq ID	Die Reference Sequence (RefSeq) ID für dieses Gen, welche von der NCBI zur Verfügung gestellt wird.
Gene Family Tag	Ein Tag, der verwendet wird um eine Genfamilie oder eine Gruppe zu der das Gen gehört zu kennzeichnen, entsprechend der Sequenzähnlichkeit oder der Information von Publikationen bzw. andere Datenbanken.
Mapped Data	Wenn ein Feld mit *mapped data* markiert ist, bedeutet das, dass die ID von externen Quellen kommt und von dem HGNC nicht explizit manuell geprüft wurde. Diese Daten sollten mit Vorsicht verwendet werden.
Mouse Genome Database ID (mapped data)	Die Gen ID von der *Mouse Genome Informatics* (MGI) Datenbank. http://www.informatics.jax.org/
Rat Genome Database ID (mapped data)	Die Gen ID von *der Rat Genome Database* (RGD) http://rgd.mcw.edu/
GDB ID (mapped data) (depreciated)	GDB war eine Quelle von qualitätsvollen Mapping Daten, welche sowohl online als auch über zahlreiche gedruckte Veröffentlichungen verfügbar gemacht wurden. Was GDB von anderen biologischen Datenbanken unterschied, war der Einsatz von Weltklasse Führungspersonen in der menschlichen Genetik, die die Daten verwaltet haben. Um einen hohen Grad von Qualität zu garantieren, wurden die Daten innerhalb von GDB ei-

	nem Begutachtungsprozess unterzogen, ähnlich wie bei einer traditionellen Publikation
Entrez Gene ID (mapped data)	Die ID, mit der das Gen in der Entrez Datenbank gefunden werden kann.
OMIM ID (mapped data)	Von der *Online Mendelian Inheritance in Man* (OMIM) Datenbank bereitgestellte ID auf der NCBI Homepage. Diese Datenbank wird als ein Katalog von menschlichen Genen und genetischen Erkrankungen beschrieben, der textliche Informationen und Links zu MEDLINE, sowie Sequenzeinträge in der Entrez Datenbank beinhaltet.
RefSeq ID (mapped data)	Die RefSeq ID für dieses Gen.
UniProt ID (mapped data)	Die Gen ID bei UniProt, welche von dem European Bioinformatics Institute (EBI) bereitgestellt wird.
Ensembl Gene ID (mapped data)	Die Ensembl ID wurde aus dem aktuellen Stand der Ensembl Datenbank bezogen und wird vom Ensembl Team zur Verfügung gestellt.
UCSC ID (mapped data)	Die UCSC ID wird von der UCSC Datenbank bezogen. Der UCSC Genome Browser wird entwickelt und gewartet von der Genome Bioinformatics Group, ein abteilungsübergreifendes Team innerhalb des Center for Biomolecular Science and Engineering (CBSE) an der University of California Santa Cruz (UCSC)

Tabelle 4
Beschreibung aller Felder, die ein HGNC-Datensatz enthält

Code

Nachfolgend ist der Code des Javaprogramms, das zum Einlesen der Daten in die Datenbank verwendet wurde. Die erste Klasse ist die Klasse *DBFiller,* die die Zeilen aus der Textdatei zerlegt, die Symbole sortiert und die Daten in die Datenbank einliest.

```
package database;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.util.GregorianCalendar;
import java.util.Vector;


public class DBFiller
{
  //Connection
  private Connection connectionMySQL = null;
  private static final String URL = "jdbc:mysql://localhost/hgnc";
  private static final String USER = "root";
  private static final String PASSWORD = "passwort";
  private static final String DRIVER = "com.mysql.jdbc.Driver";

  //Insert Statements
  private static final String HGNC_INSERT_STATEMENT = "INSERT INTO "+
   "hgnc(`HGNCID`, `LocusType`, `LocusGroup`, `Aliases`, `NameAliases`,"+
   "Chromosome, `DateApproved`, `DateModified`, `DateSymbolChanged`,"+
   "`DateNameChanged`, `AccessionNumbers`, `EnzymeIDs`, `EntrezGeneID`,"+
   "`EnsemblGeneID`, `MouseGenomeDatabaseID`,"+
   "`SpecialistDatabaseLinks`,`SpecialistDatabaseIDs`, `PubmedIDs`,"+
   "`RefSeqIDs`, `GeneFamilyName`,`RecordType`, `PrimaryIDs`,"+
   "`SecondaryIDs`, `CCDSIDs`, `VEGAIDs`,`LocusSpecificDatabases`,"+
   "`GDBIDmappedData`, `EntrezGeneIDmappedData`," +
   "`OMIMIDmappedData`, `RefSeqmappedData`, `UniProtIDmappedData`," +
   "`EnsemblIDmappedData`, `UCSCIDmappedData`," +
   "`MouseGenomeDatabaseIDmappedData`, `RatGenomeDatabaseIDmappedData`)"+
   "VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,"+
    ?,?,?,?,?)";
   private static final String SYMBOL_INSERT_STATEMENT = "INSERT INTO " +
    "symbols(`SymbolID`, `Symbol`, `SymbolFullName`, Status) values " +
    "(?,?,?,?)";
   private static final String HGNC_SYMBOL_INSERT_STATEMENT = "INSERT " +
      INTO hgnc_symbol (`HGNCID`,`SYMBOLID`) values (?,?)";
   private Vector<Symbol> symbols = new Vector<Symbol>();
   int q;
   private static final String prevSymStatus = "Previous Symbol";
   private int symbolID = 1;

   /**
   * @param args
   */
   public static void main(String[] args)
   {
      DBFiller db = new DBFiller();
      db.createConnection(); //Connect to Database(url, user, password)
      db.fillDB();//fill database
      db.closeConnection();//shutdown
   }
```

```
public void fillDB()
{
   String accessionNumbers, approvedSymbol, approvedName,
          previousSymbol, previousNames, aliases, nameAliases,
   specialistDatabaseIDs, specialistDatabaseLinks,
   locusSpecificDatabases, pubmedIDs, refseqIDs, ccdsIDs, hgncID,
   status,locusType, locusGroup, chromosome, enzymeIDs, ensembleIDs,
   mouseGenomeIDs, geneFamilyName, recordType, primaryID,
   secondaryID, vegaID, gdbIDmapped, omimIDmapped, refseqIDmapped,
   uniprotIDmapped, ensembleIDmapped, ucscIDmapped,
   mouseGenomeIDmapped, ratGenomeIDmapped;
   int entrezGeneID, entrezGeneIDmapped;
   String entrezGeneIDString, entrezGeneIDmappedString;
   GregorianCalendar dateApproved, dateModified, dateSymbolChanged,
          dateNamechanged;
   String dateApprovedString, dateModifiedString,
          dateSymbolChangedString, dateNameChangedString;
   File download = new File("hgnc_download.txt");
   boolean dateiKorrekt = this.testFile(download);
   if(!dateiKorrekt)
   {
      System.out.println("Datei fehlerhaft. Das Einlesen wurde
                                 abgebrochen!");
      System.exit(0);
   }
   String line;
   Symbol newSymbol;
   int linecounter = 0;
   try
   {
      BufferedReader reader = new BufferedReader(new
                                       FileReader(download));
      while((line = reader.readLine()) != null)
      {
         int index = 0;
         linecounter++;
         if(linecounter == 1)
         {
            continue;
         }
         else
         {
            PreparedStatement prep =
            connectionMySQL.prepareStatement(HGNC_INSERT_STATEMENT);
            //HGNC ID
            index = line.indexOf("\t");
            hgncID = line.substring(0, index);
            prep.setString(1, hgncID);

            //APPROVED SYMBOL
            line = line.substring(index+1);
            index = line.indexOf("\t");
            approvedSymbol = line.substring(0, index);
            //~withdrawn Einträge werden auf das Symbol reduziert
            int newIndex = approvedSymbol.indexOf("~");
            if(newIndex != -1)
            {
               approvedSymbol = line.substring(0, newIndex);
            }

            //APPROVED NAME
            line = line.substring(index+1);
            index = line.indexOf("\t");
```

```
approvedName = line.substring(0, index);

//STATUS
line = line.substring(index+1);
index = line.indexOf("\t");
status = line.substring(0, index);
//SYMBOL ZUM VECTOR HINZUFÜGEN
newSymbol = new Symbol(hgncID, approvedSymbol, approved
                    Name, status);
this.speicherInSymbolVector(newSymbol, hgncID);

//LOCUS TYPE
line = line.substring(index+1);
index = line.indexOf("\t");
locusType = line.substring(0, index);
prep.setString(2, locusType);

//LOCUS GROUP
line = line.substring(index+1);
index = line.indexOf("\t");
locusGroup = line.substring(0, index);
prep.setString(3, locusGroup);

//PREVIOUS SYMBOL
line = line.substring(index+1);
index = line.indexOf("\t");
previousSymbol = line.substring(0, index);

//PREVIOUS NAME
line = line.substring(index+1);
index = line.indexOf("\t");
previousNames = line.substring(0, index);
//PREVIOUS SYMBOL ZERLEGEN
if(previousSymbol.length() > 1 && previousSymbol != null)
{
   String onePreviousSymbol;
   newIndex = previousSymbol.indexOf(",");
   if(newIndex != -1)
   {
      //Mehrere Previous Symbols vorhanden
      do
      {
         onePreviousSymbol = previousSymbol.substring(0,
                                        newIndex);
         newSymbol = new Symbol(hgncID, onePreviousSymbol,
                           previousNames, prevSymStatus);
         //PREVIOUS SYMBOL ZUM VECTOR HINZUFÜGEN
         this.speicherInSymbolVector(newSymbol, hgncID);
         previousSymbol = previousSymbol.substring(
                                         newIndex+1);
         newIndex = previousSymbol.indexOf(",");
      }while(newIndex != -1);
   }

   else
   {
      //Nur ein Previous Symbol vorhanden
      newSymbol = new Symbol(hgncID, previousSymbol,
                           previousNames, prevSymStatus);
      this.speicherInSymbolVector(newSymbol, hgncID);
   }
}
```

```
else
{
   //Kein Previous Symbol vorhanden
   newSymbol = new Symbol(hgncID, previousSymbol,
                          previousNames, prevSymStatus);
   this.speicherInSymbolVector(newSymbol, hgncID);
}

//ALIASES
line = line.substring(index+1);
index = line.indexOf("\t");
aliases = line.substring(0, index);
prep.setString(4, aliases);

//NAME ALIASES
line = line.substring(index+1);
index = line.indexOf("\t");
nameAliases = line.substring(0, index);
prep.setString(5, nameAliases);

//CHROMOSOME
line = line.substring(index+1);
index = line.indexOf("\t");
chromosome = line.substring(0, index);
prep.setString(6, chromosome);

//DATE APPROVED
line = line.substring(index+1);
index = line.indexOf("\t");
dateApprovedString = line.substring(0, index);
dateApproved = createDate(dateApprovedString);
if(dateApproved == null)
{
   prep.setDate(7, null);
}
else
{
   prep.setDate(7, new(java.sql.Date
                    (dateApproved.getTimeInMillis()));
}

//DATE MODIFIED
line = line.substring(index+1);
index = line.indexOf("\t");
dateModifiedString = line.substring(0, index);
dateModified = createDate(dateModifiedString);
if(dateModified == null)
{
   prep.setDate(8, null);
}
else
{
   prep.setDate(8, new java.sql.Date
                    (dateModified.getTimeInMillis()));
}

//DATE SYMBOL CHANGED
line = line.substring(index+1);
index = line.indexOf("\t");
dateSymbolChangedString = line.substring(0, index);
dateSymbolChanged = createDate(dateSymbolChangedString);
if(dateSymbolChanged == null)
```

```
{
   prep.setDate(9, null);
}
else
{
   prep.setDate(9, new java.sql.Date
               (dateSymbolChanged.getTimeInMillis()));
}

//DATE NAME CHANGED
line = line.substring(index+1);
index = line.indexOf("\t");
dateNameChangedString = line.substring(0, index);
dateNamechanged = createDate(dateNameChangedString);
if(dateNamechanged == null)
{
   prep.setDate(10, null);
}
else
{
   prep.setDate(10, new java.sql.Date
                     (dateNamechanged.getTimeInMillis()));
}

//ACCESSION NUMBERS
line = line.substring(index+1);
index = line.indexOf("\t");
accessionNumbers = line.substring(0, index);
prep.setString(11, accessionNumbers);

//ENZYME IDS
line = line.substring(index+1);
index = line.indexOf("\t");
enzymeIDs = line.substring(0, index);
prep.setString(12, enzymeIDs);

//ENTREZ GENE ID
line = line.substring(index+1);
index = line.indexOf("\t");
entrezGeneIDString = line.substring(0, index);
if(entrezGeneIDString.trim().length() > 0)
{
   entrezGeneID = Integer.valueOf(line.substring(
                                        0, index));
   prep.setInt(13, entrezGeneID);
}
else
{
   prep.setInt(13, -1);
}

//ENSEMBLE ID
line = line.substring(index+1);
index = line.indexOf("\t");
ensembleIDs = line.substring(0, index);
prep.setString(14, ensembleIDs);

//MOUSE GENOME ID
line = line.substring(index+1);
index = line.indexOf("\t");
mouseGenomeIDs = line.substring(0, index);
prep.setString(15, mouseGenomeIDs);
```

```
//SPECIALST DATABASE LINKS
line = line.substring(index+1);
index = line.indexOf("\t");
specialistDatabaseLinks = line.substring(0, index);
prep.setString(16, specialistDatabaseLinks);

//SPECIALIST DATABASE IDS
line = line.substring(index+1);
index = line.indexOf("\t");
specialistDatabaseIDs = line.substring(0, index);
prep.setString(17, specialistDatabaseIDs);

//PUBMED IDS
line = line.substring(index+1);
index = line.indexOf("\t");
pubmedIDs = line.substring(0, index);
prep.setString(18, pubmedIDs);

//REFSEQ IDS
line = line.substring(index+1);
index = line.indexOf("\t");
refseqIDs = line.substring(0, index);
prep.setString(19, refseqIDs);

//GENE FAMILY NAME
line = line.substring(index+1);
index = line.indexOf("\t");
geneFamilyName = line.substring(0, index);
prep.setString(20, geneFamilyName);

//RECORD TYPE
line = line.substring(index+1);
index = line.indexOf("\t");
recordType = line.substring(0, index);
prep.setString(21, recordType);

//PRIMARY ID
line = line.substring(index+1);
index = line.indexOf("\t");
primaryID = line.substring(0, index);
prep.setString(22, primaryID);

//SECONDARY ID
line = line.substring(index+1);
index = line.indexOf("\t");
secondaryID = line.substring(0, index);
prep.setString(23, secondaryID);

//CCDS IDS
line = line.substring(index+1);
index = line.indexOf("\t");
ccdsIDs = line.substring(0, index);
prep.setString(24, ccdsIDs);

//VEGA ID
line = line.substring(index+1);
index = line.indexOf("\t");
vegaID = line.substring(0, index);
prep.setString(25, vegaID);

//LOCUS SPECIFIC DATABASE
line = line.substring(index+1);
index = line.indexOf("\t");
```

```
locusSpecificDatabases = line.substring(0, index);
prep.setString(26, locusSpecificDatabases);

//GDB ID (mapped data)
line = line.substring(index+1);
index = line.indexOf("\t");
gdbIDmapped = line.substring(0, index);
prep.setString(27, gdbIDmapped);

//ENTREZ GENE ID (mapped data)
line = line.substring(index+1);
index = line.indexOf("\t");
entrezGeneIDmappedString = line.substring(0, index);
if(entrezGeneIDmappedString.trim().length() > 0)
{
   entrezGeneIDmapped = Integer.valueOf(line.substring(
                                                0, index));
   prep.setInt(28, entrezGeneIDmapped);
}
else
{
   prep.setInt(28, -1);
}

//OMIM ID (mapped data)
line = line.substring(index+1);
index = line.indexOf("\t");
omimIDmapped = line.substring(0, index);
prep.setString(29, omimIDmapped);

//REFSEQ ID (mapped data)
line = line.substring(index+1);
index = line.indexOf("\t");
refseqIDmapped = line.substring(0, index);
prep.setString(30, refseqIDmapped);

//UNIPROT ID (mapped data)
line = line.substring(index+1);
index = line.indexOf("\t");
uniprotIDmapped = line.substring(0, index);
prep.setString(31, uniprotIDmapped);

//ENSEMBLE ID (mapped data)
line = line.substring(index+1);
index = line.indexOf("\t");
ensembleIDmapped = line.substring(0, index);
prep.setString(32, ensembleIDmapped);

//UCSC ID (mapped data)
line = line.substring(index+1);
index = line.indexOf("\t");
ucscIDmapped = line.substring(0, index);
prep.setString(33, ucscIDmapped);

//MOUSE GENOME ID (mapped data)
line = line.substring(index+1);
index = line.indexOf("\t");
mouseGenomeIDmapped = line.substring(0, index);
prep.setString(34, mouseGenomeIDmapped);

//RAT GENOME ID (mapped data)
line = line.substring(index+1);//Kein Index mehr, da kein
                                  Tab mehr vorhanden
```

```
                ratGenomeIDmapped = line;
                prep.setString(35, ratGenomeIDmapped);

                //INSERT TABLE HGNC
                prep.executeUpdate();
            }
        }
        //INSERT TABLE SYMBOL
        PreparedStatement prepSymbol =
        connectionMySQL.prepareStatement(SYMBOL_INSERT_STATEMENT);
        PreparedStatement prepIDTabelle =
        connectionMySQL.prepareStatement(HGNC_SYMBOL_INSERT_STATEMENT);
        Symbol tempSymbol;
        Vector<String> hgncIDs = new Vector<String>();
        String tempHgnc;
        int symbolID;
        for(q = 0; q < symbols.size(); q++)
        {
            tempSymbol = symbols.get(q);
            symbolID = tempSymbol.getSymbolID();
            prepSymbol.setInt(1, symbolID);
            prepSymbol.setString(2, tempSymbol.getSymbol());
            prepSymbol.setString(3, tempSymbol.getSymbolName());
            prepSymbol.setString(4, tempSymbol.getStatus());
            prepSymbol.executeUpdate();
            //INSERT m:n-Tabelle
            hgncIDs = tempSymbol.getHgncID();
            for(int j = 0; j < hgncIDs.size(); j++)
            {
                tempHgnc = hgncIDs.get(j);
                prepIDTabelle.setString(1, tempHgnc);
                prepIDTabelle.setInt(2, symbolID);
                prepIDTabelle.executeUpdate();
            }
        }
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}

private boolean testFile(File datei)
{
    String line;
    int anzahlSpalten = 40;
    int zahl;
    try
    {
        BufferedReader reader = new BufferedReader(
                                    new FileReader(datei));
        while ((line = reader.readLine()) != null)
        {
            zahl = tabCounter(line);
            if(zahl != anzahlSpalten)
            {
                return false;
            }
        }
    }
    catch (Exception e)
    {
        return false;
```

```
    }
    return true;
}

private static int tabCounter(String line)
{
    int counter = 1;
    int index;
    while(line.contains("\t"))
    {
        counter++;
        index = line.indexOf("\t");
        line = line.substring(index+1);
    }
    return counter;
}

private void speicherInSymbolVector(Symbol symbol,
                                    String aktuellHgncID)
{
    Symbol tempSymbol;
    boolean isNotInVector = true;
    if(symbol.getSymbol().trim().length() > 1)
    {
          //Symbol not null
          for(int i = 0; i < symbols.size(); i++)
          {
             tempSymbol = symbols.get(i);
             if(tempSymbol.getSymbol().equalsIgnoreCase(
                                    symbol.getSymbol()))
             {
                if(tempSymbol.getStatus().equalsIgnoreCase(
                                    symbol.getStatus()))
                {
                   //Symbol ist schon im Vector vorhanden und der aktu-
                     elle Status ist bei beiden gleich
                   isNotInVector = false;
                   Vector<String> hgncTemp = tempSymbol.getHgncID();
                   boolean idNotYetExists = true;
                   for(int j = 0; j < hgncTemp.size(); j++)
                   {
                      if(aktuellHgncID.equals(hgncTemp.get(j)))
                      {
                         //Die HGNC Id des neuen Symbols ist bereits
                           gespeichert und soll nicht neu aufgenommen
                           werden
                         idNotYetExists = false;
                         break;
                      }
                   }
                   if(idNotYetExists)
                   {
                      //Die HGNC ID des neuen Symbols ist noch nicht
                        vorhanden und wird gespeichert
                      tempSymbol.addHgncID(aktuellHgncID);
                   }
                   break;//Schleife wird abgebrochen, da Symbol vor-
                             handen und die HGNCID hinzugefügt wurde
                }
             }
          }
       }
```

```
    else
    {
       //Das Symbol ist null

       if(symbol.getSymbolName().trim().length() > 0)
       {
          //Symbol Name ist nicht null
          for(int i = 0; i < symbols.size();i++)
          {
             tempSymbol = symbols.get(i);
             if(tempSymbol.getSymbolName().equalsIgnoreCase
                   (symbol.getSymbolName()))
             {
                //Der (previous) Name ist schon  vorhanden
                isNotInVector = false;
                break;
             }
          }
       }
       else
       {
          //Symbol Name ist null
          for(int i = 0; i < symbols.size();i++)
          {
             tempSymbol = symbols.get(i);
             if(tempSymbol.getSymbolName().trim().length() == 0)
             {
                Vector<String> hgncTemp = tempSymbol.getHgncID();
                boolean idNotYetExists = true;
                for(int j = 0; j < hgncTemp.size(); j++)
                {
                         if(aktuellHgncID.equals(hgncTemp.get(j)))
                   {
                      //Die HGNC Id des neuen Symbols ist bereits
                         gespeichert und soll nicht neu aufgenommen
                         werden
                      idNotYetExists = false;
                      break;
                   }
                }
                if(idNotYetExists)
                {
                   //Die HGNC ID des neuen Symbols ist noch nicht
                         vorhanden und wird gespeichert
                   tempSymbol.addHgncID(aktuellHgncID);
                }
                isNotInVector = false;
                break;
             }
          }
       }
    }
    if(isNotInVector)
    {
       //Das Symbol ist noch nicht im Vector und wird gespeichert
       //ID setzen
       symbol.setSymbolID(symbolID);
       symbolID++;
       symbols.add(symbol);
    }
}
```

```
    private GregorianCalendar createDate(String dateString)
    {
       GregorianCalendar date = null;
       String copy = dateString;
       if(copy.trim().isEmpty())
       {
          return date;
       }
       if(dateString.length() > 1 && dateString != null)
       {
          int year = Integer.valueOf(dateString.substring(0, 4));
          int month = (Integer.valueOf(dateString.substring(6, 7)))-1;
          int day = Integer.valueOf(dateString.substring(8, 10));
          date = new GregorianCalendar(year, month, day);
       }
       return date;
    }

    protected void createConnection()
    {
       try
       {
          Class.forName(DRIVER).newInstance();
          connectionMySQL = DriverManager.getConnection(URL, USER, PASS
                                                        WORD);
       }
       catch(Exception e)
       {
          e.printStackTrace();
       }
    }

    protected void closeConnection()
    {
       if(connectionMySQL != null)
       {
          try
          {
             connectionMySQL.close();
          }
          catch(Exception e)
          {
             e.printStackTrace();
          }
       }
    }
}
```

Diese Klasse ist die Klasse *Symbol*. Sie erzeugt ein Symbol-Objekt, dass aus einer Symbol-ID, einem Symbol, einem Symbolnamen, einem Status, und einem Vector mit HGNC-IDs, die auf dieses Symbol verweisen.

```
package database;

import java.util.Vector;

public class Symbol
{
	private Vector<String> hgncID = new Vector<String>();
	private String symbol;
	private String symbolName;
	private String status;
	private int symbolID;

	public Symbol(String hgncID, String symbol, String symbolName,
			String status)
	{
		this.hgncID.add(hgncID);
		this.symbol = symbol;
		this.symbolName = symbolName;
		this.status = status;
	}

	public Vector<String> getHgncID()
	{
		return hgncID;
	}

	public void addHgncID(String hgncID)
	{
		this.hgncID.add(hgncID);
	}

	public String getSymbol()
	{
		return symbol;
	}

	public String getSymbolName()
	{
		return symbolName;
	}

	public void setSymbolName(String symbolName)
	{
		this.symbolName = symbolName;
	}

	public String getStatus()
	{
		return status;
	}

	public void setStatus(String status)
	{
		this.status = status;
	}

	public int getSymbolID()
	{
```

```
            return symbolID;
      }

      public void setSymbolID(int id)
      {
            this.symbolID = id;
      }
}
```