Johann Höftberger

Reverse Engineering und Erweiterung der Mikrobiom-Analyse Plattform SnoWMan

Master's Thesis

Technische Universität Graz

Institut für Genomik und Bioinformatik

Betreuer & Gutachter: Dr. Gerhard Thallinger

Graz, Mai 2013

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Danksagung

Ich bedanke mich bei Kathrin Paller, dass sie mir hilft ein besserer Mensch zu sein.

Meinem Betreuer Gerhard Thallinger danke ich für die Unterstützung bei diesem Projekt und allen Kollegen am *Institut für Genomik und Bioinformatik* die mir mit Rat und Tat zur Seite standen, insbesondere Bettina Halwachs und Peter Krempl.

Abstract

1. Deutsch

Die vorliegende Masterarbeit beschreibt die Softwarestruktur-Analyse - Reverse Engineering - der *SnoWMAn*-Web-Plattform des *Instituts für Genomik und Bioinformatik* der *TU Graz* zur Untersuchung von Mikrobiom-Daten aus Hochdurchsatz Sequenzierverfahren. Die extrahierten Architektur-Muster dienten als Wissensbasis für die Erweiterungen im zweiten Teil der Arbeit. Sie wurden analysiert und führten zum Refactoring von Software-Modulen, die für die gewünschten Erweiterungen notwendig waren. Resultierende Verbesserungsvorschläge wurden herausgearbeitet und festgehalten.
Nach Abschluss des Reverse Engineerings wurde die Plattform um die *Chimerafiltering*-Funktion erweitert, bei der geplanten *Denoising*-Erweiterung erwies sich die verfügbare Rechenleistung des Rechenclusters als Hürde.
Die positiven Auswirkungen des eingeführten *Chimerafilterings* wurden durch eigene Analysen gezeigt. Bei als eindeutig erkannten Sequenzen zeigte sich eine Reduzierung von zumindest -3,10%, die höchste Reduzierung der OTU-Anzahl trat in der *UCLUST*-Pipeline beim Ähnlichkeitsmaß 95% mit -15,15% auf.
Abschließend werden Verbesserungsvorschläge bei zukünftigen Arbeiten an *SnoWMAn* diskutiert.

Stichwörter: *SnoWMAn*, Reverse Engineering, Refactoring, Denoising-Erweiterung, Chimerafiltering-Erweiterung

2. English

The present master thesis describes the reverse engineering of the *SnoWMAn*-Web-application of the *Institute of Genomics and Bioinformatics* at the *University of Technology* in Graz which is for analysis of large microbiome sequencing data. The extracted software-patterns provided the necessary knowledge to extend the platform in the second part of the work. They were analyzed and software modules which had impeded the planned extensions were refactored. Resulting improvement suggestions have been worked out and have been collected.
After the reverse engineering had been done the platform was extended with *Chimerafiltering*, for the intended *Denoising*-extension the provided computing power of the used computer cluster turned out to be an obstacle. The positive effects of the introduced *Chimerafiltering* are shown by means of own analysis. The reduction of unique sequences was -3.10% or higher, the highest reduction of OTUs occured in the *UCLUST*-pipeline whith -15.15% at 95% similarity.
In conclusion some improvement suggestions for future works on *SnoWMAn* are discussed.

Keywords: *SnoWMAn*, reverse engineering, refactoring, denoising-extension, chimerafiltering-extension

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

1.1. Das menschliche Mikrobiom

1.1.1. Mikrobiom - Begriffserklärung

Unter einem Mikrobiom versteht man die Gesamtheit aller Mikroben in einer definierten Umgebung, deren umgebungsbedingte Interaktion und ihre genetische Information, sprich ihre Genome. Mikroben, auch als Mikroorganismen bezeichnet, sind einzellige oder wenigzellige Lebewesen und als solche äußerst divers. Der Begriff *Mikrobiom* wurde durch Joshua Lederberg geprägt [1], der die große Bedeutung im Zusammenhang mit der menschlichen Physiologie erkannte. Mikrobiota werden bestimmten Regionen zugeordnet und weisen eine ortsabhängige Zusammensetzung auf. Die Besiedelung des Menschen durch Mikroorganismen beginnt nach seiner Geburt. Mikroben werden in die Gruppen

- Bakterien
- Archaeen
- Pilze
- Viren

unterteilt. Das menschliche Mikrobiom betreffend wird der Körper in fünf Hauptregionen

- Atemwege
- Haut
- Mundhöhle
- Gastrointestinal-Trakt
- Urogenital-Trakt

eingeteilt.

1.1.2. Allgemeine Erläuterungen zum Mikrobiom

Die Anzahl der Zellen des menschlichen Mikrobioms übersteigt die Anzahl an Zellen des menschlichen Körpers um den Faktor 10 [2]. Dieses Verhältnis ist umso erstaunlicher, bedenkt man, dass das menschliche Mikrobiom bis vor kurzer Zeit noch wenig Beachtung fand und erst seit einigen Jahren genauer untersucht wird.
Wie sich gezeigt hat, wird die Zusammensetzung eines Mikrobioms von vielen großteils noch unbekannten Faktoren beeinflusst. Beim Menschen sind nachgewiesene Einflussfaktoren die Körperregion [3], die Ernährung, das kulturelle Umfeld mit den daraus resultierenden Lebensgewohnheiten, individuelle Faktoren und fragile Balancezustände zwischen unterschiedlichen mikrobiellen Gruppen in einer Region [4]. Offensichtlich gibt es einen starken Zusammenhang zwischen Mikroorganismen und den Bedingungen unter denen sie entstehen. Übertragen auf den Menschen lässt dies den Schluss zu, dass über die Untersuchung des menschlichen Mikrobioms Rückschlüsse auf seinen Gesundheitszustand gezogen werden können [5]. Betrachtet man die große Diversität von Mikroorganismen und die Vielzahl mikrobieller Zellen im Verhältnis zu menschlicher Zellen stellt sich die Frage, ob das menschliche Mikrobiom nicht ursächlich den Gesundheitszustand des Menschen beeinflusst. Gestärkt wird dieser Ansatz unter anderem durch Ergebnisse des bereits besser untersuchten Magen-Darm-Trakt-Mikrobioms, die auf einen Zusammenhang zu schweren Erkrankungen wie beispielsweise Diabetes schließen lassen [6].
Motiviert durch diese ersten Einsichten wird in der jüngsten Vergangenheit vermehrt auf eine gezielte und strukturierte Erforschung des menschlichen Mikrobioms und anderer Mikrobiota gedrängt. Durch diesen neuen Ansatz, die mikrobielle Zusammensetzungen von definierten Umgebungen in Bezug auf konkrete Krankheitsbilder zu charakterisieren, wird erhofft, unter anderem die Ursache vieler bekannter Krankheiten besser zu verstehen und im Idealfall neue Behandlungsansätze finden zu können. Durch die lange Nichtbeachtung dieses Aspekts und die vermuteten Einflüsse auf biologische Systeme, wie zum Beispiel den Menschen, deuten sich weitreichende

Anwendungsmöglichkeiten an, und tiefer gehende Untersuchungen in vielen Bereichen drängen sich auf. 2007 wurde das *H*uman *M*icrobiom *P*roject, kurz HMP, als bisher größtes Unterfangen seiner Art, zur fundierten Untersuchung der wichtigsten Aspekte des menschlichen Mikrobioms durch die *N*ational *I*nstitutes of *H*ealth, NIH, Roadmap for Biomedical Research [7] in den USA gestartet [8]. Dieses Projekt setzte sich folgende konkrete Ziele

- Unter Einsatz moderner Hochdurchsatz Sequenzierungstechnologien, mittels der Untersuchung verschiedener Körperregionen von zumindest 250 Individuen, ein tieferes Verständnis des menschlichen Mikrobioms aufzubauen
- Durch die Untersuchung von Personen mit bekannten Krankheiten Zusammenhänge zwischen Veränderungen des Mikrobioms und einer Krankheit beziehungsweise der Gesundheit des Menschen herzustellen
- Eine neue fundierte bioinformatische Datenbasis und die notwendigen technologischen Prozesse zur Ermittlung dieser für weitere Forschungen in diesem Bereich zu etablieren [9]

Durch dieses Projekt gelang erstmals eine generelle Charakterisierung des menschlichen Mikrobioms. Nach Abschluss des Projekts wurde ein Referenz Genom des menschlichen Mikrobioms vorgelegt und online zur Verfügung gestellt [10].

Etwas später, 2008, wurde in Europa das *MetaHIT*-Projekt [6] mit dem Ziel, Zusammenhänge zwischen dem menschlichen Darm-Mikrobiom und der Gesundheit beziehungsweise Krankheit eines Menschen zu finden, gestartet. Finanziert durch die Europäische Kommission, 7th *F*ramework *P*rogram - FP7, waren die Arbeiten auf viereinhalb Jahre bis 30. Juni 2012 angelegt und konzentrierten sich hauptsächlich auf zwei in Europa in den letzten Jahrzehnten vermehrt auftretende Krankheitsbilder:

- chronisch-entzündliche Darmerkrankungen - CED (Inflammatory Bowel Disease - IBD) und
- Fettleibigkeit (Obesity)

Die zwei Hauptergebnisse der Arbeiten sind das Vorlegen eines umfassenden Gen-Referenzkatalogs des menschlichen Darm-Mikrobioms (Darm-Metagenom) [11] und das Finden von drei *Enterotypen* in der weltweiten

humanen Population [12]. Die Unterteilung in Enterotypen ist eine Klassifikation der menschlichen Darm-Mikroorganismen aufgrund des vorherrschenden bakteriellen Ökosystems.

Die Interaktionsvorgänge und -mechanismen zwischen einem Mikrobiom und seiner Umgebung in der es entsteht sind großteils noch unbekannt. Aus diesem Grund versucht man Themen wie

- Phylogenie
- Taxonomie
- Biogeographie
- Ökologie
- Metabolismus und
- Funktion

bei Untersuchungen zu klären [13].

Die Vielzahl der angeführten Fragestellungen kombiniert mit der großen Anzahl an mikrobiellen Zellen und deren Diversität lässt erahnen, wie aufwendig Untersuchungen in diesem Bereich sind. Es ergeben sich umfangreiche bioinformatische und biologische Themen, die Datenanalyse ist immer mit einem hohen Ressourcenbedarf verbunden.

Zur Charakterisierung eines Mikrobioms aufgrund der analysierten Sequenzen werden verschiedene Kennzahlen ermittelt. Diese Kennzahlen können aufgrund einer mehrstufigen Verarbeitung der *Amplicons*, den DNA-Vervielfältigungsprodukten bei der Hochdurchsatz-Sequenzierung, berechnet werden. Der Aufbau einer Mikrobiom-Analyse besteht grundsätzlich aus den in Abbildung 1.1 dargestellten Schritten. Für jeden einzelnen Berechnungsschritt stehen unterschiedliche Software-Tools zur Auswahl. Beim *Sample Splitting & Sequence Filtering* werden die Eingangs-Sequenzen aufgrund von Barcodes aufgetrennt, Primer entfernt und Sequenzen mit schlechter Qualität ausgefiltert. Das *Sequence Alignment* ist ein musterbasiertes Suchverfahren das unter Beibehaltung der Nukleotidabfolge der Sequenzen die einzelnen Sequenzen zueinander ordnet, um Ähnlichkeiten in den Sequenzen zu finden. Ähnliche Nukleotidfolgen deuten auf ähnliche Funktion und evolutionäre Verwandtschaft hin. Jedes Nukleotid der Ausgangssequenz wird dabei einem Nukleotid der zu vergleichenden Sequenz oder einem *Gap* zugeordnet. Ein *Gap* resultiert aus einer *Insertion* oder *Deletion* des

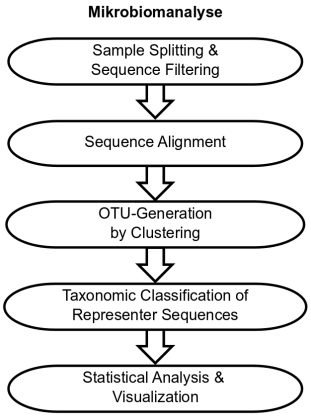

Abbildung 1.1.: Grundsätzlicher Ablauf einer Mikrobiomanalyse (mit unsupervised Clustering) mit Darstellung der notwendigen Schritte zur Verarbeitung

Nukleotids in einer oder beiden Sequenzen. Das *Sequence Alignment* misst die Ähnlichkeit von Sequenzen. Beim *Clustering* werden die Sequenzen aufgrund von festgelegten Ähnlichkeitsmaßen gruppiert und sogenannte OTUs erzeugt. Im Klassifikations-Schritt werden diese OTUs einer taxonomischen Gruppe zugeordnet. Aufgrund dieser wird mittels statistischer Analyse die Diversität (α-, β-Diversität) ermittelt. Somit werden unterschiedliche Mikrobiome vergleichbar.

1.1.3. *SnoWMAn* - Plattform zur Verarbeitung und Analyse von Mikrobiomdaten

Um die notwendigen Anforderungen bei der Mikrobiomdatenverarbeitung erfüllen zu können, bedarf es verschiedenster Werkzeuge in Form von Computerprogrammen. Aus Anwendersicht ist es wünschenswert, sich möglichst wenig um die Bedienung dieser Programme kümmern zu müssen und sich auf die verfolgten Ziele der angewandten Verarbeitung konzentrieren zu können. Zur Adressierung dieses Problems gibt es umfangreiche Computerprogramme und -plattformen. Eine dieser Plattformen ist *SnoWMAn* [14],

diese wurde am *Institut für Genomik und Bioinformatik* [15] der *TU Graz* [16] entwickelt und wird im Anschluss erläutert.

SnoWMAn verarbeitet Mikrobiomdaten aus Hochdurchsatzsequenzierverfahren und umfasst den gesamten Mikrobiom-Analyse-Prozess, bietet Unterstützung vom Preprocessing bis zur Visualisierung von Ergebnissen. Das Programm ist als Computerplattform konzipiert und mittels der *J*ava *E*nterprise *E*dition-, Java EE-, Technologie [17] umgesetzt. Für den Benutzer stellt sich das System als Webpage mit spezieller Funktionalität dar und ist mit einem aktuellen Browser (Firefox, Internet Explorer) über das Internet nutzbar. Es entbindet den Anwender von jeglicher lokalen Softwareinstallation am eigenen Computer und erlaubt den sofortigen Gebrauch der Applikation. Bei der Nutzung kann aus fünf verschiedenen Verarbeitungs-*Pipelines* gewählt und deren konkretes Verhalten beeinflusst werden. Die fünf Pipelines werden bezüglich der Charakterisierung und Klassifikation der Sequenzdaten in Taxonomie-unabhängige (*BLAT*, *JGAST*) und Sequenzunabhängige (*Mothur*, *RDP*, *UCLUST*) Varianten unterschieden.
Das grundsätzliche Nutzungsszenario ist dreistufig gestaltet und sieht folgendermaßen aus. Zu Beginn werden die Daten und ihre Meta-Daten auf die Plattform hoch geladen. Die Sequenzdaten sind dabei im *FASTA*-Format zur Verfügung zu stellen, die Meta-Daten als plain-text-Dateien anzugeben. Sind die zu analysierenden Daten auf der Plattform nutzbar, wählt man als zweiten Prozessschritt eine der fünf vorgegebenen Verarbeitungsvarianten. Das System unterstützt den Benutzer durch intelligent gewählte Preset-Werte für die Parameter des Preprocessings und der gewählten Pipeline. Sind die Parameter den Absichten entsprechend festgelegt, "startet" (beauftragt) der Anwender im Browser die Berechnung. Aufgrund der Größe der Daten und der Komplexität der angewandten mathematischen Verfahren übersteigt der Berechnungsaufwand bei weitem die Möglichkeiten eines *P*ersonal *C*omputers. Die *SnoWMAn*-Plattform verteilt während der Verarbeitung im Hintergrund vollautomatisch aufwendige Berechnungsroutinen an einen Rechencluster des *Instituts für Genomik und Bioinformatik* und reduziert dadurch deutlich die Wartezeit auf Ergebnisse. Als Anwender muss man während der Berechnung nicht online auf die Ergebnisse warten, sondern kann die Sitzung jederzeit beenden. Bei Bedarf ermöglicht die Plattform die Wiederanmeldung zu einem späteren Zeitpunkt, um sich über den Stand der Analyse zu informieren. Alle gestarteten *Pipeline*runs sind eindeutig

einem authentifizierten Nutzer des Systems zugeordnet. Falls gewünscht, ist beim Starten einer Analyse eine Emailbenachrichtigung über den Abschluss der Berechnungen wählbar, was nach Beendigung eine Benachrichtigung an die angegebenen Adresse zur Folge hat.

Liegen die Ergebnisse vor, können als dritter und letzter Schritt die Resultate statistisch ausgewertet und visualisiert werden. *SnoWMAn* erlaubt durch eine Vielzahl angepasster Diagramme die Beurteilung der α- und / oder β-Diversität, ermöglicht eine *P*rincipal *C*omponent *A*nalysis - PCA und die Darstellung der Resultate als Balken-, Torten-, Linien- und Venn-Diagramm. Alle generierten Grafiken können in den Dateiformaten .PNG und .SVG abgespeichert und die Daten im *Microsoft Excel*-Format exportiert werden.

1.1.4. Chimären und Noise in Sequenz-Daten

In vielen aktuellen Hochdurchsatzsequenzierverfahren wie zum Beispiel der 454 Pyrosequenzierung [18] wird das zu untersuchende Genmaterial vor der tatsächlichen Sequenzierung mittels *P*olymerase *C*hain *R*eaction (PCR)-Methode [19] amplifiziert. Dieser Prozessschritt des Klonens verursacht allerdings systematische Fehler bei den entstehenden Amplifikationsprodukten die der Sequenzierung zugeführt werden. Da die produzierten Klone für die weitere Untersuchung in der Masse nicht 100%ig den Ausgangsprodukten entsprechen, werden hier Fehler eingeschleust. Durch das Klonen entstehen zum Teil falsche Moleküle in Form von sogenannten *Chimären*, das sind Verbindungen von zwei oder mehreren Sequenz-Templates die fälschlicher Weise zu einer Gesamtsequenz verbunden wurden, und andere fehlerhafte Sequenz-*Kopien*. Diese Fehler erschweren in weiterer Folge die korrekte phylogenetische Analyse [20]. Weitere Fehler die im Sequenzierungsprozess auftreten sind *Single Base Substitutions* (SNPs) und *Insertionen* und *Deletionen* bei Homopolymerabschnitten [21]. Diese drei Hauptfehlerquellen führen zu verrauschten Daten, verursachen verfälschte Ergebnisse bei nachfolgenden Analysen. Es kommt zur sogenannten *O*perational *T*axonomic *U*nit (*OTU*)-*inflation*, bei der mehr OTUs unterschieden werden als tatsächlich, biologisch begründet, vorhanden sind. Wie in [21] klar nachgewiesen wird, lohnt die Anwendung von neuen mathematischen Verfahren zur Beseitigung beziehungsweise Minderung dieser drei systematischen

Fehlerquellen und verbessert signifikant nachfolgende Analyseergebnisse. Da in der *SnoWMAn*-Plattform noch keine derartigen Fehlerkorrekturmaßnahmen zur Anwendung kommen, sollen ein *Denoising* zur Reduzierung der Homopolymerfehler und *Chimerafiltering* der Input-Daten eingeführt werden.

1.2. Beschreibung der Aufgabenstellung

Die Ziele der vorliegenden Masterarbeit sind im Wesentlichen in zwei Bereiche unterteilt. Im ersten Teil soll ein

- Reverse Engineering zum Wissensaufbau und falls notwendig ein
- Refactoring bestehender Software-Module

durchgeführt werden.
Da keine Dokumentation existiert gilt es, durch Untersuchungen das notwendige Wissen über softwareinterne Abläufe und deren technischer Umsetzung aufzubauen. Durch das Reverse Engineering der *SnoWMAn*-Plattform sollen die technischen Voraussetzungen für die gewollte funktionelle Erweiterung des zweiten Teils geschaffen werden. Werden technische Probleme bzw. Schwächen entdeckt, die der gewünschten Erweiterung entgegen stehen, sind diese zu korrigieren bzw. auszubessern. Bei sämtlichen Änderungen ist darauf zu achten, keine vorhandenen Leistungsmerkmale von *SnoWMAn* zu brechen. Die zugrunde liegenden Muster der gefundenen Probleme sind zu extrahieren und in abstrakter Form heraus zu arbeiten, um sie besser analysieren zu können. Sie sollten einer genauen Betrachtung und Problemlösung unterzogen werden.

Der zweite Teil sieht das Hinzufügen der Funktionen

- Denoising und
- Chimerafiltering

der Sequenzdaten bei der Verarbeitung vor. Es sind die notwendigen technischen Änderungen in allen Teilbereichen der Software vorzunehmen, damit dem Benutzer nach der Implementierung diese Erweiterungen uneingeschränkt in jeder der fünf Pipelines zur Verfügung stehen. Als Software-Tool

das die beiden gewünschten Aufgaben konkret prozessiert ist die aktuellste Version von *mothur* [22] einzusetzen und in *SnoWMAn* zu integrieren.

2. Umsetzung

2.1. Entwicklungsumgebung - Softwaretools und deren Versionen

Die Codebasis von *SnoWMAn* ist mittels *Java EE* und verwandten Softwaretechnologien umgesetzt. *Java EE* ist konzipiert Three- beziehungsweise n-Tier-Softwarearchitekturen zu unterstützen. Abbildung 2.1 gibt die grundsätzliche Aufteilung der Schichten mit deren Zuordnung zu gedachten Computern wieder.

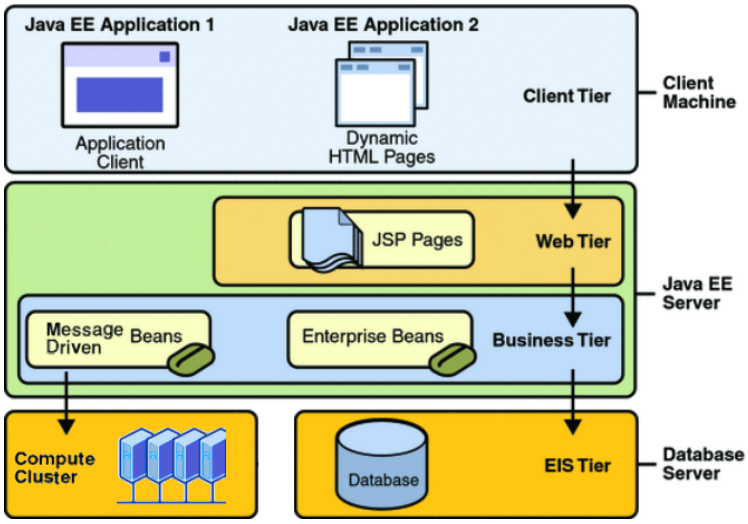

Abbildung 2.1.: Die bei *SnoWMAn* angewandte, adaptierte Three-Tier Architektur mit *Java EE* [17], die eingesetzten Technologien werden dabei den jeweiligen Schichten (*Tiers*) zugeordnet

Entsprechend des inneren Aufbaus von *SnoWMAn* werden die eingesetzten Technologien in die Bereiche *Webinterface* und *Berechnungsservice* unterteilt. Unter Webinterface sind die Aspekte für die Generierung und Auslieferung der HTML-Seiten wie auch der Benutzerinteraktion zusammengefasst. Zum Einsatz kommen hier das *Apache Struts*-Framework als *M*odell *V*iew *C*ontroller - MVC [23], *J*ava *S*erver *P*ages - JSP [24], *E*nterprise *J*ava *B*eans - EJB [25], *J*ava *S*cript - JS [26] und *C*ascading *S*tyle *S*heets - CSS [27] zur Gestaltung der ausgelieferten HTML-Seiten.
Im Berechnungsservice - Backend werden Message Driven Java Beans - MDB zur asynchronen Überwachung der Pipelineberechnungen [25], *S*imple *O*bject *A*ccess *P*rotocol (*SOAP*) basierte Webservices mit *JClusterService* [28] zur Entkopplung zwischen Applikationsserver und *JClusterService* und als Cluster Queuing System die *Sun Grid Engine* [29] eingesetzt. Im Hintergrund am Rechencluster werden die Berechnungen teilweise durch Linux-*Bash*-Skripte ausgeführt.

Um Softwareerweiterungen unter den angeführten Rahmenbedingungen entwickeln zu können, wurde folgende Entwicklungsumgebung installiert und verwendet. Teilweise waren ganz bestimmte Versionen von Programmen notwendig, die konkreten Details werden jeweils angeführt.

*J*ava *R*untime *E*nvironment - JRE, *J*ava *D*evelopment *K*it - JDK

SnoWMAn verlangt in der vorliegenden Implementierung für die Ausführung eine JRE Version 1.5 und für die Entwicklung von Erweiterungen ein JDK in Version 1.5 [30].

JBoss - Applikationsserver

JBoss [31] ist ein Applikationsserver der den *Java EE*-Standard implementiert. Er ist von Sun als Java-kompatibel zertifiziert und stellt eine standardkonforme *Java EE*-Laufzeitumgebung für Java-Programme zur Verfügung. Die *SnoWMAn*-Anwendung wird nach dem Kompilieren und Builden auf den

JBoss-Applikationsserver deployt und von diesem als Web-Anwendung, über einen Browser nutzbar, ausgeführt. Für *SnoWMAn* wird *JBoss* in der Version 4.02 verwendet.

MySQL - Relationale Datenbank

Die relationale Datenbank *MySQL* [32] wird von *SnoWMAn* zum Speichern von Benutzer- und Analysedaten genutzt. Für die Version der *MySQL*-Datenbank gibt es keine Vorgabe, es wurde die aktuellste Version 5.1.53 genutzt.

MyEclipse - Integrierte Entwicklungsumgebung, Subclipse

MyEclipse [33] ist eine integrierte Entwicklungsumgebung (IDE) für Java, die umfangreiche Hilfestellung für den gesamten Entwicklungsprozess eines *Java EE*-Projekts bietet. *MyEclipse* wurde als Erweiterung der *Eclipse*-IDE, die verschiedene Programmiersprachen unterstützt, mit verbesserter Hilfestellung für *Java EE*-Projekte entwickelt. Über Plugins kann der Funktionsumfang der IDE individuell erweitert werden.
Das installierte *Subclipse*-Plugin ermöglicht die integrierte Nutzung von *Subversion* zur Sourcecodeverwaltung. Eine *Ant*-Unterstützung zum Kompilieren, Builden und Deployen des Projektes ist bei *MyEclipse* fest integriert. Über verfügbare Plugins bietet *MyEclipse* auch einfache Reverse Engineering-Fähigkeiten und kann UML-Diagramme zu ausgewählten Software-Modulen generieren. Sämtliche Dateiformate, die aus der Vielzahl der verwendeten Programmiersprachen in diesem Projekt resultieren, werden nativ von *MyEclipse* und seinen Plugins unterstützt.
MyEclipse und die notwendige Lizenz wurden vom *Institut für Genomik und Bioinformatik* in Version 9.1 zur Verfügung gestellt und nach der Installation über den eigenen Updatemechanismus aktualisiert.

Ant - Buildtool

Ant [34] ist ein in Java implementiertes Build-Werkzeug, das über *XML*-Dateien gesteuert Sourcecode kompilieren und beliebige Programme ausführen kann. Da es selbst in Java geschrieben ist, benötigt es eine Java Laufzeit (JRE) Umgebung zum Betrieb. Ant wurde als integrierter Bestandteil von *MyEclipse* in Version 1.7.1 installiert.

Subversion (SVN) - Sourceverwaltungstool

Durch die vielfältigen Funktionen von SVN unterstützt dieses Werkzeug einen Software-Entwickler bzw. ein Team von Softwareentwicklern bei der räumlich getrennten, kontrollierten Verwaltung der entstehenden Sourcecodedateien. Zum Betrieb ist ein SVN-Server notwendig, auf den die verschiedenen Entwickler über SVN-Clients (*zum Beispiel Subclipse*) zugreifen. Der SVN-Client übernimmt lokal die Kontrolle über die zum Projekt gehörenden Dateien. Die Veränderungen an diesen Dateien werden dabei zu bestimmten Zeitpunkten (*checkin*s) protokolliert und die eingepflegten Dateien versioniert. Es ist jederzeit möglich, zu einem späteren Zeitpunkt beliebig alte Versionsstände von Dateien rückzuführen. SVN weist auf Konflikte durch gleichzeitige Änderungen an der gleichen Datei von verschiedenen Entwicklern hin und bietet die Möglichkeit, solche Probleme automatisiert zu lösen (*merge*). Bereits Projekte mit einigen wenigen Dateien profitieren vom Einsatz eines Sourceverwaltungstools wie diesem. Die Version bei Subversion spielte keine Rolle und wurde in Form eines *MyEclipse*-Plugins namens *Subclipse 1.6.x* installiert.

2.2. Reverse Engineering

Die Umsetzung der geplanten Erweiterungen an *SnoWMAn* setzen ein tieferes Verständnis über die zugrunde liegende Softwarearchitektur und angewandte Implementierungsdetails voraus. Aufgrund des Fehlens einer umfassenden Plattform-Dokumentation wird durch sogenanntes Reverse

Engineering das notwendige Wissen aufgebaut. Die vorgenommenen Maßnahmen umfassen folgende Punkte

- genaue Analyse des Sourcecodes ohne Programmausführung
- Einbau von Debug-Meldungen in den Sourcecode, um den Programmablauf bzw. interne Programm-Zustände nachvollziehen zu können
- Einbau künstlicher Exceptions, um an definierten Punkten die Aufruf-hierarchie über den Call-Stack zu ermitteln

Die dabei produzierte Ausgabe-Meldungen gehen an den default-Standard-output-Handler, landen somit in der log-Datei des *SnoWMAn*-Applikations-servers (`jboss4.02/server/default/log/jboss.log`). Dort können sie nach der Protokollierung zur Analyse eingesehen werden.

Diagramme zum Festhalten interner Strukturen von *SnoWMAn* werden ohne Einhaltung einer strengen Beschreibungssprache als Mindmap-Grafiken mittels *XMind* [35] beziehungsweise Dia [36] erstellt.

2.3. Refactoring

Bestimmte wiederkehrende Muster im Sourcecode die sich bei der Code-analyse zeigten und die Analyse erschwerten, machten die Anwendung verschiedener Refactoring-Maßnahmen notwendig. Zielsetzung dieser Maßnahmen war es nicht, grundlegende Umbauten am *SnoWMAn*-System vorzunehmen, sondern die Analyse in Hinblick auf die gewollten Erweiterungen zu erleichtern, beziehungsweise die Erweiterungen zu ermöglichen. Insbesondere durften die Refactoring-Anpassungen am Code keine bestehende Funktionalität des Systems für den Benutzer beeinträchtigen. Aus Benutzersicht blieben die umgesetzten Änderungen unbemerkt.
Bei allen angewandten Änderungen wurde versucht, stets die Lösung mit den geringsten Auswirkungen auf das bestehende System zu wählen. Jede eingeführte Änderungen an einer funktionierenden Software bringt potentiell immer Gefahren (Bugs) mit ein. Insofern erscheint es vernünftig, notwendige Änderungen so gering als möglich ausfallen zu lassen.

2.3.1. Namenskorrektur

Eine der häufigst angewandten Maßnahmen war die Abänderung der Namen von Variablen, Funktionen, Methoden und Klassen. Dabei wurden Bezeichnungen zur Einhaltung einer Namenskonvention angeglichen, Namen zur semantisch klareren Bezeichnung abgeändert und Namen durch gleiche Begriffe vereinheitlicht.

2.3.2. Bereinigung von Schnittstellen

Anmerkung: In folgenden Absätzen wird in Bezug auf *Schnittstellen* aus objektorientierter Sicht der Begriff *Methode* synonym auch für Funktionen verwendet, da die Unterscheidung zwischen einer Methode und einer Funktion bezogen auf die behandelten Themen nicht relevant und für beide Aufrufvarianten das Gesagte gleich bedeutend ist.

Zur Entkopplung der Methoden-Implementierung von Abhängigkeiten an Implementierungsdetails eines übergebenen "Welt"-Objekts, falls nur wenige Detailinformationen des "Welt"-Objekts verarbeitet werden, wurden Schnittstellen dahin gehend geändert, nur die tatsächlich benötigten Objekte zu übernehmen.
Das selbe Prinzip wurde auch bei neu erstellten Schnittstellen entsprechend *hoher Kohäsion* [37] und *geringst möglicher Kopplung* [38] angewandt.

2.3.3. Änderung der Nutzung vorhandener Schnittstellen

In Fällen in denen Objekte an eine Methode übergeben und durch die Methode nicht verarbeitet werden, wurden die Objekt-Referenzen zur Sichtbarmachung der Nichtverarbeitung bewusst auf *null* gesetzt. Nur durch Änderung der Nutzung in diesen Situationen, ohne Änderung der Schnittstelle an sich, wurde die Semantik verdeutlicht und die Überprüfbarkeit von (Objekt-)Bedingungen von der Laufzeit zur Kompilierungszeit verschoben.

Weiters war es teilweise notwendig, bei der Übergabe von Parametern an Methoden *null*-Referenzen durch Referenzen auf default-konstruierte Objekte zu ersetzen.

2.3.4. Erstellung von Sourcecode-Dokumentation

Bei den analysierten Software-Komponenten wurde teilweise eine Sourcecode-Dokumentation nachträglich eingeführt, die Erkenntnisse der Analyse dadurch festgehalten. Alle neu entwickelten Erweiterungen wurden im Code unter Nutzung der *javadoc* [39] Tags dokumentiert. Bei der Dokumentation wurde versucht, immer folgende Struktur anzuwenden.

- Erklärung der beabsichtigten Funktion (der Klasse, Methode, Funktion, Konstruktor, Destruktor und so weiter) durch ein kurzes prägnantes Statement
- Erklärung des Typs und der Funktion aller übernommenen Parameter
- Erklärung des Rückgabetyps
- Erklärung von Exceptions
- Erklärung von Constraints

2.4. Datensätze

Bei den Erweiterungsarbeiten kamen grundsätzlich zwei verschiedene Arten von Datensätzen zum Einsatz. Zum einen kleine Testdatensätze und zum anderen Realdaten zur Generierung von Analyseergebnissen.

2.4.1. Testdatensätze

Die zwei verwendeten Testdatensätze wurden von den unten beschriebenen Realdaten abgeleitet und mit bestimmten Absichten festgelegt. Wie der Name schon sagt, wurden diese Datensätze für Tests während der Entwicklung eingesetzt. Es ging dabei um die Nutzung realer Daten mit geringem

Umfang, damit die Berechnungszeiten für die Tests möglichst kurz gehalten und die funktionelle Korrektheit der Erweiterungen überprüft werden konnte.
Ein bloßes Beschneiden von Realdaten im Umfang alleine war nicht ziel führend, mussten beispielsweise beim Chimerafiltering Daten benutzt werden, die auch wirklich Chimären enthielten. Beziehungsweise bei Tests in der *no-splitting*-Variante von Pipelines die Eingabedaten nach Samples getrennt, und nicht in einer gemeinsamen Datei vorliegen. Diese Anforderungen waren nicht schwer zu erfüllen, mussten aber beachtet werden, da ansonsten die Tests keine brauchbaren Ergebnisse lieferten oder aber Probleme auftraten, die nichts mit den neu eingeführten Erweiterungen zu tun hatten.
Der erste Datensatz *vfew_small2_* enthielt 27 zufällig gewählte Sequenzen aus dem Realdatensatz aus Kapitel 2.4.2 mit genau einer bekannten Chimäre. Er beinhaltete Samples die mit fünf verschiedenen Tags (*A01*, *A03*, *A04*, *C01*, *C04*) markiert und in zwei FASTA-Dateien (*vfew1_.fa*, *vfew2_.fa*) aufgeteilt waren, zwei .txt-Dateien (*vfew1_.txt*, *vfew2_.txt*) welche die Tag-Sequenzen beinhalteten. Komplementiert wurden die Daten durch eine Datei (*vfew1__primers.fa*) in der die Sequenzen für die beiden verwendeten Forwardprimer gespeichert waren und einer .qual-Datei (*vfew1_.qual*). Dieser Datensatz wurde für alle Pipelines mit aktivierter *merging*-Option verwendet.
Der zweite Testdatensatz (*vfewSmallSplitted*) wurde aus dem ersten weiter reduziert und beinhaltete insgesamt nur 13 Sequenzen, im Gegensatz dazu waren diese nicht in zwei sondern pro Tag in einer eigenen Datei (*A01.fa*, *A03.fa*, *A04.fa*, *C01.fa*, *C04.fa*) gespeichert. Dieser Datensatz wurde für alle Pipelines mit *no-merging*-Option genutzt.

2.4.2. Realdaten

Um letztendlich die eingebauten Erweiterungen verifizieren zu können, wurde mit jeder der fünf verfügbaren Pipelines mit und ohne aktiviertem *Chimerafiltering* der selbe Realdatensatz analysiert. Bei diesem Datensatz handelt es sich um Daten aus einer Studie des Darmmikrobioms, wobei die 16S-Regionen V1-2 sequenziert wurden [40].

Der Datensatz ist 52,1 MB groß und beinhaltet zirka 515000 Sequenzen von 22 unterschiedlichen Samples.

3. Ergebnisse

3.1. Aufbau eines Testsystem

Shutdowns, Reboots und Moduldeaktivierungen machten den Aufbau eines parallelen Test-Systems von *SnoWMAn* ähnlich dem Live-System mit gleicher Funktionalität notwendig. Für die Entwicklung der Erweiterungen war wichtig, uneingeschränkt mit der gleichen Umgebung wie im Live-System arbeiten zu können. So wurde das *JClusterService*-System am Rechencluster zur Ausführung der benötigten *SOAP*-gesteuerten Berechnungs-Services "geklont" und ein eigener Applikationsserver (jboss 4.02) am lokalen PC in Betrieb genommen. Dieses parallele System wurde für sämtliche Arbeiten während der Erweiterungsarbeiten heran gezogen, es gab somit die geringst mögliche Beeinflussung des Live-Systems. Abgesehen von einigen Details bei der Installation, die durch try-and-error erarbeitet werden mussten, bestand die größte Herausforderung in der richtigen Konfiguration, so dass der lokale Applikationsserver die richtige, geklonte *JClusterService*-Instanz zur Ausführung der Berechnungsaufgaben im Hintergrund benutzte.

Ein hoher Aufwand beim Aufbau des Testsystems entstand durch den Wechsel der Entwicklungsplattform von Windows (*XP*) auf Linux (*OpenSuse 11.4*). Obwohl *SnoWMAn* als *Java EE*-Plattform nicht an ein bestimmtes Betriebssystem gekoppelt ist, gibt es im bestehenden Projekt viele konkrete Abhängigkeiten, Konfigurationen, die durch das verwendete Betriebssystem festgelegt sind.

3.2. Beschreibung der internen Strukturen und Abläufe von *SnoWMAn*

Der gesamte Sourcecode der Anwendung hat einen Umfang von 134270 Lines of Code (loc), siehe Tabelle A.1, es sind in der Implementierung neun verschiedene Softwaretechnologien zu einem großen Ganzen zusammengeführt worden.

Beim Reverse Engineering sind einige unterschiedliche Strukturdiagramme zu verschiedenen Teilen der *SnoWMAn*-Plattform entstanden, die hier aufgelistet werden.

In Abbildung 3.1 wird der grundsätzliche Aufbau der verteilten *SnoWMAn*-Plattform dargestellt.

Die einzelnen Pipelines innerhalb von *SnoWMAn* können bei Bedarf auch als Commandline-Programme ausgeführt werden. Für das Handling der übergebenen Parameter zur Steuerung einer Pipeline wird das Apache Commons Command Line Interface (CLI) [41] eingesetzt, siehe Abbildung 3.2.

Die Erarbeitung des softwaretechnischen Aufbaus der einzelnen Pipelines resultierte in einem Strukturdiagramm das als Ganzes schwer auf einer A4-Seite darstellbar ist. Der Vollständigkeit halber wird es hier angeführt, siehe Abbildung B.3. Bei den Arbeiten wurde es interaktiv mittels *XMind*-Software verwendet.

Die fünf in der Web-Oberfläche nutzbaren Pipelines (*BLAT*, *JGAST*, *Mothur*, *RDP*, *UCLUST*) sind mittels sechs Klassen implementiert. Die *RDP*-Pipeline wurde dabei in der Software durch zwei Klassen umgesetzt, eine für mit und ohne Merging. Was die zugrunde liegende Software-Architektur der Pipeline-Klassen betrifft, folgen alle außer der *mothur*-Pipeline der selben (neuen) *Pipeline-Command*-Struktur, siehe Kapitel 3.4. Die *mothur*-Klasse ist als monolithische Klasse mit allen notwendigen "Kommandos" in Form von eigenen Methoden programmiert. Tabelle 3.1 zeigt die Zuordnung der Pipelines zu ihren Klassen.

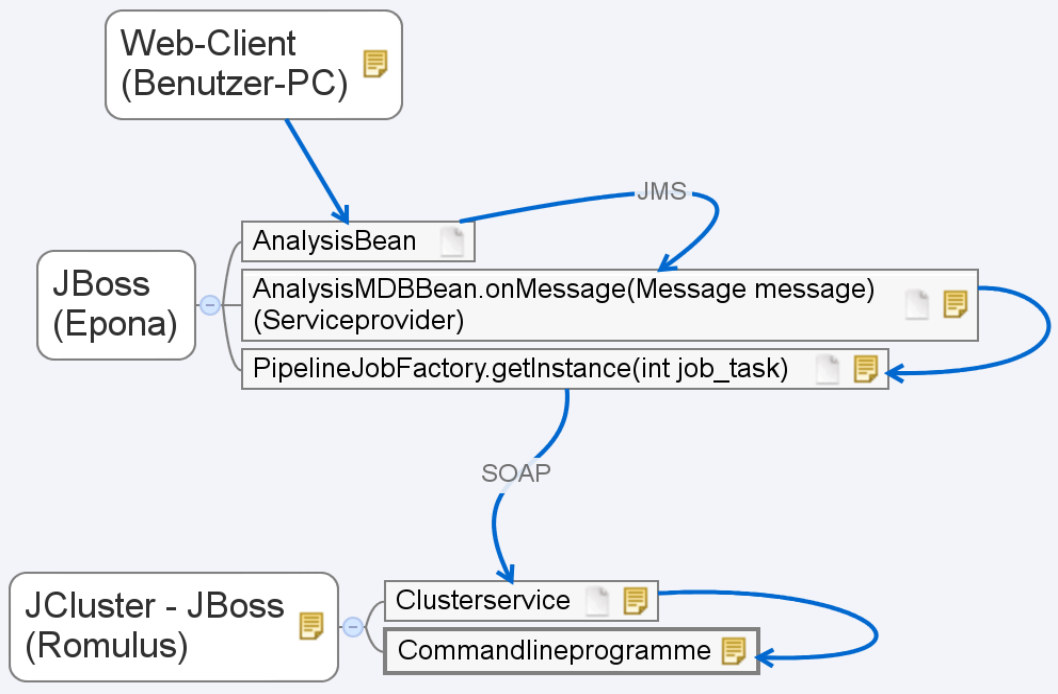

Abbildung 3.1.: Wesentliche Teile die beim Starten einer neuen Analyse durch den Benutzer im Browser (*Web-Client*) zusammen spielen, die Objekttypen (Klassennamen) der beteiligten Objekte, die wichtigsten Methoden, die verwendeten Protokolle und die Zuordnung der Komponenten zu Computern, *Epona* ist der Name des Applikationsserver, *Romulus* bezeichnet den genutzten Rechencluster

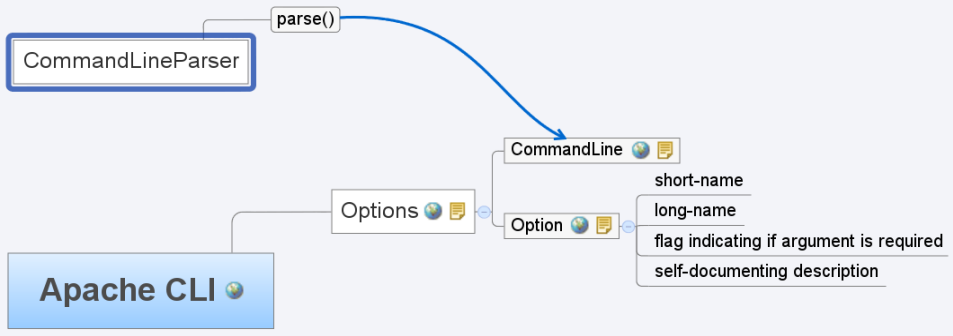

Abbildung 3.2.: In *SnoWMAn* eingesetztes *Apache CLI* zum Commandline-Parameter-Handling, dargestellt sind die grundsätzlichen *Apache CLI*-Objekte, -Methoden, -Properties und deren Beziehung zueinander

Tabelle 3.1.: Auflistung der Pipeline-Klassen Zuordnung, alle Klassen sind Teil des *at.tugraz.genome.biojava.cli.mbiom.cmd*-Packages

Pipeline	**Klassenname**	**Beschreibung**
Mothur	CombinedMothurPipeline.java	monolithische Struktur, gesamter Pipeline-Code in einer Klasse implementiert, für jedes "Kommando" der Pipeline eigene Methode
RDP (Default)	GlobalRefOTUPipeline.java	RDP-Pipeline mit "merging" *Pipeline-Command* Struktur
RDP "without merging"	CombinedRDPPipeline.java	RDP-Pipeline ohne merging *Pipeline-Command* Struktur
BLAT	BlatTaxonomyPipeline.java	*Pipeline-Command* Struktur
JGast	JGastPipelineUnifiedOutputCommand.java	*Pipeline-Command* Struktur
UCLUST	GlobalUclustRefOTUPipeline.java	*Pipeline-Command* Struktur

3.3. Änderung der Nutzung vorhandener Schnittstellen

An einigen Stellen wurde durch bewusste Übergabe von *null*-Referenzen anstatt Referenzen auf unbenutzte Objekte an Methoden die Aussagekraft des Sourcecodes verstärkt und die Analyse dadurch vereinfacht. Stellvertretend für diese Fälle wird hier ein konkreter Sourcecodeauszug aus der Klasse `at.tugraz.genome.biojava.cli.mbiom.cmd.mothur.CombinedMothurPipeline` angeführt.
Vorher:

```
String ret = runTagBasedClusterJob(command, cmdlineOptions, outputInputMappings, params, true, STEP_CHIMERA);
```

Nachher:

```
cmdlineOptions = null; // (JHo) because this parameter is not used in runTagBasedClusterJob
final boolean ENABLE_PICKUP = true;
String ret = runTagBasedClusterJob(command, cmdlineOptions, outputInputMappings, params, ENABLE_PICKUP, STEP_CHIMERA);
```

In Fällen in denen *null*-Referenzen übergeben wurden, obwohl die Verarbeitung innerhalb der Methode default-konstruierte oder pseudo-initialisierte Objekt-Referenzen verlangte, wurde auf die richtig konstruierten Objekt-Referenzen umgestellt. Stellvertretend ein Auszug aus der Klasse `at.tugraz.genome.mbiom.snowman.web.actions.workflow.IGBStandardWorkflow`:
Vorher:

```
pip.setupPipelineImplementation(null, null);
```

Nachher:

```
Options options = new Options();
String[] arguments = new String[0];
if (mode.contains(this.MODE_CHIMERACHECKING))
{
  this.buildChimeracheckingOption(options);
  arguments = new String[]{ "-" + GlobalRefOTUPipeline.OPTION_CHIMERACHECKING_DATABASE };
}
CommandLineParser parser = new BasicParser();
CommandLine command = null;
try
{
  command = parser.parse(options, arguments);
```

```
}
catch (Exception exception)
{ // Nothing to do, this case should not occur in the current case
}
options = null;
pip.setupPipelineImplementation(command, options);
```

3.4. Adaptierung der internen SW-Struktur zur Ausführung von Pipelines

Die auf der *SnoWMAn*-Plattform durch den Benutzer wählbaren Pipelines sind im Sourcecode durch eigene Klassen repräsentiert. Instanzen dieser Klassen nutzen während der Ausführung wiederum sogenannte *Command*-Instanzen für die konkrete Ausprägung einer Pipeline aufgrund der Benutzerentscheidung im Webinterface. Eine *Command*-Instanz stellt eine kompakte Verarbeitungseinheit, zum Beispiel repräsentativ für das Chimerafiltering (*ChimeracheckingCommand*), innerhalb des Systems dar die sich um alle Kommando-relevanten Aufgaben wie Datentransfer zum und vom *JClusterService*, Datei-Handling und so weiter kümmert.

Chimerafiltering als wählbare Option in das System einzuführen bedeutet, unter anderem die Pipeline-Instanzen entsprechend der Benutzerentscheidung Gebrauch von Chimerafiltering-*Command*-Instanzen machen zu lassen. Genau bei dieser Zuordnung von Pipeline-Instanzen zu *Command*-Instanzen gab es erhebliche Probleme mit der vorhandenen Implementierung.

Obwohl Pipeline-Instanzen und *Command*-Instanzen dynamisch, aufgrund der Benutzerentscheidung im Webinterface zur Laufzeit, zueinander in Beziehung stehen sollten, waren diese Assoziationen im ursprünglichen Code starr kodiert. Vor der vorliegenden Erweiterung beeinflussten Benutzerentscheidungen nicht die Reihenfolge von Kommandos in Pipelines, sondern nur die Ausprägung von Kommandos. Die *Command*-Reihenfolgen war starr umgesetzt und die flexible Behandlung dieser in der Implementierung ausgespart. Die Anforderung eines optional wählbaren Chimärafilterings stand im direkten Widerspruch mit der Festlegung in der Implementierung und konnte nicht direkt umgesetzt werden. Die existierende Softwarearchitektur

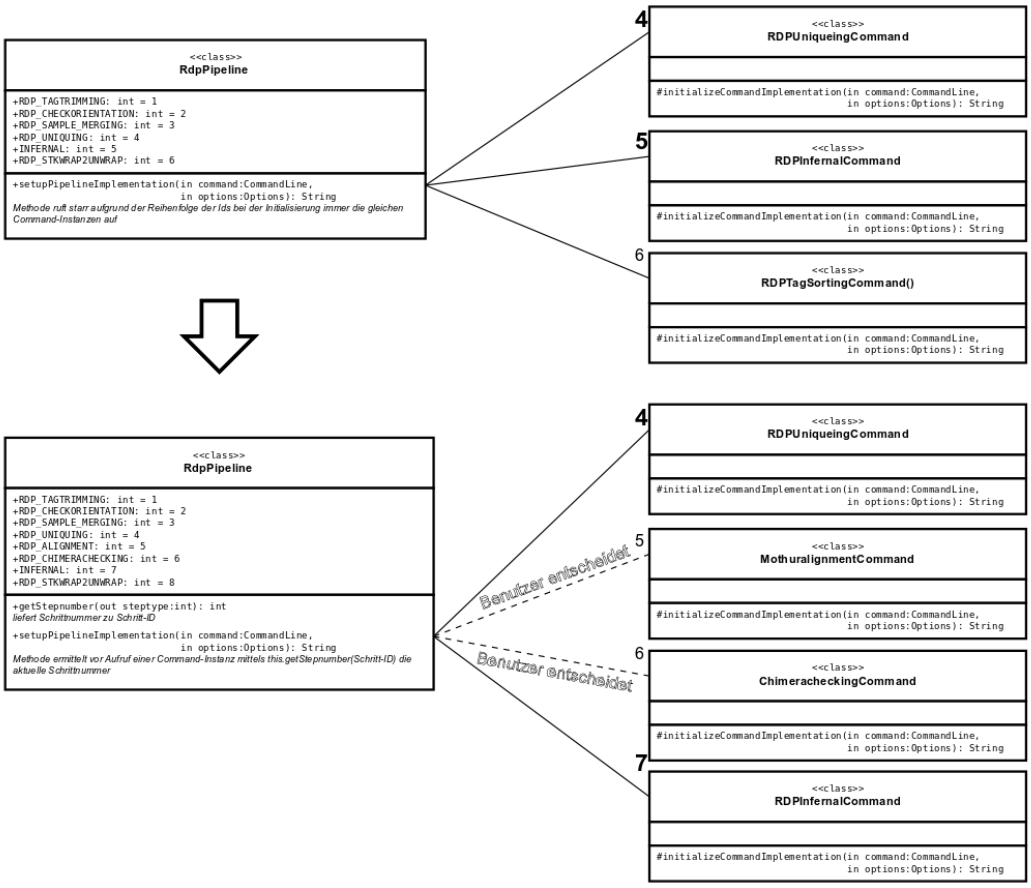

Abbildung 3.3.: (*RDP*-)Pipeline-*Command* Relation vor und nach der Adaptierung, nach der Adaptierung können *Command*s dynamisch verwaltet werden, die Ausführungsreihenfolge der *Command*s wird erst zur Laufzeit bestimmt und ist nicht mehr starr im Sourcecode kodiert

musste adaptiert werden, so dass von Pipeline-Instanzen genutzte *Command*-Instanzen flexibel gereiht und aufgerufen werden konnten, Abbildung 3.3 stellt den Unterschied dar.

Alle *Command*-Instanzen waren intern mit einer eigenen vordefinierten ID versehen. Diese IDs wurden genutzt und über einen neu geschaffenen Mapping-Mechanismus auf tatsächliche Positionen in der *Command*-Reihenfolge innerhalb einer Pipeline-Instanz verlinkt.

In der ursprünglichen Implementierung wurden *Command*-Instanzen immer über ihre IDs angesprochen, für die Erweiterung mussten alle diese Aufrufe im Code gefunden und durch die Anfrage beim neuen Mapping-Mechanismus ersetzt werden, damit die neue Indirektion greifen konnte. Bei diesem Umbau kam erschwerend die frühere Festlegung der IDs auf Klassen-Ebene und nicht auf Objekt-Ebene hinzu, wodurch für die Anfragen an den Mapping-Mechanismus häufig kein passender Objekt-Kontext vorlag. Letztendlich konnte dieser aber an allen notwendigen Stellen hergestellt werden.

3.5. Neues Service am *JClusterService*-Server

Die Implementierung des Chimerafilterings bedingte die Einführung eines neuen Services am *JClusterService*-Server. Das neue Service wurde in die bestehenden eingegliedert und kann über die existierende *JClusterService*-API genutzt werden. Die Services werden über die Datei `clusterservice-definition.xml` konfiguriert. Konkret wurde *MOTHURCHIMERA* zum Filtern der Chimären mit folgender Konfiguration hinzugefügt.

```
  <clusterservice name="MOTHURCHIMERA"
   program-path="/usr/local/bioinf/bin/repomanager_wrapper.sh -C /usr/local/bioinf/bin/mothur_chimerauchime.sh -P r
-R /netapp/BioInfo/MicroBiom/databases"
   nodes="1" validate="true">
    <description>
      Uses mothur uchime chimera checking for removing chimeras in
      sequencedata:

      Basic usage
      -i fasta-inputfile
      -a accno-ouputfile
      -c chimera-outputfile
      -o chimeraremoved-fasta-outputfile
      -r reference-inputfile
      -n names-inputfile
      -g groups-inputfile
      -p num of processes
     </description>
    <parameterlist>
      <parameter name="inputfile1" switch="-i" position="0" mandatory="true">
        <parameter-value isdefault="true" type="inputfile" name="inputfile2" />
        <parameter-value type="resultfile" externalservice="MUSCLE" name="default"/>
        <parameter-value type="resultfile" externalservice="MOTHURALIGN" name="default"/>
      </parameter>
      <parameter name="accnofile1" switch="-a" position="1" mandatory="true">
        <parameter-value isdefault="true" type="resultfile" name="accnofile2" />
      </parameter>
      <parameter name="chimerarefdatabase" switch="-D" position="2" mandatory="false">
<!--        (JHo) Defaultvalues MUST CORRESPOND with repositorymanager managed (chimerachecking reference)
database entry! -->
        <parameter-value isdefault="true">chimerarefdbs</parameter-value>
      </parameter>
      <parameter name="chimerarefdatabaseversion" switch="-V" position="3" mandatory="false">
        <parameter-value isdefault="true">09-Feb-2012_16S_aligned</parameter-value>
      </parameter>
<!--      <parameter name="referencefile" switch="-r" position="4" mandatory="false"> -->
<!--      </parameter>-->
      <parameter name="chimerafile1" switch="-c" position="4" mandatory="true">
        <parameter-value isdefault="true" type="resultfile" name="chimerafile2" />
      </parameter>
      <parameter name="outputfile1" switch="-o" position="5" mandatory="true">
        <parameter-value isdefault="true" type="resultfile" name="outputfile2" />
      </parameter>
      <parameter name="namesfile" switch="-n" position="6" mandatory="false">
      </parameter>
      <parameter name="groupsfile" switch="-g" position="7" mandatory="false">
      </parameter>
      <parameter name="numprocessor" switch="-p" position="8" mandatory="false">
      </parameter>
      <parameter name="uniquefile" switch="-u" position="9" mandatory="true">
        <parameter-value isdefault="true" type="inputfile" name="uniquefiledefault" />
        <parameter-value type="resultfile" externalservice="MOTHURUNIQ" name="default"/>
        <parameter-value type="resultfile" externalservice="RDP_CHECKORIENTATION" name="resultfile"/>
      </parameter>
      <parameter name="uniquenamefile" switch="-v" position="10" mandatory="true">
        <parameter-value isdefault="true" type="inputfile" name="uniquenamefiledefault" />
        <parameter-value type="resultfile" externalservice="MOTHURUNIQ" name="nameoutput"/>
      </parameter>
```

```
<!--      (JHo) Although I don‘t need the following two paramets I have introduced them because without
the datahandling didn‘t work correct.   -->
      <parameter name="pseudo1" switch="-d" position="11" mandatory="false">
        <parameter-value isdefault="true" type="resultfile" name="filteredUniquingFastafile" />
      </parameter>
      <parameter name="pseudo2" switch="-e" position="12" mandatory="false">
        <parameter-value isdefault="true" type="resultfile" name="filteredUniquingNamefile" />
      </parameter>
    </parameterlist>
    <inputfilelist>
      <inputfile name="inputfile2">##-STD_INPUTFILE-##.fa</inputfile>
      <inputfile name="uniquefiledefault">##-STD_INPUTFILE-##.unique.fa</inputfile>
      <inputfile name="uniquenamefiledefault">##-STD_INPUTFILE-##.unique.name</inputfile>
    </inputfilelist>
    <resultfilelist>
      <resultfile name="accnofile2">##-RESULTFILE-##.accno</resultfile>
      <resultfile name="chimerafile2">##-RESULTFILE-##.chimera</resultfile>
      <resultfile name="outputfile2">##-RESULTFILE-##.uchime.fa</resultfile>
      <resultfile name="filteredUniquingFastafile">##-RESULTFILE-##.unique.filtered.fa</resultfile>
      <resultfile name="filteredUniquingNamefile">##-RESULTFILE-##.unique.filtered.name</resultfile>
    </resultfilelist>
  </clusterservice>
```

Das Chimerafiltering benötigt seinerseits ein sequenzbasiertes Alignment als Vorgängerschritt, damit die Eingabedaten aligned verarbeitet werden können. Dies konnte in Form des bereits existierenden Services *MOTHURALIGN* umgesetzt werden.
Die tatsächliche Berechnungsarbeit hinter dem neuen Chimerafiltering-Service wird durch das Chimerafiltering-Command *chimera.uchime* [42] des *mothur*-Tools in Version 1.23.1 ausgeführt. Wenn man so will, ist das eingeführte *MOTHURCHIMERA-JClusterService*-Service eine Indirektion, um das Chimerafiltering-Command des *mothur*-Tools ansprechen zu können. Die *mothur*-Version 1.23.1 war zum Zeitpunkt der Implementierung die aktuellste die verfügbar war. Sie wurde in Form des Sourcecodes von der Projektseite [43] als 64 bit-Version runter geladen, mit Threading- und *MPI*-[44] Unterstützung kompiliert und am Rechencluster unter */usr/local/bioinf/bin/mothurmpi.1.23.1* installiert. Aufgrund einiger kleinerer Fehler des Tools bei der Eingabedatenverarbeitung und der Benennung von Ausgabedateien waren Workarounds in Form eines selbst geschriebenen *bash*-Skriptes notwendig. Das *mothur*-Tool ist dadurch nicht mehr direkt mit dem *JClusterService* "verbunden", sondern in eine eigene *bash*-Hülle eingebunden. Durch den *bash*-Code werden die *mothur*-Probleme ausgemerzt und der Datenaustausch mit dem *JClusterService* berichtigt.

3.6. Erweiterung aller Pipelines um Chimerafiltering-Option

Die erläuterte Entkopplung der fixen Bindung zwischen Pipeline-Instanzen und ihren verwendeten *Command*-Instanzen ermöglichte erst die Einführung der neuen Chimerafiltering-Option. Alle vorhandenen Pipelines wurden damit erweitert, der Benutzer kann im Webfrontend diese per Maus auswählen, siehe Abbildung 3.4. Wird Chimerafiltering aktiviert muss die dafür

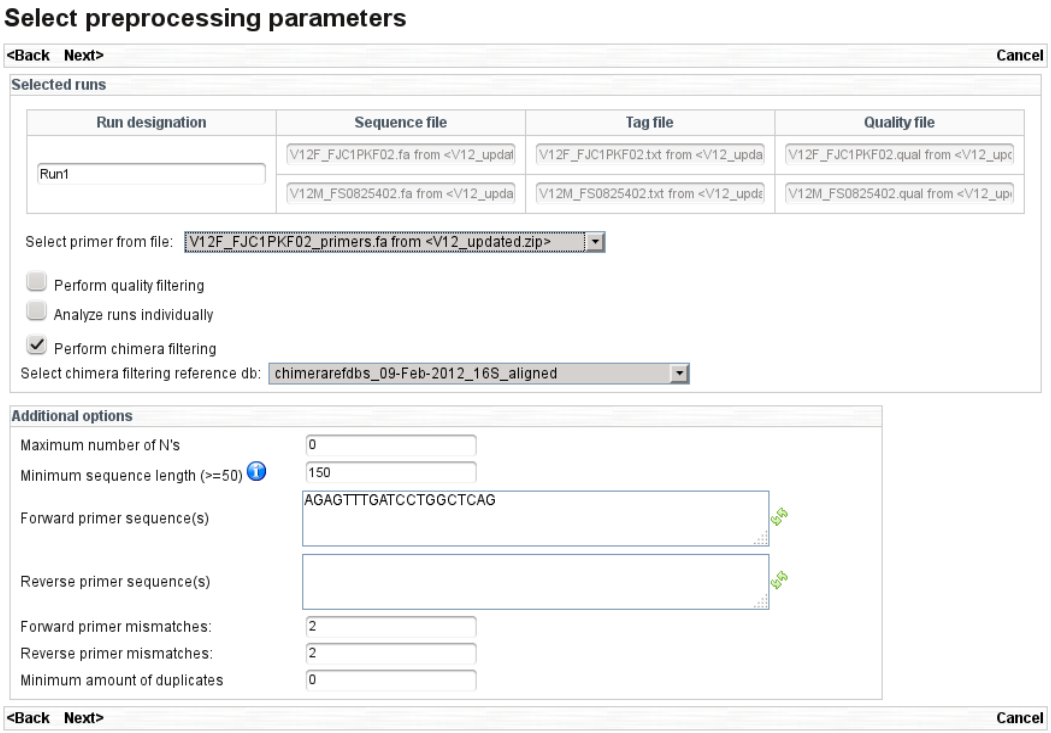

Abbildung 3.4.: Option zum Chimerafiltering (in der *RDP*-Pipeline), die Auswahlmöglichkeit *Perform chimera filtering* und das zugehörige Drop-Down-Menü für die Referenz-Datenbank wurde in der Web-Benutzeroberfläche eingeführt

verwendete Referenzdatenbank mittels eines Dropdown-Menüs festgelegt werden. Alle Datenbanken die zur Auswahl stehen sind am Rechencluster

durch den *Repositorymanager* verwaltet. Dies ist eine Administrationssoftware für Flatfile Datenbanken [45]. Wird über den Repositorymanager eine neue Datenbank in die (gleiche) Gruppe *CHIMERA* hinzugefügt erscheint diese ohne weitere Anpassung automatisch im Webfrontend zur Auswahl. Die Referenz-Datenbanken sind am Rechencluster unter */netapp/databases/MicroBiom/databases/chimerarefdbs* gespeichert.

3.7. Pipeline-Analyseergebnisse mit Chimerafiltering

In den zu analysierenden Sequenzdaten enthaltene Chimären erscheinen als einzigartige Sequenzabschnitte und führen zur Bildung zugehöriger OTUs. Da diese Sequenzabschnitte nicht biologisch begründet sind, verfälscht die erhöhte OTU-Anzahl die Analyseergebnisse, sogenanntes OTU-*inflation* tritt auf. Chimerafiltering wirkt dem OTU-*inflation* entgegen und korrigiert die OTU-Anzahl nach unten.

Das angewandte Chimerafiltering führte bei den Tests mit dem V12-Datensatz unabhängig von der gewählten Pipeline zu einer Reduzierung der als eindeutig erkannten Sequenzen von mindestens -3,10% (*JGAST*, *UCLUST*, *Mothur*: -3,43%, *RDP*: -3,70%, *BLAT*: -3,10%). Die Reduzierung der Anzahl erkannter OTUs variierte je nach Pipeline und nach zugrunde gelegtem Ähnlichkeitsmaß. Die höchste durch Chimerafiltering erreichte Reduzierung trat in der *UCLUST*-Pipeline beim Ähnlichkeitsmaß 95% mit -15,15% auf.

Tabelle 3.2 zeigt die minimalen und maximalen relativen OTU-Reduzierungen aufgrund des Chimerafilterings in den verschiedenen Pipelines. Die detaillierten Daten für die einzelnen Pipelines können in den Tabellen im Anhang A.2 eingesehen werden.

Tabelle 3.2.: Auflistung der minimalen und maximalen relativen OTU-Reduzierungen der *SnoWMAn-Pipelines* aufgrund von Chimerafiltering

Pipeline	**minimale relative OTU-Reduzierung**	**maximale relative OTU-Reduzierung**
JGast	-1,67% (RDP)	-3,43% (Unique Seqs)
UCLUST	-3,43% (Unique Seqs)	-15,18% (0,05)
Mothur	-3,43% (Unique Seqs)	-12,29% (0,06)
RDP	-3,70% (Unique Seqs)	-7,83% (0,05)
BLAT	0,00% (NCBI)	-4,60% (Ludwig)

3.8. Denoising

Ähnlich wie beim Chimerafiltering führt Noise in Form von Homopolymerfehlern in den Sequenzdaten zu einer erhöhten OTU-Anzahl und dadurch zum OTU-*inflation*. OTU-*inflation* hat immer die schlechtere Abbildung der biologischen Realität zur Folge. Denoising wirkt der OTU-*inflation* entgegen und korrigiert die OTU-Anzahl nach unten.

Zum Denoising wurde das *shhh.seqs* Command [46] des *mothur*-Tools [22] eingesetzt. In *mothur* ist der Denoising Algorithmus *SeqNoise* von Christopher Quince [21] implementiert.

Das Denoising des V12-Datensatzes dauerte am genutzten Rechencluster des *Instituts für Genomik und Bioinformatik* 12 Tage. Aufgrund dieses hohen Zeitbedarfs für einen einzelnen Schritt in der Verarbeitung einer Pipeline wurde entschieden, die Denoising-Option vorerst nicht in *SnoWMAn* einzuführen, auf die Implementierung zu verzichten. Diese Option hat den akzeptablen Zeitbedarf einer gesamten Analyse für den Benutzer deutlich überschritten.

Um die Auswirkungen des Denoisings trotzdem bewerten zu können, wurde manuell auf der Commandline der V12-Datensatz denoised und anschließend das Ergebnis mit der *RDP*-Pipeline (mit Chimerafiltering)

in *SnoWMAn* analysiert. Somit konnten die Analyseergebnisse der *RDP*-Pipeline von denoised versus undenoised chimera-gefilterten Daten verglichen werden, siehe Tabelle A.7.

Durch das Denoising wurden die *Unique Sequencies* um zwei Drittel auf ein Drittel reduziert, von 66648 Sequenzen in *V12_updatedRDPWithChimeraAfterStatcorrection* auf 21943 Sequenzen in *V12_shhh_FastaExtended_ChimeraCheckedRDP* (-67,1%). Die größte Reduzierung der OTU-Anzahl tritt bei Distanz *0,0* mit -52,5%, die kleinste bei Distanz *0,06* mit -33,68% auf.

Die Analysen wurden zu unterschiedlichen Zeitpunkten durchgeführt, aufgrund des Implementierungsfortschritts stand Chimerafiltering auf der Live-Instanz nicht zur Verfügung. Deshalb wurden die denoised Sequenzdaten für die Live-Instanz manuell über die Konsole chimera-gefiltert. Das Verhältnis der Gesamtlaufzeiten des Entwicklungs-PCs (09:22:13, in hh:mm:ss) und dem Rechnencluster - Live-Instanz (01:11:16) beträgt 7,89 zu 1.

Die Laufzeit-Statistik zu den beiden *RDP*-Pipeline-Analysen ist in der Tabelle 3.3 einzusehen.

Tabelle 3.3.: Laufzeit-Statistik Denoising - *RDP*-Pipeline mit Chimerafiltering, Vergleich Analysen ohne Denoising (*V12_updatedRDPWithChimeraAfterStatcorrection*) am Entwicklungs-PC und mit Denoising (*V12_shhh_FastaExtended_Chimera-CheckedRDP*) am Live-System, Denoising-Schritt wurde nicht über *SnoWMAn*-Plattform ausgeführt, sondern manuell auf der Konsole, deshalb sind keine Zeitangaben dafür in den Daten

V12_updatedRDPWithChimeraAfterStatcorrection (PC)				
Sample splitting	506218	Sequences	10:40 AM	00:03:19
Trimming and Filtering	485287	Sequences	10:44 AM	00:03:34
Orientation check	435669	Sequences	10:47 AM	00:02:39
Merging	435669	Sequences	10:50 AM	00:02:23
Uniquing	435669	Sequences	10:52 AM	00:00:45
Aligmnent-Moth	69212	Sequences	10:53 AM	01:34:15
Chimerafiltering	69212	Sequences	12:27 PM	00:53:44
Aligmnent-Infern	66648	Sequences	1:21 PM	00:18:22
Clustering	66648	Sequences	1:43 PM	06:00:17
Dereplication	73573	Clusters	7:43 PM	00:04:37
Classification	73573	Sequences	7:48 PM	00:06:56
Rarefaction			7:55 PM	00:10:37
Summary	73573	Clusters	8:05 PM	00:00:26
OTUDistribution			8:06 PM	00:00:19
Summe				09:22:13

V12_shhh_FastaExtended_ChimeraCheckedRDP (Rechencluster)				
Sample splitting	459390	Sequences	12:21 PM	00:00:50
Trimming and Filtering	455873	Sequences	12:22 PM	00:00:50
Orientation check	452363	Sequences	12:23 PM	00:00:47
Merging	452363	Sequences	12:23 PM	00:00:50
Uniquing	452363	Sequences	12:24 PM	00:00:43
Aligmnent	21943	Sequences	12:25 PM	00:05:20
Clustering	21943	Sequences	12:31 PM	00:55:28
Dereplication	38095	Clusters	1:26 PM	00:00:40
Classification	38095	Sequences	1:27 PM	00:03:25
Rarefaction			1:31 PM	00:01:47
Summary	38095	Clusters	1:33 PM	00:00:21
OTUDistribution			1:34 PM	00:00:15
Summe				01:11:16

4. Diskussion

4.1. Rückblick

Bei der vorliegenden Masterarbeit wurde nach Aufbau eines parallelen Test-Entwicklungssystems der bestehenden *SnoWMAn*-Plattform des *Instituts für Genomik und Bioinformatik* durch Reverse Engineering die notwendige Wissensbasis geschaffen, um die gewünschten funktionellen Erweiterungen umsetzen zu können. Aufgrund der Konzeption von *SnoWMAn* als *Java EE*-Plattform wurde die Abhängigkeit der Konfiguration an das verwendete Betriebssystem beim Wechsel der Systemumgebung des Entwicklungssystems von *Windows XP* auf Linux unterschätzt, letztendlich aber erfolgreich bewältigt.
Mittels Software-Refactoring sind Hindernisse, die das Verständnis über Zusammenhänge behinderten, oder der technischen Erweiterung im Weg standen, ausgeräumt worden. Nach Abschluss dieser notwendigen Eingriffe wurden alle fünf Pipelines (*BLAT*, *JGAST*, *Mothur*, *RDP*, *UCLUST*) um die Chimerafiltering-Option erweitert und alle System-Module angepasst, damit der Benutzer diese neue Funktion über das Web-Interface nutzen kann. Statistische Auswirkungen des Chimerafilterings auf die Analysen sind in allen Pipelines erfasst und ausgewertet worden.

Von der ursprünglich geplanten Implementierung einer Denoising-Funktion zur Reduzierung von Homopolymer-Fehlern in den Eingangs-Sequenzdaten wurde aufgrund der verfügbaren Rechenleistung des genutzten Rechenclusters vorerst abgesehen. Zur Beurteilung der (positiven) Auswirkung des Denoisings auf die Analyseergebnisse bezüglich OTU-Häufigkeiten sind Eingangs-Sequenzdaten (“manuell”) auf der Commandline denoised und anschließend exemplarisch mit der *SnoWMAn-RDP*-Pipeline analysiert worden.

Auf einige der angeführten Themen wird im Anschluss genauer eingegangen.

4.2. Refactoring-Patterns

Beim durchgeführten Reverse Engineering und dem nachfolgenden Refactoring an Teilen des *SnoWMAn*-Codes zeigten sich bestimmte wiederkehrende Problemmuster die hier aufgelistet werden. Die zugrunde liegenden Ursachen und deren Auswirkungen werden analysiert, um daraus für zukünftige Arbeiten Ansatzpunkte für Verbesserungen zu liefern.

4.2.1. Softwareentwicklungsprozess - Programmierung

Etablierung von Coding Guidelines

Das *SnoWMAn* Projekt ist über mehrere Jahre von unterschiedlichen Entwicklern erarbeitet worden. Analysiert man den vorliegenden Code kann dieser Umstand deutlich am Sourcecode "abgelesen werden". Unterschiedliche Teile des Projektes zeigen unterschiedliche Stile in der Programmierung. Dies beginnt bei der Art der Benennung von Variablen, Kürzel versus ausgeschriebene Tokens, und setzt sich über die Gewohnheit Methoden-Aufrufe einzusetzen, standardmäßige Mehrfachverschachtelungen von Methodenaufrufen innerhalb des eigenen Objektes mit Übergabe immer mehr Default-Parametern, fort.
Aus Projektsicht ist ein möglichst homogener Kodierstil wünschenswert, da dieser in Zukunft erheblich die Wartung eines Projektes erleichtert, und somit entstehende Kosten mindert. Gleichzeitig muss den individuellen Eigenheiten der einzelnen Entwickler Rechnung getragen werden. Um dieses Thema besser kanalisieren zu können, ist die Etablierung und geprüfte Einhaltung projektweiter Coding Guidelines unumgänglich. Will man beim fertigen Produkt dem Sourcecode eine einheitliche Linie in einem Team umsetzen, muss man bei dessen Erstellung dafür Sorge tragen, jeden Mitarbeiter die selben Kodiervorgaben verständlich zu machen. Nur wenn ein Entwickler den Sinn dieser Vorgaben und seine Auswirkungen auf das

gesamte Projekt versteht, wird er in der täglichen Umsetzung diese auch anwenden wollen. Da die Verpflichtung zur Selbstkontrolle unter gleichberechtigten Mitgliedern eines Entwicklerteams häufig nicht funktioniert, besonders bei Streitfällen wo die subjektiven Einschätzungen auseinander gehen, bedarf es einer Überwachung der Einhaltung von Guidelines durch die Projektleitung. Nur mit einem funktionierenden Codingstandard, der von den Teammitgliedern akzeptiert und von der Projektleitung in der Umsetzung überprüft wird, kann eine homogene Sourcecodequalität, die mehr oder weniger unabhängig vom individuellen Entwickler ist, gewährleistet werden. Versäumnisse in diesem Bereich können später nur mit sehr hohem Aufwand korrigiert werden, bedeutet dies doch meistens eine Neuimplementierung.

Projekteinheitliche Schnittstellen-Designvorgaben

Ein Schnittstelle einer Klasse muss von außen nach innen, und nicht umgekehrt von innen nach außen, entworfen werden. Das heißt, die Anforderungen, das Was, entscheidet über ihre konkrete Ausprägung und nicht das Wie. Schnittstellen sollten eine hohe Kohäsion, Robustheit gegen Implementierungsänderungen und eine möglichst geringe Kopplung, Abhängigkeit von der Umgebung aufweisen. Aufgrund dieser Grundregeln in Kombination mit dem *Law of Demeter* [47] dürfen an eine Methode (/ Funktion) nur Parameter übergeben werden, die von ihr auch verarbeitet werden. Dieses Vorgehen ist projektweit zu kommunizieren und einzufordern, um Kaskaden von Methodenaufrufen mit Pseudo-Parametern zu vermeiden. Werden diese Regeln nicht eingehalten, entstehen Schnittstellen, die von Implementierungen abhängig sind und deren Aussagekraft verwässert ist. Die Wartung des Sourcecodes wird dadurch erschwert.

4.2.2. Steuerung des Entwicklungsprozesses

Die Entwicklung umfangreicher Software, die bei *SnoWMAn* über Jahre erfolgte, ist ein langwieriger, fließender Prozess. Will man den Output in Form des Sourcecodes optimieren, ist es notwendig, permanent auf den Entwicklungsprozess kontrollierend einzuwirken, die Selbstkontrolle der

Entwickler reicht für ein optimales Ergebnis nicht aus. Wird die Position der Projektleitung mit ihrer Kontrolle auf den Entwicklungsvorgang nicht oder zu schwach wahrgenommen, entsteht ein gewisser “Wildwuchs” in den entstehenden Softwaremodulen der unterschiedlichen Entwickler. Jeder Programmierer folgt seinen persönlichen Motivationen und nicht den projekteinheitlichen Richtlinien, dies vielleicht sogar ohne sich dessen bewusst zu sein. Fehlt also für die Entwickler die kontrollierende Rückmeldung im Arbeitsalltag, kommt es zu einem sehr heterogenen Softwarekonglomerat und unterschiedliche Codequalitäten halten Einzug in das Endprodukt. Dies erschwert wiederum die Wartung der Software, und erhöht ihre Kosten. Um diese Effekte so gering wie möglich ausfallen zu lassen, sollten bei der Kontrolle der Entwicklung folgende Punkte beachtet werden

- ins Projekt einfließender Code muss systematisch, beispielsweise durch Stichproben, nach dem Vieraugenprinzip überprüft und der Entwickler auf Schwächen hingewiesen werden. Dabei ist
- die Codequalität zu beurteilen,
- die Codekompatibilität im Sinne des Projektes zu bewerten,
- eine existierende Dokumentation einzufordern,
- auf die Qualität der Dokumentation zu achten, und
- bei Falschinterpretationen etablierter Entwurfs-Muster korrigierend einzugreifen.

Werden diese Kontrollmechanismen am Beginn eines Projektes etabliert und konsequent verfolgt, sollten mit fortlaufender Dauer der Entwicklung die notwendigen Nachjustierungen abnehmen, auch wenn die Kontrolle nie völlig entfallen darf.

4.3. Denoising

Wie Quince in [21] ausführt, verursacht das Denoising der Sequenzierungsdaten mit den aktuellen Algorithmen einen wesentlichen Berechnungsaufwand zugunsten einer exakteren Bestimmung von OTUs bei der anschließenden Klassifikation. Da es für diese Problematik noch nicht sehr lange Lösungen in Form von nutzbaren Computerprogrammen gibt, und die Verbesserung in den Klassifikationsergebnissen vielversprechend sind, ist

es naheliegend diesen Verarbeitungsschritt in eine Plattform wie *SnoWMAn* aufzunehmen.
Wie hoch der Ressourcenbedarf mit der vorliegenden Systemstruktur von *SnoWMAn* tatsächlich ist konnte ohne konkret durchgeführte Tests schwer abgeschätzt werden. Es zeigte sich, dass die Rechenleistung des Rechenclusters nicht ausreichte, um das Denoising in einer für den Benutzer akzeptablen Verarbeitungszeit zu berechnen. Obwohl die technische Umsetzung kein Problem dargestellt hätte, wäre die benötigte zusätzliche Berechnungszeit des Denoising-Schritts mit dem derzeitigen System unbefriedigend lange ausgefallen. Im Sinne der qualitativen Verbesserung der *SnoWMAn*-Analyseergebnisse und angesichts der Fortschritte in der Hardwareentwicklung wird diese Erweiterung vermutlich mit der nächsten Ausbaustufe des Rechenclusters umgesetzt werden.

Seit Kurzem (April 2012) steht ein weiteres Tool zur Korrektur von Homopolymer-Fehlern namens *Acacia* [48] zur Verfügung. Es findet ähnlich viele Homopolymer-Fehler wie *AmpliconNoise* und *Denoiser* ist aber gegenüber Substitutions-Fehlern unempfindlicher als *AmpliconNoise*. Der Vorteil von *Acacia* liegt in der deutlich kürzeren Berechnungszeit gegenüber den beiden anderen Tools, das Programm benötigt weniger Computer-Ressourcen zur Datenverarbeitung. *Acacia* wurde in *Java* geschrieben und benötigt nur ein *Java Runtime Environment* zur Ausführung. Es kann als Commandline-Programm oder mit grafischer Benutzeroberfläche genutzt werden, sein Daten-Ausgabeformat ist *QIIME* [49] konform.

Die Laufzeiten der Tabelle 3.3 zeigen deutlich den Performance-Unterschied zwischen dem eingesetzten Entwicklungs-PC und dem Rechencluster der Live-Instanz. Das Laufzeit-Verhältnis über die gesamte Analyse mit der *RDP*-Pipeline mit Chimerafiltering beträgt 7,89 zu 1. Da Denoising nicht in die *SnoWMAn*-Plattform integriert wurde, geht die Berechnungszeit von zirka 12 Tagen für das Denoising nicht in diesen Vergleich ein. Denoising wurde als eigener Schritt im Vorfeld manuell auf der Commandline durchgeführt.
Nimmt man die Summe aus Analysezeit der Live-Instanz, 01:11:16, und die 12 Tage für das Denoising und setzt sie in Relation zur Analysezeit der Live-Instanz ergibt sich ein Faktor von >= 240. Bei diesem Anwendungsbeispiel wird offensichtlich, wie der Berechnungsaufwand des Denoisings den Zeitrahmen für eine gesamte (*RDP*-)Analyse “sprengt”.

4.4. Flexibilisierung der Pipelinestruktur

Bei der ursprünglichen Entwicklung der *SnoWMAn-Pipelines* war immer angedacht, die innere Softwarestruktur zur Einbindung der *Command*-Objekte hinter den Pipelines flexibel zu gestalten, um notwendige Erweiterungen von Pipelines mit neuen Commands zu ermöglichen. Dieser Aspekt wurde bei der Implementierung etwas aus den Augen verloren, weshalb die vor Beginn dieser Masterarbeit vorliegende Codestruktur zu starr ausgefallen ist, die gewünschten Erweiterungen unmöglich umgesetzt werden konnten. Durch Umbau der Strukturen und Einführung des neuen Mappingmechanismus zur Adressierung verwendeter Command-Objekte wurde es ermöglicht, Commands aufgrund der Benutzerentscheidung zur Laufzeit zu aktivieren. Diese Erweiterung legt die Basis für den Einbau der gewünschten Chimerafiltering-Option in alle Pipelines. Sie schafft den Unterschied bei der Festlegung der Abfolge von angewandten Command-Objekten in einer Pipeline zum Kompilierungszeitpunkt versus zur Laufzeit.
In der Implementierung, wie sie nach Beendigung dieser Masterarbeit vorliegt, werden die potentiell nutzbaren Command-Objekte einer Pipeline zwar zum Kompilierungszeitpunkt festgelegt, die Auswahl der angewandten Kommandos erfolgt aber erst zur Laufzeit aufgrund der Benutzerentscheidungen. Dieser Umbau erhöht deutlich die Adaptionsmöglichkeiten der vorliegenden Pipelines an zukünftige Anforderungen, die sich auf die Abfolge der auszuführenden Kommandos auswirken.

4.5. Chimerafiltering

Das *Chimerafiltering* ist ebenso wie das *Denoising* ein sehr aktuelles Thema in der Mikrobiom-Datenverarbeitung zur Reduzierung der sogenannten OTU-*inflation*. Enthalten die zu klassifizierenden Sequenzdaten Fehler in Form von Chimären oder Sequenzierungsfehlern, resultiert daraus immer eine zu hohe Anzahl OTUs. OTUs sind ihrerseits Ursprung resultierender biologischer Bewertungen, wie zum Beispiel der Biodiversität, und somit so exakt wie möglich zu ermitteln. Aufgrund der stetig steigenden Rechenleistung aktueller Computersysteme, gehören diese beiden Verfahren

vermutlich bald zu den angewandten Standardverfahren bei der Mikrobiom-Datenverarbeitung, und der einhergehende erhöhte Berechnungsaufwand spielt nur mehr eine geringe Rolle.

Entsprechend einer der Leitideen der *SnoWMAn*-Plattform, den Benutzer von Nutzungsdetails der verwendeten Software-Tools zu entlasten, wurde die *Chimerafiltering*-Option zur einfachen Nutzung in die Web-Oberfläche integriert, für den Benutzer in Form einer auswählbaren Checkbox zur Verfügung gestellt. Um die Daten-Ein- und -Ausgabe an das *mothur*-Chimerafiltering-Tool im Hintergrund muss sich der Anwender keine Gedanken machen.

Obwohl das Chimerafiltering nur sequenzbasiert-aligned Daten verarbeiten kann, und dieser Vorverarbeitungsschritt in den wenigsten Pipelines bereits vorhanden war, wurde bei dieser Erweiterung darauf geachtet, vorhandene Pipeline-Funktionen nicht zu beeinträchtigen, und negative Auswirkungen auf andere Pipeline-Commands zu vermeiden. Um das zu gewährleisten, mussten teilweise die Datenflüsse innerhalb eine Pipeline adaptiert werden. Letztendlich wurde dieses Ziel aber uneingeschränkt umgesetzt.

4.5.1. Verbesserung der Analyseergebnisse durch Chimerafiltering

Das eingeführte Chimerafiltering führte minimal zu einer OTU-Reduktion von -3,10% in der *BLAT*-Pipeline bei eindeutig erkannten Sequenzen und maximal zu einer Reduktion von -15,18% in der *UCLUST*-Pipeline bei einem Ähnlichkeitsmaß von 95% (entspricht 5% Differenz) innerhalb eines OTU-Clusters.
Bei den durchgeführten Tests dauert die Berechnung des Chimerafiltering-Schritts in der längsten Variante *JGAST*-Pipeline 2:41h, 46min benötigt der zusätzlich notwendige Alignment-Schritt, insgesamt also 3:27h. Bezogen auf die Berechnungszeit der Pipeline ohne Chimerafiltering von 13:11h ergibt das eine Verlängerung um 23,7% für die langsamste Variante. Alle anderen Pipelines benötigen relativ betrachtet weniger Zeit.
Der Vorteil durch die Verbesserung bei der Anzahl der klassifizierten OTUs, die dadurch besser die biologische Realität wiedergibt, überwiegt den

Nachteil der zusätzlich notwendigen Berechnungszeit von <= 24% für das Chimerafiltering im langsamsten Fall, und wird aller Voraussicht nach durch den Benutzer akzeptiert werden. Somit erscheint diese Erweiterung gerechtfertigt.

4.6. Schwierigkeiten bei der Informationsbeschaffung

Zu *SnoWMAn* existieren keine Design-Dokumente, welche die geschaffene Architektur aus Entwicklersicht beschreiben, um bei Erweiterungen für neue Entwickler die notwendige Wissensbasis zur Verfügung zu stellen. Die Größe und Komplexität der Software, mit ihrer Vielzahl an eingesetzten *Java*-Technologien, macht es sehr schwierig, einen Überblick zu entwickeln. Auch wenn durch den Einsatz von eingebauten Debug-Meldungen die großen Design-Entscheidungen nachvollzogen werden können, bleiben viele Hintergründe für kleinere Festlegungen unbekannt. Die Struktur der Plattform als verteiltes entkoppeltes System mit zumindest drei Haupt-"Schauplätzen", dem Benutzer-Frontend - Browser, dem Applikationsserver und dem *JClusterService*-Server, erschwert zusätzlich das Nachvollziehen von Prozessflüssen. Sie ist auch der Grund, warum ein Debugging ohne explizites Hintergrundwissen zum Design grundsätzlich sehr zeitaufwendig ist und vor allem auch seine Grenzen hat. So ist es beispielsweise nicht direkt möglich, dynamische Konfigurationsdetails des *Apache Struts*-Frameworks, das über XML-Dateien gesteuert wird, nachzuvollziehen. Viele Vorgänge laufen hier ausschließlich in dynamischen Framework-Libraries ab. Die interessanten Designfestlegungen sind in umfangreichen XML-Konfigurationsdateien verborgen, die nicht unter Zuhilfenahme der Laufzeitumgebung debuggt werden können.
Die Nutzung von vorkompilierten Libraries deren Sourcecode und Dokumentation nicht zur Verfügung stehen, ist per se eine besondere Herausforderung beim Reverse Engineering.

Auch wenn ein vorliegendes Design mittels Reverse Engineering-Methoden extrahiert werden konnte, können teilweise die Gründe für unlogisch erscheinende Implementierungsdetails unbeantwortet bleiben. Versteht ein

Entwickler bei Erweiterungen diese Details nicht, oder glaubt, diese zu verstehen, obwohl er nur Teile davon begreift, wirkt sich das bei neu implementierten Modulen aus. Hier besteht die große Gefahr, die Konsistenz zwischen altem und neuen Code zu gefährden.

Stellt die Extrahierung einer unbekannten Architektur ohne eigener Dokumentation grundsätzlich eine große Herausforderung dar, und ist diese vielfach sehr zeitaufwändig, wurden bei der vorliegenden Arbeit alle relevanten Design-Details extrahiert, um die gewünschten Ziele erfolgreich umzusetzen.

4.7. Schlussfolgerungen für zukünftige Arbeiten an *SnoWMAn*

Hoher Projektanteil für Reverse Engineering unter gegebenen Bedingungen

Die Herausforderungen die bei diesem Master-Projekt aufgetreten sind, zeigen die Notwendigkeit auf, bei zukünftigen Arbeiten an der Plattform durch neue mit *SnoWMAn* nicht vertraute Entwickler, einen hohen Anteil des Gesamtprojekts für Reverse Engineering-Maßnahmen einzuplanen. Gelingen mit dem richtigen Hintergrundwissen Erweiterungsarbeiten in Stunden, kann die Erarbeitung dieses Wissens Wochen in Anspruch nehmen. Diese Arbeiten zum Wissenserwerb von Systeminterna erscheinen für den Benutzer des Systems uninteressant, sind aber für den ausführenden Entwickler unumgänglich.

Umfangreiches SW-Entwicklungswissen

Weiters erschwert im vorliegenden Fall die komplexe Anwendung der genutzten *Java*-Technologien die Arbeiten neuer Entwickler. Potentielle Entwickler benötigen nicht nur Wissen in ein oder zwei Programmiersprachen,

sondern müssen den vollständigen Software-Lebenszyklus mit all den damit angewandten Prozessschritten überblicken.

Erleichterung bei Änderung an eingeführten Erweiterungen

In den Software-Modulen die bei dieser Arbeit entstanden sind wurde darauf geachtet, eine umfangreiche Sourcecode-Dokumentation einzuführen. Zukünftige Änderungen welche die vorliegenden Erweiterungen betreffen, sollten für die ausführenden Entwickler durch diese Dokumentation erleichtert werden. Dies spart Entwicklungszeit in diesen Bereichen ein.

Gefordertes Projektmanagement

Die Vielzahl der beschriebenen Herausforderungen mit all ihren Details erschwert das Management eines Projektes an *SnoWMAn*. Ist der Entwickler vor allem mit technischen Aspekten des Systems in seinem Arbeitsalltag konfrontiert, muss das Projektmanagement zusätzlich noch die inhaltlichen, biologischen Ziele hinter den Arbeiten im Auge behalten. Dies macht diese Aufgabe zu keiner leichten.

5. Ausblick

Das Ziel hinter *SnoWMAn*, für den Benutzer die Analyse von Mikrobiom-Daten möglichst einfach zu gestalten, und die technische Umsetzung bieten viel Raum für zukünftige Erweiterungen des Systems. Einige dieser Möglichkeiten sind zum Teil bereits angedacht und sollen hier kurz erläutert werden.

5.1. Nicht umgesetzte Erweiterungen

Für das ursprünglich geplante Denoising steht momentan (noch) zu wenig Rechenleistung am Rechencluster zur Verfügung. Betrachtet man das schnelle Fortschreiten der technischen Entwicklungen im Hardwarebereich, ist davon auszugehen, Denoising in naher Zukunft in *SnoWMAn* als Erweiterung nutzen zu können. Die interne Softwarestruktur von *SnoWMAn* bietet bereits alle notwendigen Schnittstellen für dieses neue Service.
Abhilfe kann auch durch neue weniger rechenintensive Tools wie *Acacia* [48] geschaffen werden, die mit der vorhandenen Rechenleistung auskommen. Wahrscheinlich ist dieser Lösungsansatz schneller umsetzbar.

5.2. Angedachte Erweiterungen

Derzeit sind nach einer erfolgreichen Analyse die Ergebnisse in Form von Tabellen, Grafiken und Dateien für den Benutzer abrufbar. Dies ist notwendig und sinnvoll. Praktisch wäre die Erweiterung um einen zusammenfassenden Bericht in Form eines *PDF*-Berichts, der die wichtigsten Informationen

enthält. Vorarbeiten für diese Erweiterung wurden im Seminarprojekt des Autors "Extensions of *SnoWMAn*" geleistet.

Neben den bestehenden fünf *Pipeline*s (*JGAST*, *UCLUST*, *Mothur*, *RDP*, *BLAT*) könnte eine neue *QIIME*-Pipeline (Quantitative Insights Into Microbial Ecology) [49] eingeführt werden. Dies würde den Nutzen für den Benutzer weiter erhöhen und vermutlich mehr Nutzer der Plattform anziehen.

5.3. Refactoring

Will man aus Projektmanagementsicht zukünftige Erweiterungen an *SnoWMAn* erleichtern, wird ein Refactoring des gesamten Projekts mit folgenden Zielsetzungen empfohlen.

- Vereinheitlichung der angewandten Kodierrichtlinien
- Vereinheitlichung der internen Struktur aller Pipelines
- Erstellung von Software-Architektur-Dokumentation
- Einführung umfassender Sourcecode-Dokumentation
- Berichtigung / Vereinfachung von Schnittstellen
- Erstellung von Best Practice Anweisungen zur gezielten Erweiterung der Plattform um neue *Command*s
- Erstellung einer aktuellen Wissensdatenbank im Intranet für Entwickler zur Archivierung und Weitergabe des individuellen Entwicklerwissens

In diese Themen kann viel Arbeit investiert werden, ohne einen sichtbaren Effekt aus Benutzersicht erzielen zu können. Insofern werden Projekte mit diesen Zielsetzungen nicht einfach umsetzbar sein.

Für den Autor stellte diese Arbeit ein sehr interessantes und umfangreiches Projekt dar, in dem er trotz einiger Hürden den Überblick behalten konnte. Die Vielfalt in den verfolgten biologischen Zielen der Analysen und der technischen Aspekte der Umsetzung ist ein interessantes Spannungsfeld zwischen kompliziertem Domain-Wissen und vielschichtigem technischen Know How. Viele weitere Arbeiten in unterschiedlichsten Bereichen sind zu erwarten.

Literatur

1. Hooper L, Gordon J: **Commensal Host-Bacterial Relationships in the Gut.** *Science* 2001, **292:** 1115–1118.
2. Kyrpides NC: **Fifteen years of microbial genomics: meeting the challenges and fulfilling the dream.** *Nat Biotechnol* 2009, **27:** 627–632.
3. Hamady M, Knight R: **Microbial community profiling for human microbiome projects: Tools, techniques, and challenges.** *Genome Research* 2009, **19**(7)**:** 1141–1152.
4. Morgan XC, Huttenhower C: **Chapter 12: Human Microbiome Analysis.** *PLoS Computational Biology* 2012, **8**(**12**)**.**
5. Littman DR, Pamer EG: **Role of the commensal microbiota in normal and pathogenic host immune responses.** *Cell Host & Microbe* 2011, **10**(4)**:** 311–323.
6. MetaHIT Metagenomics of the Human Intestinal Tract. `http://www.metahit.eu`. May, 2013.
7. National Institutes of Health Office of Strategic Coordination - The Common Fund. `http://nihroadmap.nih.gov/`. May, 2013.
8. Group NHW, Peterson J, *et al.:* **The NIH Human Microbiome Project.** *Genome Research* 2009: 2317–2323.
9. Turnbaugh PJ, Ley RE, *et al.:* **Feature The human microbiome project.** *Nature* 2007, **449:** 804–810.
10. University of California. NIH Human Microbiome Project Catalog. `http://www.hmpdacc-resources.org/hmp_catalog/main.cgi?section=Home&page=Home`. May, 2013.
11. Qin J, Li R, *et al.:* **A human gut microbial gene catalogue established by metagenomic sequencing.** *Nature* 2010, **464:** 59–65.

12. Arumugam M, Raes J, *et al.:* **Enterotypes of the human gut microbiome.** *Nature* 2011, **473:** 174–180.

13. Gevers D, Pop M, Schloss PD, Huttenhower C: **Bioinformatics for the Human Microbiome Project.** *PLoS Computational Biology* 2012, **8(11).**

14. SnoWMAn 1.11. `https://epona.genome.tugraz.at/snowman/`. May, 2013.

15. Institute for Genomics and Bioinformatics. `https://genome.tugraz.at/`. May, 2013.

16. Technische Universität Graz. `http://portal.tugraz.at`. May, 2013.

17. The Java EE 5 Tutorial. `http://docs.oracle.com/javaee/5/tutorial/doc/docinfo.html`. May, 2013.

18. Margulies M, Egholm M, *et al.:* **Genome sequencing in microfabricated high-density picolitre reactors.** *Nature* 2005, **437:** 376–380.

19. Bartlett JMS, Stirling D: **A Short History of the Polymerase Chain Reaction**. In *PCR Protocols.* 2003: 3–6.

20. Von Mering C, Hugenholtz P, *et al.:* **Quantitative Phylogenetic Assessment of Microbial Communities in Diverse Environments.** *Science* 2007, **315:** 1126–1130.

21. Quince C, Lanzen A, Davenport RJ, Turnbaugh PJ: **Removing Noise From Pyrosequenced Amplicons.** *BMC Bioinformatics* 2011, **12.**

22. Schloss PD, Westcott SL, *et al.:* **Introducing mothur: Open-Source, Platform-Independent, Community-Supported Software for Describing and Comparing Microbial Communities.** *Appl Environ Microbiol* 2009, **75:** 7537–7541.

23. Apache Struts. `http://struts.apache.org/`. May, 2013.

24. JavaServer Pages Technology. `http://www.oracle.com/technetwork/java/javaee/jsp/index.html`. May, 2013.

25. Enterprise JavaBeans Technology. `http://www.oracle.com/technetwork/java/javaee/ejb/index.html`. May, 2013.

26. JavaScript/DOM. `http://de.selfhtml.org/javascript/`. May, 2013.

27. Cascading Style Sheets. `http://www.w3.org/Style/CSS/`. May, 2013.

28. Stocker G, Fischer M, *et al.:* **iLAP: a workflow-driven software for experimental protocoll development, data acquisition and analysis.** *BMC Bioinformatics* 2009, **10:** 390.

29. Sun Grid Engine. http://gridengine.sunsource.net. May, 2013.

30. J2SE 5.0. http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javase5-419410.html. May, 2013.

31. JBoss. http://www.jboss.org. May, 2013.

32. MySql. http://dev.mysql.com/. May, 2013.

33. MyEclipse. http://www.myeclipseide.com/. May, 2013.

34. Apache Ant. http://ant.apache.org. May, 2013.

35. XMind - Professional & Powerful Mind Mapping Software. http://www.xmind.net/. May, 2013.

36. Dia. http://projects.gnome.org/dia/. May, 2013.

37. Deacon J: **Cohesion**. In *Object-Oriented Analysis and Design.* 2004: 253.

38. Deacon J: **Minimizing the class coupling**. In *Object-Oriented Analysis and Design.* 2004: 389–390.

39. Javadoc Tool. http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html. May, 2013.

40. Gorkiewicz G, Thallinger GG, *et al.:* **Alterations in the Colonic Microbiota in Response to Osmotic Diarrhea.** *PLoS ONE* 2013: e55817.

41. Apache. The Apache Commons CLI. http://commons.apache.org/proper/commons-cli/. May, 2013.

42. Schloss PD. Chimera.uchime. http://www.mothur.org/wiki/Chimera.uchime. May, 2013.

43. Schloss PD. mothur. http://www.mothur.org/. May, 2013.

44. Message Passing Interface. http://de.wikipedia.org/wiki/Message_Passing_Interface. May, 2013.

45. Flat file database. http://en.wikipedia.org/wiki/Flat_file_database. May, 2013.

46. Schloss PD. mothur. http://www.mothur.org/wiki/Shhh.seqs. May, 2013.

47. Lieberherr KJ, Holland IM: **Assuring good style for object-oriented programs.** *IEEE, Software* 1989, **6(5):** 38–48.

48. Bragg L, Stone G, Imelfort M, Hugenholtz P, Tyson GW: **Fast, accurate error-correction of amplicon pyrosequences using Acacia.** *Nat Methods* 2012, **9:** 425–426.

49. Caporaso JG, Kuczynski J, *et al.:* **QIIME allows analysis of high-throughput community sequencing data.** *Nat Methods* 2010, **7:** 335–336.

50. Danial A. cloc (Count Lines Of Code). http://sourceforge.net/projects/cloc/. May, 2013.

Appendix

Anhang A

Tabellen

A.1 *SnoWMAn*-Sourcecode Statistik

Die Statistik über den Sourcecode von *SnoWMAn* wurde mittels *cloc* [50] ermittelt, siehe Tabelle A.1.

A.2 Pipeline-Analyse Statistiken

Die angeführten Statistiken zu den einzelnen Pipeline-Analysen wurden mit *Libre Office Calc* im Mai 2013 erstellt.

Anhang A Tabellen

Tabelle A.1: Sourcecodestatistik von *SnoWMAn*, zeigt die Anzahl von Dateien und Zeilen des gesamten Projekts geordnet nach den verwendeten Programmiersprachen

Language	**files**	**blank**	**comment**	**code**
HTML	1167	38501	20362	299574
Java	764	14889	16853	77207
XML	54	398	397	40938
JSP	86	616	391	7669
Bash	57	442	1325	3648
Javascript	11	413	560	2784
CSS	5	184	105	1034
Perl	5	55	96	383
Ant	8	64	140	277
SQL	1	21	32	130
ASP.Net	1	0	0	88
Python	2	12	5	62
Bash	2	0	52	50
DTD	1	2	0	38
DOS Batch	1	0	0	1
Summe über alle Sprachen außer DTD, Dos Batch und HTML	996	17094	19956	134270

Tabelle A.2: Auswirkungen Chimerafiltering - *JGAST*-Pipeline, Vergleich Analyse ohne (*V12_updatedJGAST_NoChimera - A*) und mit (*V12_updatedJGAST_WithChimera - B*) Chimerafiltering

	Sequences	Unique Seqs	Number of phylotypes
			RDP
Analysis results for A	438736	76111	240
Analysis results for B	434118	73500	236
B – A	-4618	-2611	-4
(B – A) / A	-1,05%	-3,43%	-1,67%

V12_updatedJGAST_NoChimera				
Preprocessing	506218	Sequences	3:21 PM	00:00:19
Merging	438738	Sequences	3:21 PM	00:00:11
Uniquing	438738	Sequences	3:21 PM	00:03:05
JGast	76113	Sequences	3:24 PM	13:05:35
Summary	76111	Sequences	4:30 AM	00:00:10
Rarefaction	76111	Sequences	4:30 AM	00:01:50
OTUDistribution	76111	Cluster	4:32 AM	00:00:00

V12_updatedJGAST_WithChimera				
Preprocessing	506218	Sequences	4:34 AM	00:00:19
Merging	438738	Sequences	4:34 AM	00:00:11
Uniquing	438738	Sequences	4:34 AM	00:01:14
Alignment	76113	Sequences	4:35 AM	00:45:50
Chimerafiltering	76113	Sequences	5:21 AM	02:41:06
JGast	73502	Sequences	8:03 AM	12:47:30
Summary	73500	Sequences	8:50 PM	00:00:08
Rarefaction	73500	Sequences	8:50 PM	00:01:59
OTUDistribution	73500	Cluster	8:52 PM	00:00:00

Tabelle A.3: Auswirkungen Chimerafiltering - *UCLUST*-Pipeline, Vergleich Analyse ohne (*V12_updatedUCLUST_NoChimera - A*) und mit (*V12_updatedUCLUST_WithChimera - B*) Chimerafiltering

	Sequences	Unique Seqs	Number of phylotypes						
			unique	0,01	0,02	0,03	0,04	0,05	0,06
Analysis results for A	438738	76113	76113	15232	7062	5018	3886	3057	2513
Analysis results for B	434120	73502	73502	13795	6100	4297	3297	2593	2174
B – A	-4618	-2611	-2611	-1437	-962	-721	-589	-464	-339
(B – A) / A	-1,05%	-3,43%	-3,43%	-9,43%	-13,62%	-14,37%	-15,16%	-15,18%	-13,49%

V12_updatedUCLUST_NoChimera				
Preprocessing	506218	Sequences	1:59 PM	00:00:20
Merging	438738	Sequences	2:00 PM	00:00:11
Uniquing	438738	Sequences	2:00 PM	00:01:00
UCLUST	76113	Sequences	2:01 PM	00:02:12
RefSeq	76113	Cluster	2:03 PM	00:01:05
Classification	112881	Sequences	2:04 PM	00:10:57
Rarefaction			2:16 PM	00:17:12
Summary	112881	Cluster	2:33 PM	00:01:29
OTU Distribution	112881	Cluster	2:35 PM	00:00:27

V12_updatedUCLUST_WithChimera				
Preprocessing	506218	Sequences	2:37 PM	00:00:22
Merging	438738	Sequences	2:37 PM	00:00:12
Uniquing	438738	Sequences	2:37 PM	00:01:21
Alignment	76113	Sequences	2:39 PM	00:54:24
Chimerafiltering	76113	Sequences	3:33 PM	02:40:44
UCLUST	73502	Sequences	6:14 PM	00:02:08
RefSeq	73502	Cluster	6:16 PM	00:01:25
Classification	105758	Sequences	6:18 PM	00:10:18
Rarefaction			6:28 PM	00:10:46
Summary	105758	Cluster	6:39 PM	00:01:20
OTU Distribution	105758	Cluster	6:40 PM	00:00:24

Tabelle A.4: Auswirkungen Chimerafiltering - *mothur*-Pipeline, Vergleich Analyse ohne (*V12_updatedMothurNoChimera - A*) und mit (*V12_updatedMothurWithChimera - B*) Chimerafiltering

	Sequences	Unique Seqs	Number of phylotypes							
			unique	0,00	0,01	0,02	0,03	0,04	0,05	0,06
Analysis results for A	438738	76113	76112	75838	39089	22329	14445	10620	8454	7056
Analysis results for B	434120	73502	73501	73230	37200	20819	13182	9513	7466	6189
B – A	-4618	-2611	-2611	-2608	-1889	-1510	-1263	-1107	-988	-867
(B – A) / A	-1,05%	-3,43%	-3,43%	-3,44%	-4,83%	-6,76%	-8,74%	-10,42%	-11,69%	-12,29%

V12_updatedMothurNoChimera				
Preprocessing	506218	Sequences	3:12 PM	00:00:22
Merging	438738	Sequences	3:13 PM	00:00:13
Uniquing	438738	Sequences	3:13 PM	00:01:05
Alignment	76113	Sequences	3:14 PM	00:42:12
Distance	76113/1093	Seqs/Length	3:56 PM	01:22:00
Clustering	88001102	Distances	5:18 PM	08:13:31
RefSeq	76112	Clusters	1:32 AM	00:43:03
Classification	253943	Sequences	2:15 AM	01:13:29
Rarefaction			3:28 AM	00:12:40
Summary	253935	Clusters	3:41 AM	00:02:35
OTU Distribution	253935	Clusters	3:44 AM	00:00:53

V12_updatedMothurWithChimera				
Preprocessing	506218	Sequences	3:46 AM	00:00:21
Merging	438738	Sequences	3:46 AM	00:00:13
Uniquing	438738	Sequences	3:47 AM	00:01:13
Alignment	76113	Sequences	3:48 AM	00:41:23
Chimerafiltering	76113	Sequences	4:29 AM	02:41:50
Distance	73502/1093	Seqs/Length	7:11 AM	01:27:00
Clustering	87242510	Distances	8:38 AM	07:58:17
RefSeq	73501	Clusters	4:36 PM	00:42:52
Classification	241100	Sequences	5:19 PM	01:08:38
Rarefaction			6:28 PM	00:15:03
Summary	241092	Clusters	6:43 PM	00:02:27

Tabelle A.5: Auswirkungen Chimerafiltering - *RDP*-Pipeline, Vergleich Analyse ohne (*V12_updatedRDPNoChimera - A*) und mit (*V12_updatedRDPWithChimeraAfterStatcorrection - B*) Chimerafiltering

	Sequences	Unique Seqs	Number of phylotypes						
			0,0	0,01	0,02	0,03	0,04	0,05	0,06
Analysis results for A	435669	69212	30947	22014	9077	5622	3988	3141	2480
Analysis results for B	431071	66648	29750	21124	8549	5238	3704	2895	2313
B – A	-4598	-2564	-1197	-890	-528	-384	-284	-246	-167
(B – A) / A	-1,06%	-3,70%	-3,87%	-4,04%	-5,82%	-6,83%	-7,12%	-7,83%	-6,73%

V12_updatedRDPNoChimera				
Sample splitting	506218	Sequences	3:21 PM	00:01:04
Trimming and Filtering	485287	Sequences	3:22 PM	00:03:36
Orientation check	435669	Sequences	3:26 PM	00:02:20
Merging	435669	Sequences	3:28 PM	00:02:00
Uniquing	435669	Sequences	3:30 PM	00:00:48
Aligmnent-Infern	69212	Sequences	3:31 PM	00:19:39
Clustering	69212	Sequences	3:55 PM	02:36:22
Dereplication	77269	Clusters	6:31 PM	00:05:01
Classification	77269	Sequences	6:36 PM	00:06:19
Rarefaction			6:42 PM	00:11:05
Summary	77269	Clusters	6:54 PM	00:00:28
OTUDistribution			6:54 PM	00:00:20

V12_updatedRDPWithChimeraAfterStatcorrection				
Sample splitting	506218	Sequences	10:40 AM	00:03:19
Trimming and Filtering	485287	Sequences	10:44 AM	00:03:34
Orientation check	435669	Sequences	10:47 AM	00:02:39
Merging	435669	Sequences	10:50 AM	00:02:23
Uniquing	435669	Sequences	10:52 AM	00:00:45
Aligmnent-Moth	69212	Sequences	10:53 AM	01:34:15
Chimerafiltering	69212	Sequences	12:27 PM	00:53:44
Aligmnent-Infern	66648	Sequences	1:21 PM	00:18:22
Clustering	66648	Sequences	1:43 PM	06:00:17
Dereplication	73573	Clusters	7:43 PM	00:04:37
Classification	73573	Sequences	7:48 PM	00:06:56
Rarefaction			7:55 PM	00:10:37
Summary	73573	Clusters	8:05 PM	00:00:26
OTUDistribution			8:06 PM	00:00:19

Tabelle A.6: Auswirkungen Chimerafiltering - *BLAT*-Pipeline, Vergleich Analyse ohne (*V12_updatedBLATNoChimera - A*) und mit (*V12_updatedBLATWithChimera - B*) Chimerafiltering

	Sequences	Unique Seqs	Number of phylotypes						
			G2_chip	Hugenholtz	Ludwig	NCBI	NCBI_ID	Pace	RDP
Analysis results for A	330077	98917	55	144	87	101	220	40	119
Analysis results for B	95851	95851	53	140	83	101	218	39	116
B – A	-234226	-3066	-2	-4	-4	0	-2	-1	-3
(B – A) / A	-70,96%	-3,10%	-3,64%	-2,78%	-4,60%	0,00%	-0,91%	-2,50%	-2,52%

V12_updatedBLATNoChimera				
Preprocessing	506218	Sequences	12:26 AM	00:00:20
Uniquing	438738	Sequences	12:26 AM	00:00:10
BLAT	99479	Sequences	12:26 AM	01:36:48
Classification	99479	Sequences	2:03 AM	00:01:55
Statistics	98917	Sequences	2:05 AM	00:00:04
Summary	98917	Sequences	2:05 AM	00:00:13
Rarefaction	98917	Sequences	2:06 AM	00:19:05
OTUDistribution	98917	Cluster	2:25 AM	00:00:00

V12_updatedBLATWithChimera				
Preprocessing	506218	Sequences	2:27 AM	00:00:19
Uniquing	438738	Sequences	2:27 AM	00:00:09
Aligmnent-Moth	99479	Sequences	2:27 AM	00:55:56
Chimerafiltering	99479	Sequences	3:23 AM	00:16:39
BLAT	96411	Sequences	3:40 AM	01:35:48
Classification	96411	Sequences	5:16 AM	00:01:41
Statistics	95851	Sequences	5:17 AM	00:00:03
Summary	95851	Sequences	5:17 AM	00:00:08
Rarefaction	95851	Sequences	5:18 AM	00:08:41
OTUDistribution	95851	Cluster	5:26 AM	00:00:00

Tabelle A.7: Auswirkungen Denoising - *RDP*-Pipeline mit Chimerafiltering, Vergleich Analysen ohne (*V12_updatedRDPWithChimeraAfterStatcorrection - A*) und mit (*V12_shhh_FastaExtended_ChimeraCheckedRDP - B*) Denoising

	Sequences	Unique Seqs	Number of phylotypes						
			0,0	0,01	0,02	0,03	0,04	0,05	0,06
Analysis results for A	431071	66648	29750	21124	8549	5238	3704	2895	2313
Analysis results for B	452363	21943	14131	10501	4766	3059	2251	1853	1534
B – A	21292	-44705	-15619	-10623	-3783	-2179	-1453	-1042	-779
(B – A) / A	+4,94%	-67,08%	-52,50%	-50,29%	-44,25%	-41,60%	-39,23%	-35,99%	-33,68%

Anhang B

Diagramme

Die angeführten Strukturdiagramme sind im Rahmen des Reverse Engineerings von *SnoWMAn* zu den verschiedenen Pipelines entstanden und dienten als Gedächtnisstütze. Sie unterliegen keiner formalen Struktur und wurden mit Dia [36] erstellt. Sie stellen ausgehend von den *Pipeline*-Klassen, mit ihren *Commands*, aus dem Sourcecode herauskristallisierte Zusammenhänge der Datenflüssen zwischen den einzelnen Kommandos dar. Die einzelnen *Commands* werden mit Rechtecken, die von ihnen erzeugten bzw. verarbeiteten Dateien mit Ellipsen dargestellt. Durchgezogene Linien stellen starre Verbindungen, gestrichelte Linien optionale Verbindungen dar. Optionale Verbindungen entstehen meistens aufgrund von Benutzerentscheidungsmöglichkeiten die Funktion der *Pipeline* betreffend.

Das Diagramm B.3 stellt die Zusammenhänge der für diese Arbeit relevanten Pipeline-Software-Module dar.

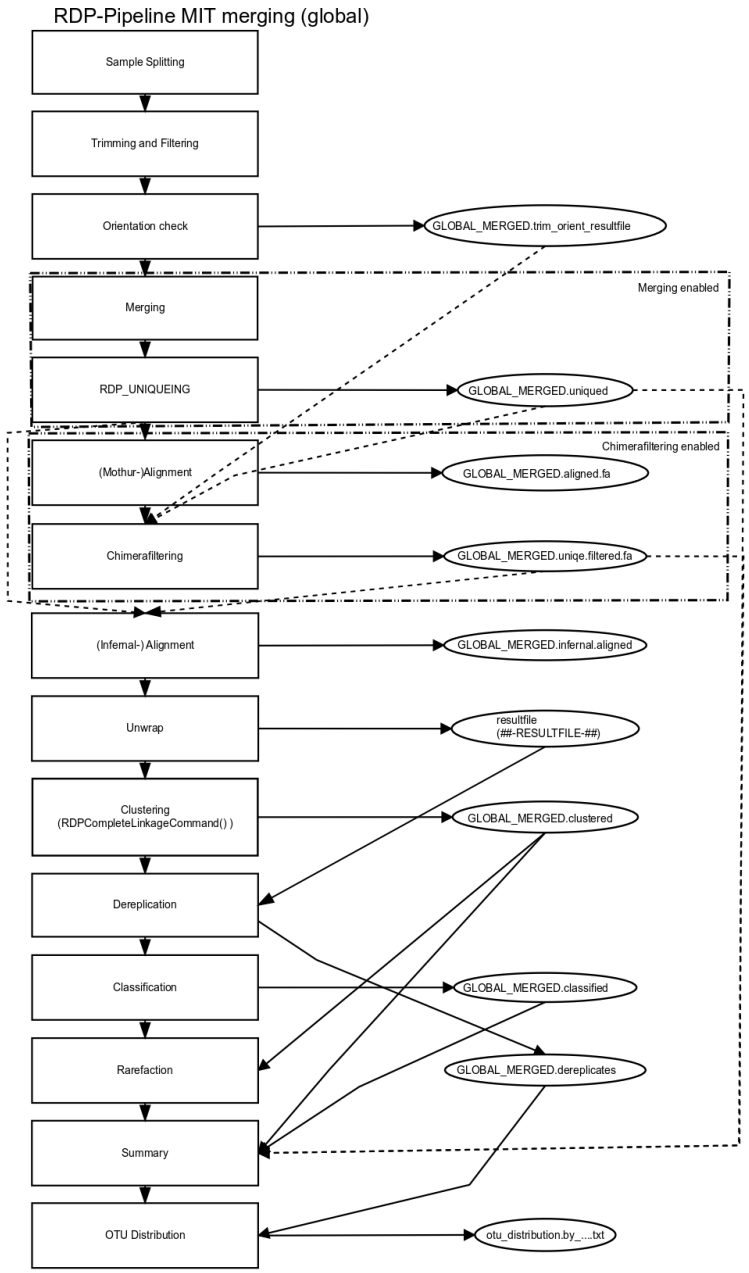

Abbildung B.1: Strukturdiagramm *RDP-Pipeline*, beinhaltet alle *Command*s der Pipeline (Rechtecke) und zeigt Dateien (Elipsen) die unter den Kommandos ausgetauscht werden, durchgezogene Linie zeigen einen fixen Datenaustausch, gestrichelte Linien stellen optionale Datenflüsse dar, diese sind von Benutzerentscheidungen abhängig, die gestrichelt eingerahmten Kommandos mit ihren Dateien sind optional aufgrund der Benutzerfestlegung

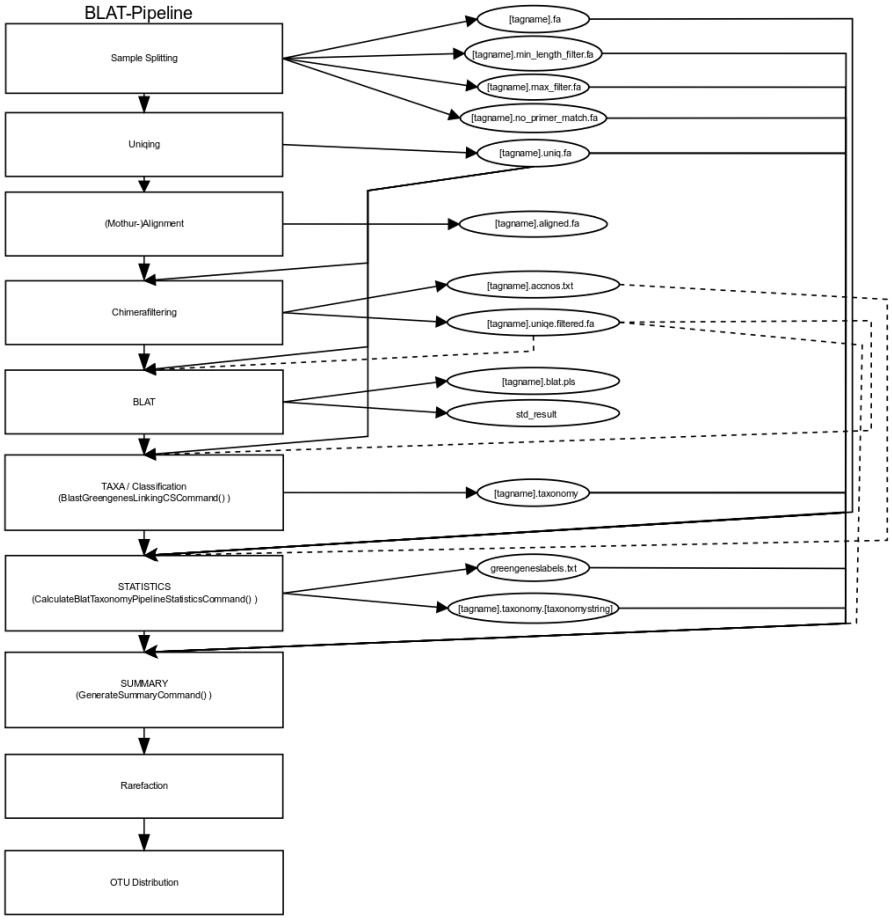

Abbildung B.2: Strukturdiagramm BLAT-*Pipeline*, beinhaltet alle *Command*s der Pipeline (Rechtecke) und zeigt Dateien (Elipsen) die unter den Kommandos ausgetauscht werden, durchgezogene Linie zeigen einen fixen Datenaustausch, gestrichelte Linien stellen optionale Datenflüsse dar, diese sind von Benutzerentscheidungen abhängig

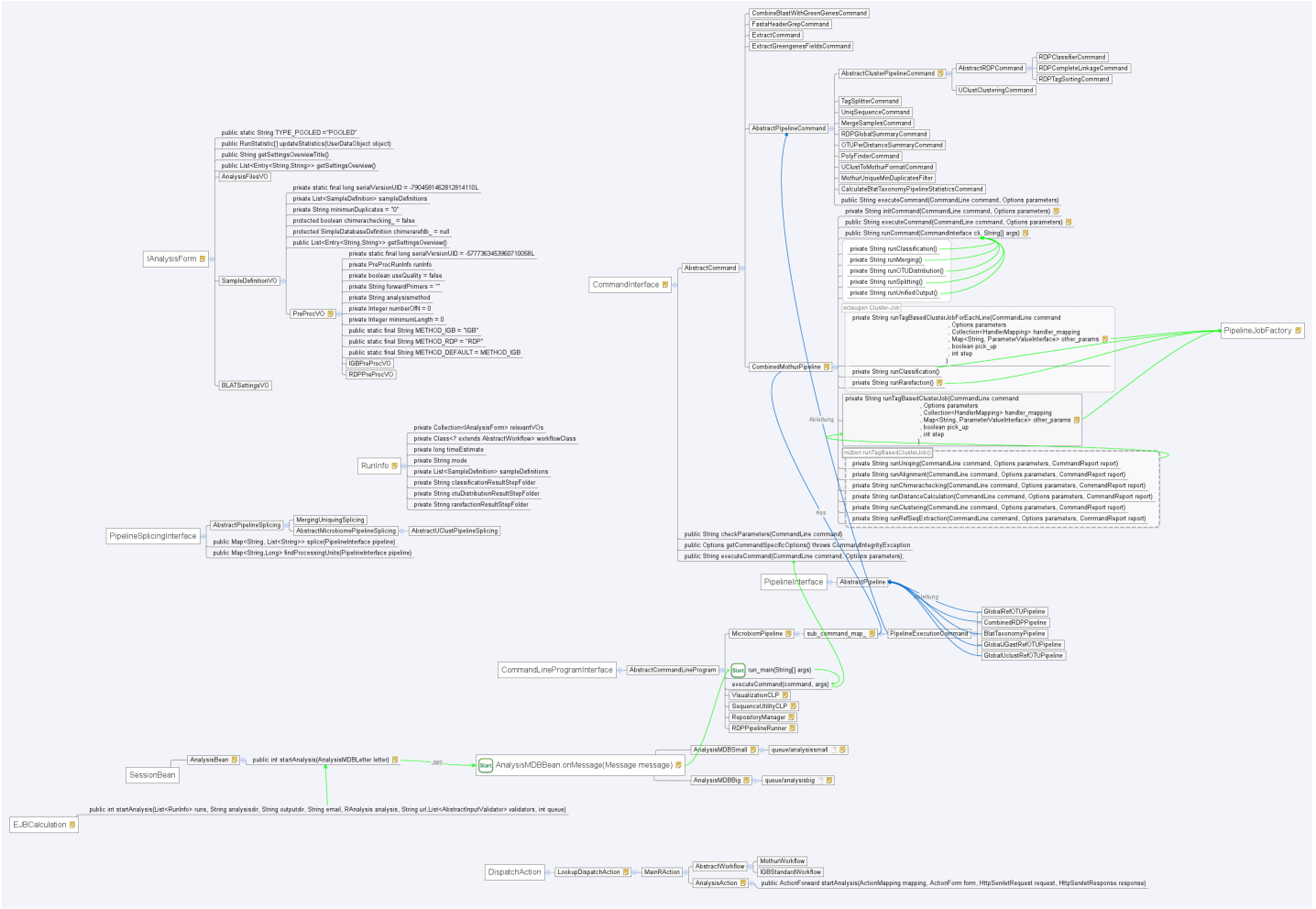

Abbildung B.3: Softwarestruktur der *SnoWMAn*-Pipelines, zeigt Zusammenhänge der wichtigsten Klassen und Methoden, Rechtecke –> Klassen, blaue Linie mit Pfeil –> Ableitung, blaue Linie ohne Pfeil –> Assoziation, grüne Linie mit Pfeil –> Methodenaufruf