

Ein schlankes Workflow-Tool für Bioinformatik

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Alexander Albl

Matrikelnummer 9825612

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung

Betreuer: Univ.Prof. Mag.rer.soc.oec. Dr.rer.soc.oec. Schahram Dustdar

Mitwirkung: Univ.Ass. Mag.rer.soc.oec. Dr.rer.soc.oec. Ivona Brandic

Wien, 21.09.2010

(Unterschrift Verfasser)

(Unterschrift Betreuer)

Alexander Albl
Westbahnstraße 31/2/33
1070 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, den 21. September 2010

Alexander Albl

Abstract

Workflows in Bioinformatics face problems not very common with other well known operational procedures. Huge amounts of sequenced genomic data in different formats are explored and, moreover, have to be stored for a long time to comply to scientific standards.

In detail, researchers of the Vienna Chair of Bioinformatics, located at University of Natural Resources and Life Sciences examine genes or genomes of mice for contaminations with different bacteria, eg. *Escherichia coli*. This requires taking random samples with different or even newly introduced parameters and comparing them to reference data sets. Those workflows rarely change fundamentally, but in detail. Currently, those questions are tackled with high staff assignments as no alternative is available to provide standardized workflow descriptions and parameter support.

Existing systems, be it from business process management or applications that specifically cater for scientific workflows, consequently take use of graphical user interfaces, while a more lightweight approach for a coordinated way of executing and storing workflows is not yet known. Moreover, it has to be ensured that existing scripts and other applications can still be used as before.

This master thesis addresses these problems by introducing a framework supporting researchers of the Chair of Bioinformatics, Department of Biotechnology at the University of Natural Resources and Life Sciences, Vienna in describing their workflows in a standardized way, providing a means of executing those workflows and helping to keep a history of both results and workflows. Describing workflows is achieved by a self-developed XML-syntax which can be edited with any texteditor, while data-retention is gained by accessing a subversion-repository (allowing for standardized tracking of changes).

Evaluation is done by testing the system with a real-life workflow accounting for a better part of most common problems faced by researchers. This workflow consists of several tasks, beginning with downloading large data sets from different databases if not locally accessible. Further on, those data are transformed using an external script to generate indices and then splitted to multiple chunks allowing for parallel execution. As a last step, those chunks are tested for contamination with *E. coli* bacteria. The results are finally provided with an outlook on forthcoming versions.

Kurzfassung

Arbeitsabläufe im Umfeld der Bioinformatik-Forschung unterliegen diversen Problemen, die in ihrer speziellen Kombination in keinem anderen Umfeld auftreten. Große Datenmengen an sequenzierten Genomen in unterschiedlichen Formaten müssen oftmals bearbeitet werden, und darüber hinaus im Sinne der wissenschaftlichen Nachvollziehbarkeit über einen langen Zeitraum gespeichert werden.

Spezielle Fragestellungen der Forscher des Chairs of Bioinformatics der Universität für Bodenkultur Wien erfordern die stichprobenartige Analyse von Genen oder Genomen von Mäusen zur Identifizierung von Kontaminationen mit verschiedenen Bakterien, etwa *Escherichia coli*. Dabei werden unter oftmaliger Veränderung bzw. Neueinführung von Parametern die Daten in grundsätzlich gleichen Arbeitsabläufen untersucht und mit Referenzdatensätzen verglichen. Aktuell erfolgt diese Arbeit mit hohem Personalaufwand, da es keine Möglichkeit gibt, die Tätigkeit durch genormte Arbeitsabläufe mit Unterstützung von Parametern und Resultatsvergleichen zu vereinfachen.

Bestehende Systeme aus dem Geschäftsprozessmanagement genauso wie bestehende Applikationen für die Bearbeitung von wissenschaftlichen Workflows bauen konsequent auf die Nutzung einer graphischen Benutzeroberfläche auf, ein leichtgewichtigerer Ansatz zur koordinierten Ausführung und Speicherung der Arbeitsabläufe ist zum aktuellen Zeitpunkt nicht bekannt. Darüber hinaus besteht die Notwendigkeit, bestehende Skripte und andere Applikationen weiterhin zu verwenden.

In dieser Arbeit wird auf diese Problematik eingegangen, indem ein neu entwickeltes Framework vorgestellt wird, dass die Forscher des Vienna Science Chair of Bioinformatics der Universität für Bodenkultur Wien dabei unterstützt, ihre vorhandenen sowie zukünftige Arbeitsabläufe in einer standardisierten Form zu beschreiben, diese in einer vorgegebenen Umgebung ablaufen zu lassen, und die Ergebnisse zu historisieren. Als Beschreibungssprache kommt dabei eine selbst entwickelte XML-Syntax zum Einsatz, die mithilfe eines beliebigen Editors auf einfache Art und Weise erstellt werden kann. Die Historisierung der Daten erfolgt über den Zugriff auf ein zentrales Subversion-Repository, sodass Änderungen an Arbeitsabläufen in einer standardisierten Form nachvollziehbar sind.

Zur Evaluierung der erstellten Applikation wird ein Workflow aus der wissenschaftlichen Praxis herangezogen, dessen Aufgaben beispielhaft für einen Großteil der erwarteten Problemstellungen sind. Dieser Arbeitsablauf sieht vor, mehrere Datensets

von unterschiedlichen Datenbanken herunterzuladen (sofern diese Daten lokal noch nicht verfügbar sind), diese weiters durch mehrere Transformationsschritte aufzubereiten und anschließend auf Kontaminierungen mit *E. coli*-Bakterien zu untersuchen. Die Transformationen beinhalten die Bearbeitung mit einem extern verfügbaren Skript zur Erstellung von Indizes, darüber hinaus werden die Daten zur parallelen Bearbeitung in mehrere Teile aufgesplittet. Die Ergebnisse werden abschließend mit einem Ausblick auf zukünftige Versionen bewertet.

Inhaltsverzeichnis

Abstract	i
Kurzfassung	i
Abbildungsverzeichnis	vii
Tabellenverzeichnis	ix
Algorithmenverzeichnis	xi
1 Einführung	1
1.1 Problembeschreibung	2
1.2 Aufgabenstellung und Anwendungsszenario	3
1.3 Ergebnisse	5
1.4 Aufbau der Arbeit	5
2 Grundlagen und Stand der Technik	8
2.1 Begriffsdefinitionen	8
2.1.1 Geschäftsprozessmanagement	9
2.1.2 Geschäftsprozess	9
2.1.3 Workflow	10
2.1.4 Workflowmanagementsystem	11
2.1.5 Versionsverwaltung	11
2.2 Workflowsysteme aus dem Umfeld des Geschäftsprozessmanagements .	13
2.2.1 YAWL - Yet Another Workflow Language	13
2.2.2 JBoss jBPM	15
2.3 Workflowsysteme aus dem Umfeld der Bioinformatik	17
2.3.1 Taverna	18
2.3.2 Triana	20

2.4	Zusammenfassung	21
3	Architektur	23
3.1	Systemumfeld	23
3.2	Anforderungen	23
3.2.1	Schlankheit und Einsetzbarkeit	24
3.2.2	Wiederverwendung und Orchestrierung von bestehenden Skripten	24
3.2.3	Einbindung externer Applikationen	24
3.2.4	Erweiterbarkeit der Applikation	25
3.2.5	Versionsmanagement	25
3.2.6	Gliederung von Arbeitsabläufen	25
3.2.7	Parametrisierung von Arbeitsaufgaben	26
3.3	Architektur	26
3.3.1	Anwendungsfälle	26
3.3.2	Verarbeitung von Workflowbeschreibungen	27
3.3.3	Ausführungskomponente	27
3.3.4	Subversion	27
3.4	Zusammenfassung	28
4	Implementierung	29
4.1	Implementierungsentscheidungen	29
4.1.1	Paradigmen	29
4.1.2	Entwicklungsumgebung	30
4.2	Eingesetzte Bibliotheken	31
4.2.1	Spring Framework	31
4.2.2	SVNkit	32
4.2.3	Apache Commons CLI	33
4.2.4	Apache Commons Exec	33
4.2.5	JDOM	34
4.2.6	log4j	34
4.2.7	HyperSQL	35
4.2.8	Apache Ant	35
4.3	Programmstruktur	37
4.3.1	Pakete	37
4.3.2	Klassenübersicht	38
4.3.3	Detailbeschreibung der Klasse WorkflowParser	39
4.3.4	Detailbeschreibung der Klasse Workflow	39

4.3.5	Detailbeschreibung der Klasse SubversionConnector	41
4.4	Anwenderschnittstelle	42
4.4.1	Struktur von Arbeitsablaufsbeschreibungen	43
4.4.2	Kommandozeilenparameter	48
4.4.3	Konfigurationsdateien	48
4.5	Subversion-Integration	50
4.6	Erweiterungsmöglichkeiten	51
4.7	Zusammenfassung	52
5	Validierung	53
5.1	Beschreibung des Anwendungsfalls zur Sequenzierung eines Mausgenoms	53
5.2	Umsetzung des Anwendungsfalles	54
5.2.1	Vorbereitungsarbeiten	54
5.2.2	Erstellung der Workflowbeschreibung	55
5.2.3	Ausführen des Arbeitsablaufs	56
5.3	Resultate	57
5.4	Verbesserungspotential	58
5.4.1	Löschen aus dem Repository	60
5.4.2	Überprüfung des freien Arbeitsspeichers	60
5.4.3	Verifikation von Resultaten	61
5.4.4	Automatische Fehlerbehandlung	61
5.4.5	Datenzentrierter Ansatz	62
5.5	Zusammenfassung	62
6	Schlussbetrachtungen	63
6.1	Ergebnisse der Arbeit	63
6.2	Ausblick	64
A	Next Generation Sequencing Analysis Workflow	66
B	Umsetzung des Next Generation Sequencing Analysis Workflow	72
	Literatur	77

Abbildungsverzeichnis

1.1	Workflow zu Projektbeginn.	2
1.2	Skizzen zur Anforderungsanalyse.	3
2.1	Architektur von YAWL (nach [van der Aalst 04])	14
2.2	Beispielworkflow in YAWL	14
2.3	Architektur von jBPM (nach [Community 06])	15
2.4	Beispielworkflow in jBPM in graphischer und XML-Darstellung	17
2.5	Architektur von Taverna (nach [Paolo Missier 10])	18
2.6	Beispielworkflow in Taverna	19
2.7	Architektur von Triana (nach [Taylor 04])	21
2.8	Beispielworkflow in Triana	21
3.1	Anwendungsfalldiagramm	26
4.1	Screenshot der Entwicklungsumgebung „Eclipse“	31
4.2	Beispielausgabe mit Apache Commons CLI formatiert.	33
4.3	Darstellung der Paketstruktur.	37
4.4	Klassendiagramm.	38
4.5	Sequenzdiagramm.	40
5.1	Schematischer Workflow.	54
5.2	Screenshot des WorkflowExecutor bei der Ausführung	57

Tabellenverzeichnis

4.1	Auflistung möglicher genereller Aktivitäts-Parameter	44
4.2	Auflistung möglicher Aktivitäts-Parameter für den Task execTask . . .	44
4.3	Auflistung möglicher Aktivitäts-Parameter für den Task dbTask	45
4.4	Auflistung möglicher Aktivitäts-Parameter für die Aktivität iteratePar- allelTask	45

Algorithmenverzeichnis

1	XML-Darstellung eines einzelnen Triana-Tasks	22
2	Vereinfachtes Beispiel zur Dependency Injection mit Spring	32
3	Konfigurationsbeispiel für log4j.	35
4	Auszug aus dem Apache Ant buildfile.	36
5	Exemplarischer Arbeitsablauf	43
6	Arbeitsablauf mit iterativer Parallelausführung	46
7	Arbeitsablauf mit Aktivitätswiederholung	46
8	Arbeitsablauf mit Parallelaktivitäten	47
9	Beispiel für taskbeans.xml.	49
10	Einzelner Task aus dem Referenzworkflow	56
11	Workflow mit Parametrisierung	58
12	Workflow ohne Parametrisierung	59

Kapitel 1

Einführung

Wissenschaftliche Forschung im Bereich der Biologie hat mit der Einführung von fortschrittenen Computertechniken einen enormen Schub erhalten, der sogar zur Entwicklung einer komplett neuen Forschungsrichtung geführt hat: *die Bioinformatik*. Diese relativ junge Forschungsdisziplin beschäftigt sich unter anderem mit der Sequenzierung von DNA und Proteinen, was aufgrund der zu bearbeitenden Datenmengen erst durch den koordinierten Einsatz von Datenbanken und Großrechnerverbünden möglich geworden ist.

Einen zusätzlichen Komplexitätsfaktor bilden in diesem Umfeld die verbreiteten unterschiedlichen Datenformate, die vor der Bearbeitung oftmals erst aufwändig konvertiert werden müssen, um überhaupt in der notwendigen Form vorzuliegen. Darüber hinaus müssen aufgrund möglicher wechselnder Zugangsbeschränkungen (etwa durch Kommerzialisierung), Änderungen an bestehenden Datenbeständen, aber auch um Datenverluste zu verhindern, lokale Kopien angelegt werden, um erzielte Ergebnisse langfristig reproduzieren zu können. Darüber hinaus besteht der wissenschaftliche Anspruch, alle Forschungsergebnisse inklusive der Zwischenberechnungen auch über lange Zeiträume reproduzieren zu können und für die Beschleunigung weiterer Arbeitsabläufe bestehende Zwischenergebnisse wiederzuverwerten.

Dies führt in weiterer Folge zu Arbeitsabläufen, deren Verwaltung und Ausführung derzeit umständlich zu handhaben sind, oftmals jedoch den Einsatz von hochspezialisierten Werkzeugen nicht rechtfertigen. Deshalb werden Ablaufbeschreibungen in diesem Komplexitätsgrad durch hohen personellen Einsatz in Handarbeit in Laborbüchern dokumentiert, was eine erhöhte Fehlerquelle darstellt und im Falle der raschen Nachvollziehbarkeit durch nicht direkt in das jeweilige Projekt involvierte Personen oftmals

zu zusätzlichem Arbeitsaufwand führt.

Ziel dieser Arbeit¹ ist es somit, in diesem Umfeld für die beteiligten Wissenschaftler durch die Konzeption und Entwicklung einer Anwendung zur Beschreibung, Ausführung und langfristigen Speicherung von Arbeitsabläufen eine gewisse Arbeitserleichterung zu schaffen.

1.1 Problembeschreibung

Besondere Aufmerksamkeit hinsichtlich der Umsetzung dieser Arbeit ist auf jedem Fall der Interdisziplinarität der beteiligten Forschungsgebiete zu widmen. Forscher im Umfeld der Bioinformatik sind oftmals entweder Biologen mit Informatikkenntnissen oder Informatiker mit rudimentärem Biologiehintergrund. Dadurch ergeben sich oftmals unterschiedliche Auffassungen hinsichtlich der Komplexität von Änderungswünschen, aber auch Verständnisprobleme für die zugrundeliegenden Anforderungen. Während der Entwicklungszeit der in dieser Arbeit vorgestellten Applikation wurde diesem Umstand durch lange Gespräche Rechnung getragen, in denen der Kern der gewünschten Anforderungen herausgearbeitet wurde. Eine wesentliche Erleichterung für den Projektfortschritt während der Implementierung war der Einsatz von *Rapid Prototyping*², sodass bei Folgetreffen immer wieder neu umgesetzte Funktionen präsentiert und diskutiert werden konnten.

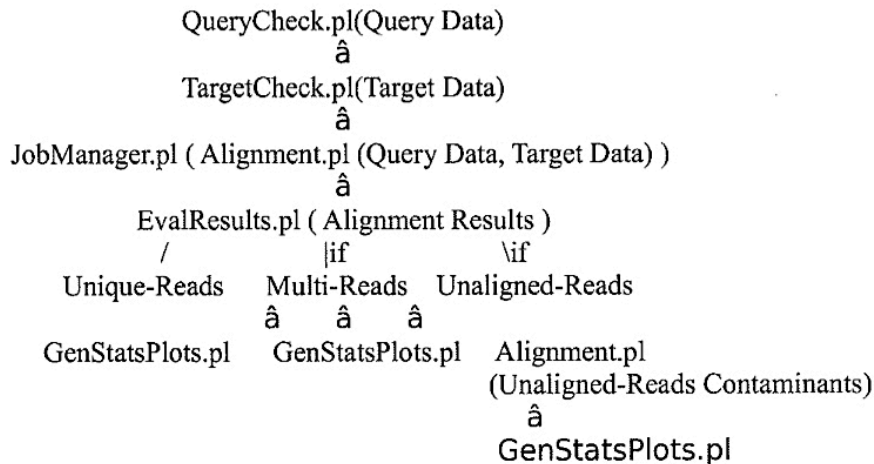


Abbildung 1.1: Workflow zu Projektbeginn.

¹Der vorliegende Text ist auf Basis des Latex-Templates zu [Gockel 08] erstellt.

²vgl. für Details http://www.ergo-online.de/site.aspx?url=html/software/↔software_entwicklung_prototyp/protot_konzept.htm, abgerufen am 05. Sep. 2010

Zu Beginn der Anforderungsanalyse existierte bereits eine graphische Darstellung (vgl. Abb. 1.1) eines exemplarischen Workflows, der als Einstiegspunkt für die weiteren Anforderungen diente. Darin war bereits erkennbar, dass die Ausführungsreihenfolge vorgegeben ist, und existierende Skripte weiterverwendet werden sollten.

Über verschiedene Zwischenschritte (vgl. Abb. 1.2) wurde erarbeitet, dass ein Zugriff auf ein bestehendes Subversion-Repository notwendig ist, und alle einzelnen Aktivitäten durch Parameter steuerbar sein müssen, um die explorative Natur der Forschungstätigkeit zu unterstützen.

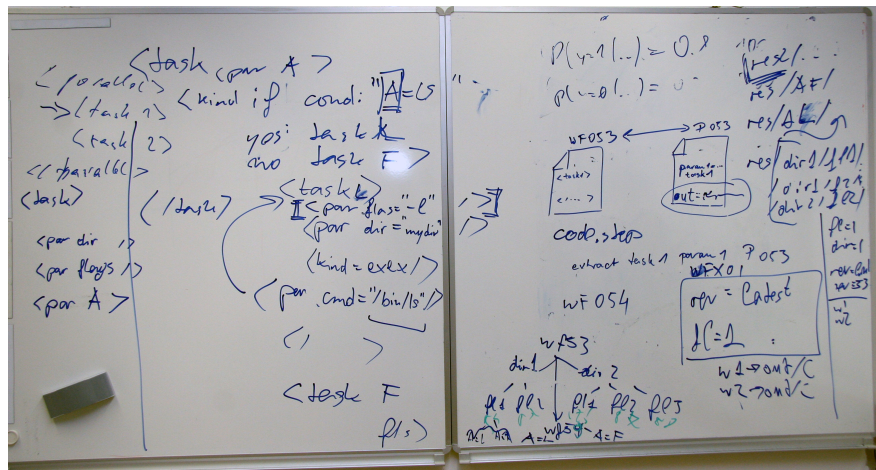


Abbildung 1.2: Skizzen zur Anforderungsanalyse.

Letztendlich hat sich das Tool als Folge dieses iterativen Prozesses zur Anforderungsanalyse von einem „simplen“ Werkzeug zum Ausführen von bestehenden Skripten hin zu einer Applikation entwickelt, die über die aufeinanderfolgende Abarbeitung von Kommandos hinaus die Komplexität der explorativen Forschungstätigkeit unterstützt. Dafür wird darauf Rücksicht genommen, dass während des Prozesses zur Erstellung eines Workflows mehrere Zwischenschritte notwendig sein können, die unterschiedliche Parameterwerte oder überhaupt neue Faktoren berücksichtigen. Diese Unterstützung eines „trial and error“-Verfahrens wird insbesondere durch die Wiederverwendung von identen Zwischenergebnissen betont.

1.2 Aufgabenstellung und Anwendungsszenario

Die Rahmenbedingungen, die der erstellten Applikation zugrundeliegen, entspringen der täglichen Praxis der Forscher des Chairs of Bioinformatics der Universität für Bo-

denkultur³ und können wie folgt kurz umrissen werden:

Wiederverwendung und Orchestrierung von bestehenden Skripten Es muss ohne zusätzlichen Aufwand möglich sein, bereits existierende Skripte einzubinden und in unterschiedlichen Konstellationen neu zu kombinieren.

Schlankheit Das Werkzeug muss ohne großen Aufwand installierbar, einsetzbar und wartbar sein. Das Hauptaugenmerk der Forscher muss weiterhin auf dem Inhalt der Arbeitsabläufe liegen können, ohne zu sehr durch administrative Tätigkeiten abgelenkt zu werden.

Benutzbarkeit Der Einsatz des Tools muss die aktuelle Situation maßgeblich verbessern, indem bekannte Arbeitsabläufe unterstützt und neue Wege eröffnet werden.

Versionsmanagement Aufgrund eines bereits vorhandenen Subversion-Repositories ist dessen Unterstützung der Vorzug vor anderen Datenspeicherlösungen (etwa Datenbanken) zu geben.

Kommandozeilenmodus Das zu erstellende Werkzeug soll insofern nicht mit bestehenden Workflowmanagementsystemen konkurrieren, indem es konsequent auf eine graphische Benutzeroberfläche zur Erstellung und Wartung von Arbeitsabläufen verzichtet.

Um diese theoretischen Anforderungen auch in der Praxis überprüfen zu können, wurde die ausführliche textuelle Beschreibung eines realen Arbeitsablaufes herangezogen, der viele der typischen Problemstellungen beinhaltet. Für die vollständige Version dieses Ablaufbeschreibung sei auf Anhang A ab Seite 66 verwiesen, eine kurze Übersicht über die darin enthaltenen Teilaufgaben kann wie folgt skizziert werden:

Bearbeitung großer Datenmengen Die Applikation muss in der Lage sein, mit großen Datenmengen (der vorgegebene Arbeitsablauf umfasst (trotz reduziertem Datenset) eine Größe von etwas mehr als 10 GB) effizient umgehen zu können bzw. darf für bestehende Skripte keinen zusätzlichen Flaschenhals darstellen.

voneinander abhängige Teilaufgaben Die Gliederung des Arbeitsablaufs sieht vor, dass zuerst die notwendigen Datenpakete von verschiedenen über das Internet verfügbaren Servern heruntergeladen werden, anschließend werden diese entpackt und mittels unterschiedlicher Software bearbeitet. Diese Abhängigkeit muss klar modelliert werden können.

³siehe <http://www.biotec.boku.ac.at/bioinf.html>, abgerufen am 13. August 2010

Parallelabläufe Um die vorhandene Infrastruktur möglichst optimal ausnutzen zu können, besteht bei einigen Teilaufgaben die Möglichkeit des parallelen Ablaufs. Eine gutes Beispiel für eine sinnvolle Parallelisierung wäre etwa das Herunterladen der Datenpakete.

Parametrisierbare Aufgaben Manche Aufgaben erfordern das mehrmalige Ausführen von gleichen Programmen mit unterschiedlichen Parametern, etwa das Entpacken der Daten in verschiedenen Dateien. Solche Tasks sollen pro Arbeitsablauf nur einmal auftreten.

Explorative Gestaltung von Bioinformatik-Workflows Eine implizite Arbeits erleichterung durch die Beschleunigung des Ablaufs ist die Wiederverwendung von bereits existierenden (Zwischen-) Ergebnissen. So ist es etwa sinnvoll, bei einem etwaigen zweiten Durchlauf des gleichen Workflows bereits heruntergeladene Datenpakete nicht nochmals zu kopieren, sondern die Bestehenden erneut zu verwenden. Das Programm soll solche Situationen erkennen und darauf entsprechend reagieren.

1.3 Ergebnisse

Der Einsatz der im Rahmen dieser Diplomarbeit erstellten Applikation hat sowohl mit selbsterzeugten Testarbeitsabläufen als auch mit dem exemplarischen Referenzwork flow (siehe Anhang A ab Seite 66 für die textuelle Beschreibung, Anhang B ab Seite 72 für die Umsetzung) interessante Ergebnisse geliefert. So hat sich etwa die Einarbeitungszeit als sehr gering herausgestellt (es reichte die Betrachtung eines Beispiels zur selbstständigen Verfassung des Referenzarbeitsablaufs) und die implizite Dokumentation durch die verwendete Beschreibungsstruktur wurde positiv hervorgehoben. Als wichtiger Beitrag kann auch die Umsetzung von parametrisierbaren Arbeitsaufgaben angesehen werden, mit deren Hilfe der Umfang von Arbeitsabläufen deutlich verringert werden kann.

1.4 Aufbau der Arbeit

In Kapitel 2 wird das notwendige Grundverständnis für die vorliegende Arbeit vermittelt und gleichzeitig ein Einblick in den aktuellen Stand der Technik gegeben. Dabei werden unter anderem Begriffe für den aktuellen Kontext erläutert bzw. im Falle

möglicher Überschneidungen mit anderen Fachgebieten zu diesen abgegrenzt. Allgemeine Fachbegriffe, die zum speziellen Verständnis von Java-Programmcodes notwendig sind, werden jedoch als Voraussetzung angesehen und nicht im Rahmen dieser Ausarbeitung erläutert. Für das Verständnis dieser Begriffe sei auf Fachliteratur zum Thema verwiesen.

In den Abschnitten 2.2 „Workflowsysteme aus dem Umfeld des Geschäftsprozessmanagements“ und 2.3 „Workflowsysteme aus dem Umfeld der Bioinformatik“ wird ein kleiner Einblick in aktuelle Systeme zur Definition und Ausführung von Arbeitsabläufen gegeben, ohne jedoch den Anspruch auf Vollständigkeit zu erheben. Die Auswahl dieser exemplarischen Systeme erfolgte aufgrund ihrer beispielgebenden Implementierung bzw. wegen ihrer weiten Verbreitung.

Nach dem Überblick über vorhandene Referenzsysteme erfolgt im nachfolgenden Kapitel 3 die Vorstellung der Architektur der Applikation. Dabei wird zuerst auf das Systemumfeld eingegangen, und anschließend werden im Abschnitt 3.2 „Anforderungen“ die Grundlagen im Detail angeführt. Basierend auf diesen Anforderungen wird im nächsten Abschnitt die grundsätzliche Architektur vorgestellt, wobei drei primäre Bereiche herausgearbeitet werden.

Das nächste Kapitel 4 liefert die Beschreibung der konkreten Umsetzung der vorgestellten Applikation und geht dabei auf unterschiedliche Gesichtspunkte ein. Zuerst werden im Abschnitt 4.1 Entscheidungen, die im Rahmen der Implementierung getroffen wurden, erläutert. Abschnitt 4.2 dient als Referenz auf eingesetzte Softwarebibliotheken, während die Programmstruktur in Abschnitt 4.3 erläutert wird. Darauf folgt im Abschnitt 4.4 eine Übersicht über die Anwenderschnittstelle mit Beschreibungen der verwendeten Struktur von Arbeitsabläufen, möglichen Kommandozeilenparametern und der Einstellung von Konfigurationsdateien. Den Abschluss des Kapitels bildet Abschnitt 4.6, indem mögliche Erweiterungen des bestehenden Programmcodes aufgezeigt werden.

Das vorletzte Kapitel 5 dient der Beschreibung der Validierung der vorgelegten Lösung. Dabei wird zuerst der zugrundeliegende beispielhafte Geschäftsfall in Abschnitt 5.1 beschrieben, anschließend wird im Abschnitt 5.3 das erzielte Ergebnis evaluiert. Im darauffolgenden Abschnitt 5.4 wird auf vorhandene Schwächen und Verbesserungspotential eingegangen, die sich aus der Evaluierung des Programms ergeben haben. Eine kurze Zusammenfassung bildet den Abschluss des Kapitels.

Das letzte Kapitel 6 „Schlussbetrachtungen“ widmet sich abschließend den Schlussfol-

gerungen aus den erreichten Resultaten, indem zuerst in Abschnitt 6.1 die Ergebnisse der Arbeit erläutert werden, anschließend liefert Abschnitt 6.2 einen Ausblick auf zukünftige Versionen.

Kapitel 2

Grundlagen und Stand der Technik

Dieses Kapitel gibt eine erste Einführung in die adressierte Thematik. Es werden wichtige und möglicherweise unklare Begriffe definiert und anschließend ein Überblick über den aktuellen Stand der Technik geliefert. Dabei wird die Situation aus zwei unterschiedlichen Sichtweisen betrachtet, um ein möglichst breites Spektrum abzudecken: Zuerst werden existierende Workflowsysteme aus dem Blickwinkel des Geschäftsprozessmanagements analysiert, da diese durch die adressierte Zielgruppe eine größere Verbreitung und dementsprechend auch eine größere Anzahl von unterstützenden Entwicklern haben.

Als zweite Gruppe der betrachteten Systeme dienen Workflowsysteme aus dem Umfeld der Bioinformatik, die zwar durch ihre Spezialisierung sehr gut für ihren Einsatzzweck geeignet sind, jedoch aufgrund ihres relativen Nischendaseins (im Vergleich zur Verbreitung von Workflowsystemen aus der ersten Gruppe) eine weniger breite Unterstützergruppe haben.

2.1 Begriffsdefinitionen

Die Bearbeitung und Ausführung von Arbeitsabläufen steht in den letzten Jahren in engem Zusammenhang mit verschiedenen Computertechnologien, wobei zu beachten ist, dass die dabei verwendeten Begriffe nicht ausschließlich in diesem Kontext verwendet werden. Aus diesem Grund ist es notwendig, für den aktuellen Einsatzbereich die notwendigen Abgrenzungen und Definitionen zu geben.

2.1.1 Geschäftsprozessmanagement

Aufgabe und Ziel des *Geschäftsprozessmanagements* (engl. business process management, BPM) ist es, aus einer übergeordneten unternehmensweiten Sicht auf vorhandene Geschäftsprozesse (vgl. Abs. 2.1.2) Verbesserungspotentiale zu identifizieren und umzusetzen. Dabei kommen unter anderem klassische Elemente der Betriebswirtschaft wie etwa die Auswertung von erhobenen Kennzahlen und des Rechnungswesens - beispielsweise die finanzielle Bewertung durch die Prozesskostenrechnung - zum Einsatz, die durch Informationstechnologie unterstützt werden.

Vor allem die durch verstärkten IT-Einsatz erlangte Effizienz und die dadurch erreichbaren Verbesserungen haben in den letzten Jahren zu einer neuen Sensibilisierung für das Thema geführt, sodass eine Vielzahl neuer Produkte (genauso wie Aktualisierungen bestehender Produkte) auf den Markt drängten. Obwohl diese Applikationen ihren Fokus eher auf der Verbesserung von bestehenden Prozessen haben, können sie hinsichtlich ihrer Ausführungslogik und ihres Funktionsumfanges auch als Referenz für wissenschaftliche Arbeitsabläufe dienen.

2.1.2 Geschäftsprozess

Der Begriff des *Geschäftsprozesses* ist sowohl in der Literatur als auch im gängigen Sprachgebrauch sehr breit definiert und umreißt im Allgemeinen die Summe der Aktivitäten zur vollständigen Erledigung einer genau definierten Arbeitsaufgabe:

Allgemein ist ein Geschäftsprozeß eine zusammengehörende Abfolge von Unternehmungsverrichtungen zum Zweck einer Leistungserstellung. Ausgang und Ergebnis des Geschäftsprozesses ist eine Leistung, die von einem internen oder externen „Kunden“ angefordert und abgenommen wird.

Häufig wird von einem Geschäftsprozeß auch ein wesentlicher Beitrag zur Wertschöpfung der Unternehmung verlangt oder ein Kundenbezug bei der Leistungserstellung, um ihn mit einer gewissen Bedeutung, also auch einer größeren Anzahl von Verrichtungen, zu versehen [Scheer 02].

Im Gegensatz dazu wird wenige Jahre später beim Definitionsversuch des identen Begriffs sehr klar deutlich, dass in letzter Zeit die Abwicklung von Geschäftsprozessen stark durch den Einsatz von Computern geprägt ist:

Ein Geschäftsprozess besteht aus einer zusammenhängenden abgeschlos-

senen Folge von Tätigkeiten, die zur Erfüllung einer betrieblichen Aufgabe notwendig sind. Die Tätigkeiten werden von Aufgabenträgern in organisatorischen Einheiten unter Nutzung der benötigten Produktionsfaktoren geleistet. Unterstützt wird die Abwicklung der Geschäftsprozesse durch das Informations- und Kommunikationssystem IKS des Unternehmens [Staud 06].

Durch diese Änderung in der Auffassung des Begriffs entsteht in weiterer Folge die Notwendigkeit, die eher generische¹ Bedeutung des Begriffs „Geschäftsprozess“ weiter zu verfeinern, um dadurch eine inhaltliche Unterscheidung zu primär computerunterstützten Arbeitsabläufen zu schaffen.

2.1.3 Workflow

Eine Möglichkeit zur begrifflichen Unterscheidung von Geschäftsprozessen und computerunterstützten Arbeitsabläufen liefert die Workflow Management Coalition in ihrem *Workflow Reference Model* [Hollingsworth 95]. Darin wird die folgende einführende Erklärung bzw. Definition gegeben:

Workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal.

Definition - Workflow

The computerised facilitation or automation of a business process, in whole or part [Hollingsworth 95].

Die enge Verknüpfung mit dem Begriff des Geschäftsprozesses wird auch bei der in [Gadatsch 08]² gelieferten Definition des Begriffs ersichtlich:

Ein Workflow ist ein formal beschriebener, ganz oder teilweise automatisierter Geschäftsprozess. Er beinhaltet die zeitlichen, fachlichen und ressourcenbezogenen Spezifikationen, die für eine automatische Steuerung des Arbeitsablaufes auf der operativen Ebene erforderlich sind. Die hierbei anzustoßenden Arbeitsschritte sind zur Ausführung durch Mitarbeiter oder durch Anwendungsprogramme vorgesehen [Gadatsch 08].

¹für eine ausführliche Diskussion siehe [Staud 06] und [Rosenkranz 06]

²ebd. werden auch weitere Definitionen des Begriffs erläutert.

Auch für die im aktuellen Kontext zur Anwendung gelangende Definition gelten die meisten der oben angeführten Punkte. So wird hier unter einem Workflow die formale Beschreibung einer oder mehrerer Aktivitäten verstanden, die im Rahmen der Abarbeitung automatisch in der Ausführungsumgebung exekutiert werden kann.

2.1.4 Workflowmanagementsystem

Der Begriff des *Workflowmanagementsystems* verlässt die statische Ebene der reinen Beschreibung von Arbeitsabläufen und fügt eine dynamische Ausführungsdimension hinzu. Damit einher geht gleichzeitig die Anforderung der (formal korrekten) Modellierung von Arbeitsabläufen und deren Überwachung und Steuerung während der Ausführung. [Gadatsch 08] liefert basierend auf verschiedenen Auffassungen die folgende Definition des Begriffs:

Ein Workflow-Management-System ist ein anwendungsunabhängiges, dem Middlewarebereich zuzuordnendes Softwaresystem, das die Modellierung, die Ausführung und das Monitoring von Workflows, sowie gegebenenfalls weitere Funktionen wie die Simulation und die Analyse von Workflows, unterstützt; insbesondere ist es in der Lage, (semi-)formale Workflow-Spezifikationen zu interpretieren, die Ausführung von Prozessschritten durch die vorgesehenen Aktivitätsträger - Mitarbeiter oder Anwendungsprogramme - zu veranlassen und gegebenenfalls erforderliche Arbeitsanweisungen, Werkzeuge, Anwendungsprogramme, Informationen und Dokumente bereitzustellen Gehring (1998) [Gehring 98]³.

2.1.5 Versionsverwaltung

Der Begriff der *Versionsverwaltung* entstammt dem Umfeld der Softwareentwicklung, wo sicherlich auch der Haupteinsatzzweck liegt, jedoch nicht der einzige. Zuallererst dient Versionskontrolle dazu, Änderungen an Dateien auch nachträglich nachvollziehbar zu machen, indem deren historische Versionen gespeichert werden. Darüber hinaus hilft sie dabei, die Zusammenarbeit zwischen mehreren Softwareentwicklern zu vereinfachen bzw. teilweise überhaupt erst zu ermöglichen, indem ein konsistenter (im Sinne von „zu einem beliebigen Zeitpunkt reproduzierbaren“) Entwicklungsstand in einem zentralen Softwareverzeichnis (Repository) verwaltet wird. Im aktuellen Kontext

³zitiert nach [Gadatsch 08]

bietet Versionsverwaltung eine Möglichkeit, auf erprobte Art und Weise ein System einzuführen, dass die Historisierung von Arbeitsabläufen ermöglicht. Um diese Aufgaben zu erfüllen, müssen Versionskontrollsysteme verschiedene Eigenschaften unterstützen, die in Folge kurz erläutert werden.

Entwicklungszweige Die Unterstützung von parallelen Entwicklungszweigen („*branches*“) ermöglicht es, für ein Projekt verschiedene Seitenäste, etwa für den letzten aktuellen Entwicklungsstand und bereits veröffentlichte Revisionen anzulegen. Dadurch entsteht eine baumartige Struktur, in der der Hauptpfad den Stamm bildet und die unterschiedlichen Ableger die Äste und Blätter.

Versionsbezeichnung Zur leichten Auffindbarkeit verschiedener Ableger im Repository werden diese meist mit eindeutigen Bezeichnungen („*tag*“) versehen. Beispiele für geläufige Namen sind *head* bzw. *trunk* für den aktuellsten Entwicklungsstand und Releasebezeichnungen wie „1.0“.

Synchronisation Durch die Möglichkeit des gleichzeitigen Zugriffs auf Dateien durch verschiedene Entwickler („*concurrent access*“) besteht die Notwendigkeit, Änderungen zu synchronisieren. Dabei können aktuelle Versionsverwaltungssysteme viele Konflikte automatisch auflösen, teilweise sind jedoch manuelle Eingriffe (vor allem auch bei binären Dateien) notwendig. Nach erfolgreicher Synchronisation besteht ein konsistenter Stand zwischen einem lokalen Entwicklungszweig und dem zentralen Repository.

Hinzufügen von Dateien Es werden nicht automatisch alle Dateien eines Projektes unter Versionskontrolle gestellt, sondern erst durch manuelles Hinzufügen durch einen Entwickler. Diese Aktion bezeichnet man als „*checkin*“ bzw. „*commit*“. Nach dem erfolgreichen Hinzufügen einer Datei ist diese durch andere Entwickler bearbeitbar.

Editieren von Dateien Nach dem ersten Hinzufügen einer Datei kann diese durch die Aktion „*checkout*“ durch einen Entwickler zur Bearbeitung reserviert werden. Nach erfolgter Änderung wird die Datei wieder in das zentrale Repository zurückgestellt, wodurch eine neue Version angelegt wird.

2.2 Workflowsysteme aus dem Umfeld des Geschäftsprozessmanagements

Wie bereits in der einführenden Begriffsdefinition beschrieben, erfährt das Geschäftsprozessmanagement in den letzten Jahren einen rasanten Aufschwung durch den Einsatz von automatisierten Workflowsystemen. Aufgrund mangelnder Standardisierung sind jedoch eine Vielzahl unterschiedlicher Produkte entstanden, die miteinander nicht kompatible Prozessbeschreibungen⁴ verwenden. Die hier ausgewählten Produkte *YAWL* und *JBoss jBPM* zeichnen sich dadurch aus, dass sie bezüglich der verwendeten Syntax eine Vorreiterrolle haben bzw. durch ihre weite Verbreitung in der Industrie eine breite Unterstützung erfahren.

2.2.1 YAWL - Yet Another Workflow Language

Unter dem Akronym *YAWL* (Yet Another Workflow Language)⁵ wird sowohl eine Prozessbeschreibungssprache als auch ein auf dieser Sprache basierendes BPM/Workflow System verstanden.

Die Sprache (zur besseren Unterscheidung oftmals als *YAWL language* im Gegensatz zu *YAWL system* für das Workflow System bezeichnet) basiert unter anderem auf der Analyse von mehr als 30 bestehenden Workflowmanagementsystemen respektive deren Arbeitsablaufsbeschreibungssyntax. Darüber hinaus wurden während der Entwicklung der Sprache unterschiedliche Workflow-Patterns⁶ identifiziert und beschrieben, deren Unterstützung ebenfalls Einfluss in die Sprachdefinition gefunden haben. Im Vergleich bestehender Systeme haben die Autoren die grundlegenden Vorteile von Petri-Netzen gegenüber gewöhnlicher Workflowmanagementsysteme identifiziert, gleichzeitig aber einige Schwächen aufgezeigt, deren Behebung letztendlich zur Entwicklung von *YAWL (language)* geführt haben (vgl. [Van Der Aalst 05]).

Als vollständig implementierte Lösung dient *YAWL system* sowohl als Referenz als auch zur Demonstration der Konzepte, die mit *YAWL language* vorgestellt wurden. Die Architektur (siehe Abb. 2.1) des Systems beinhaltet verschiedene Komponenten, die untereinander zusammenarbeiten. Arbeitsabläufe werden mit Hilfe eines graphischen

⁴<http://www.gridworkflow.org/snips/gridworkflow/space/Workflow+Description+Languages> listet am 25.07.2010 ohne Anspruch auf Vollständigkeit 24 verschiedene „Workflow Languages“ auf

⁵siehe auch <http://www.yawlfoundation.org/>, abgerufen am 13. August 2010

⁶vgl. [van Der Aalst 03]

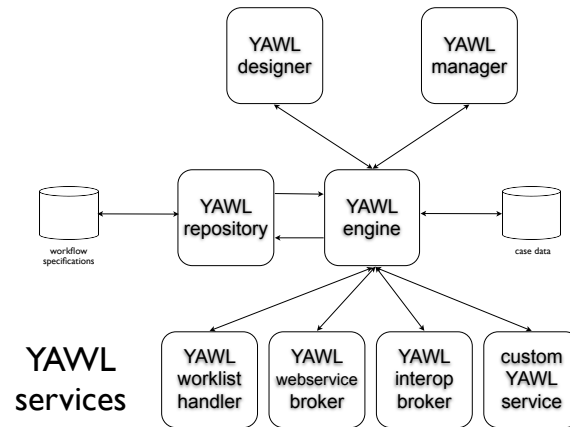


Abbildung 2.1: Architektur von YAWL (nach [van der Aalst 04])

Editors, dem *YAWL designer* erstellt und anschließend dem zentralen Element, der *YAWL engine* übergeben. Dort werden alle notwendigen Überprüfungen durchgeführt, und der verifizierte Arbeitsablauf anschließend im *YAWL repository* abgelegt. Sobald ein Workflow ordnungsgemäß erstellt und überprüft wurde, können in der YAWL engine Instanzen davon erstellt und ausgeführt werden. Um notwendige manuelle Eingriffe (wie etwa für das Löschen einer Workflow-Instanz oder einer Spezifikation) in die Workflow-Umgebung zu ermöglichen, aber auch um Statusinformationen über laufende Workflows zu erhalten, kommuniziert der *YAWL manager* ebenfalls direkt mit der YAWL engine.

Die Interaktion mit der nicht von YAWL kontrollierten Umwelt erfolgt über die vier verschiedenen YAWL services *YAWL worklist handler*, *YAWL webservice broker*, *YAWL interoperability broker* und *custom YAWL services*. Der YAWL worklist handler bildet die Schnittstelle zwischen verfügbaren Arbeitsaufgaben und Benutzern des Systems. Mithilfe des YAWL webservice broker ist es möglich, externe Webservices zu konsumieren, während der YAWL interoperability broker für die Zusammenarbeit mit anderen Workflowsystemen zuständig ist. Das letzte Service dient als generische Schnittstelle zur Kommunikation mit externen Geräten, wie etwa Druckern oder Mobiltelefonen [van der Aalst 04].

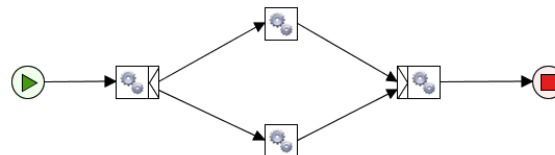


Abbildung 2.2: Beispielworkflow in YAWL

Aufgrund der Unterstützung komplexer Workflow-Patterns wurde bei der Entwicklung

von YAWL konsequent auf den Einsatz eines grafischen Editors (für ein Beispiel eines simplen Workflows in YAWL siehe Abb. 2.2) hingearbeitet, der letztendlich auch eine eigene Komponente des Systems bildet. Daraus resultiert auch - trotz Unterstützung von XML-Standards wie JDom⁷, Saxon⁸ und Xerces⁹ - dass es nicht einfach möglich ist, einen Arbeitsablauf von YAWL manuell zu bearbeiten, was dazu führt, dass das System für die aktuelle Problemstellung nur bedingt einsetzbar ist.

2.2.2 JBoss jBPM

Als zweites Beispiel für die Gruppe der geschäftsprozess-nahen Workflowmanagementsystem dient das weit verbreitete Framework JBoss jBPM. Es wird genauso wie YAWL unter einer Open-Source Lizenz vertrieben und erhebt für sich selbst den Anspruch, die "Brücke zwischen Wirtschaftsanalysten und Entwicklern zu bilden"¹⁰.

Das System ähnelt vom Aufbau her YAWL system, indem es eine zentrale Ausführungskomponente, die sog. *Core Component*, gibt, mit der verschiedene Kontrollmodule kommunizieren (vgl. Abb. 2.3). Die Implementierung erfolgt in der Programmiersprache Java, sodass die Applikation auf unterschiedlichsten Betriebssystemen eingesetzt werden kann. Durch den modularen Aufbau besteht die einfache Möglichkeit, die zentralen Komponenten in bestehende Systeme zu integrieren.

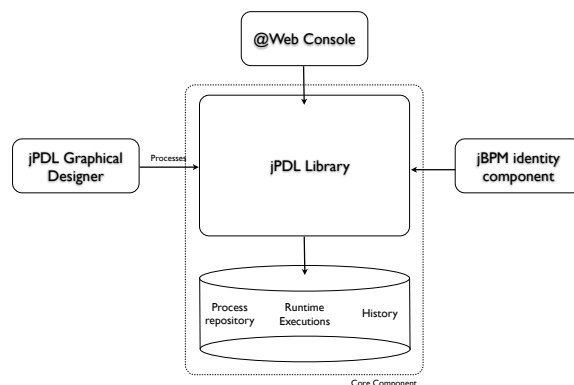


Abbildung 2.3: Architektur von jBPM (nach [Community 06])

Die zentrale Komponente, tlw. auch als core process engine bezeichnet, ist zuständig für die Ausführung von Prozessinstanzen. Dabei werden jedoch nicht nur aktuelle Instanzen verwaltet, auch die Historisierung alter Prozesse erfolgt über diese zentrale

⁷<http://www.jdom.org>, abgerufen am 13. August 2010

⁸<http://saxon.sourceforge.net>, abgerufen am 13. August 2010

⁹<http://xerces.apache.org/xerces-j/>, abgerufen am 13. August 2010

¹⁰Eigenübersetzung nach <http://jboss.org/jbpm>, abgerufen am 23. Juli 2010

Komponente. Ein Vorteil dieser Vorgehensweise ist, dass dadurch gleichzeitig statistische Informationen über die Geschäftsprozesse extrahiert werden können.

An erster Stelle der Module befindet sich der *jPDL Graphical Designer*. Dieser hat den Zweck, mit Hilfe einer graphischen Benutzeroberfläche (die als Plugin für die Entwicklungsumgebung Eclipse¹¹ implementiert ist) die Gestaltung neuer Workflows zu ermöglichen. Dabei kommt im Normalfall die eigene Prozessdefinitionssprache *jBPM Process Definition Language (jPDL)* zum Einsatz, wobei aber auch andere Sprachen unterstützt werden.

Als Komponente zur Steuerung der Ausführung genauso wie zur Überwachung und Administration von Geschäftsprozessen dient die *jBPM web console*. Ähnlich wie beim YAWL manager können hier zur Laufzeit Informationen über Workflows abgerufen werden, um bei entstehenden Engpässen entsprechend reagieren zu können.

Als letzte Komponente dient die *jBPM identity component* dem Management von Benutzern und deren Zugriffsberechtigungen. [Wohed 09]

Die bereits erwähnte primär zum Einsatz gelangende Prozessbeschreibungssprache ist jPDL. Die Syntax der Sprache wird mittels eines XML-Schemas¹² validiert und kann daher auch relativ einfach gelesen werden. Der hierarchische Aufbau eines Prozesses (siehe Abb. 2.2.2 für ein exemplarisches Beispiel in graphischer und textueller Form) ist durch einen Start- und Endknoten gekennzeichnet, dazwischen können verschiedene Aktivitäten stattfinden. Es gibt unterschiedliche Elemente zur Realisierung mehrerer Statusübergänge, wie etwa Parallel-Aktivitäten, Entscheidungsknoten oder Subprozesse.

Anders als bei YAWL können bei Einsatz von jBPM eigene Workflows auch ohne Verwendung des graphischen Editors erstellt und editiert werden, jedoch besteht hier das Problem, dass einige Funktionalitäten nur durch manuelle Programmierung erreicht werden können, was ebenfalls gegen den Einsatz für die aktuelle Problemstellung spricht.

¹¹<http://www.eclipse.org>, abgerufen am 13. August 2010

¹²<http://docs.jboss.org/jbpm/xsd/jpdl-3.1.xsd>, abgerufen am 13. August 2010

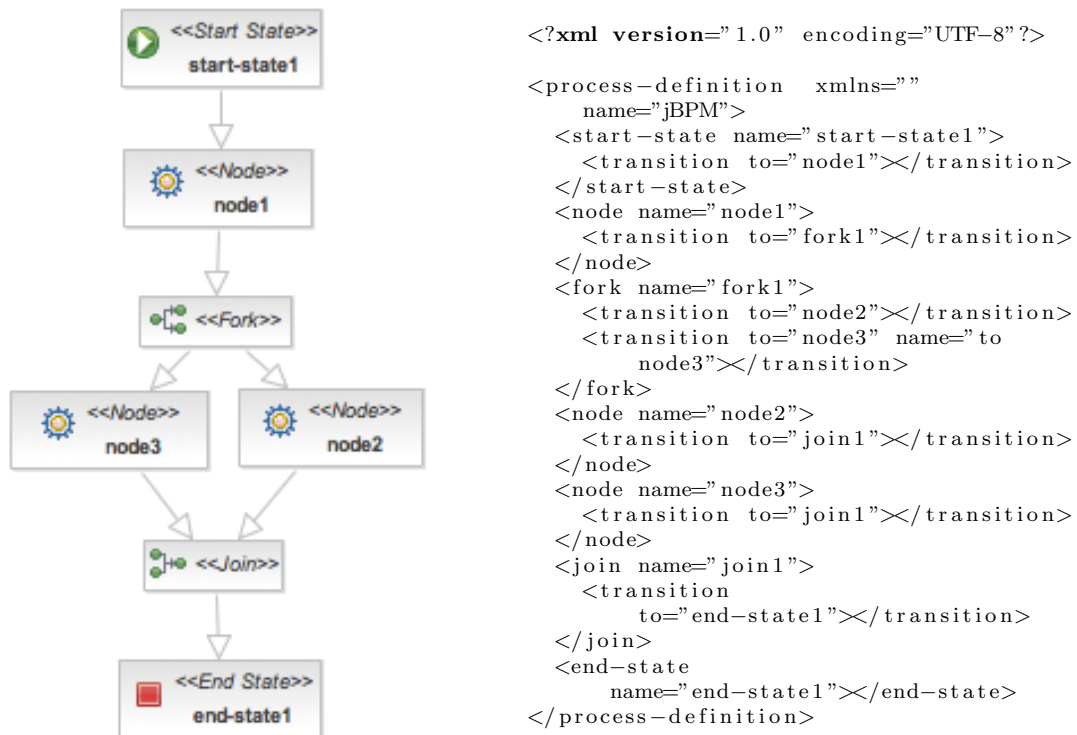


Abbildung 2.4: Beispielworkflow in jBPM in graphischer und XML-Darstellung

2.3 Workflowsysteme aus dem Umfeld der Bioinformatik

Bereits im Jahr 1996 wurde von Brayner und Weske der Einsatz eines (aus damaliger Sicht) *state-of-the-art* Workflow-Management-Systems „mit dem Ziel der Modellierung sowie der kontrollierten und effizienten Ausführung von Geschäftsprozessen“ [Brayner 96] für die Anwendung in der Molekularbiologie hin untersucht, mit dem ernüchternden Ergebnis, dass das Produkt für diesen Einsatzzweck einigen Einschränkungen unterliegt. Aufgrund der langen Zeit seither ist diese Aussage nicht mehr voll belastbar, jedoch sind nach wie vor einige Unterschiede zwischen wissenschaftlichen Arbeitsabläufen (vgl. [Kreil 01]) und Geschäftsprozessen auszumachen.

Um diesem Umstand gerecht zu werden, gibt es für diesen Anwendungszweck ebenfalls eine große Anzahl an Produkten (für eine kleine Übersicht siehe auch [Kreil 01]), die teilweise sehr spezielle Anwendungszwecke erfüllen. Einen etwas breiteren Einsatzbereich umfassen die beiden hier exemplarisch vorgestellten Werkzeuge *Taverna* sowie *Triana*.

2.3.1 Taverna

Das Open-Source Workflow-Management-System *Taverna* [Hull 06] wurde von der Forschungsgruppe myGrid¹³ entwickelt und liegt derzeit in der Version 2.2¹⁴ vor. Das Werkzeug ist in der Programmiersprache Java implementiert und erhebt für sich selbst den Anspruch, auch Anwendern mit geringen Vorkenntnissen in Computerprogrammierung in die Lage zu versetzen, komplexe wissenschaftliche Arbeitsabläufe zu erstellen und ablaufen zu lassen.

Die ursprüngliche Intention zur Entwicklung von Taverna war der Wunsch, das Zusammenstellen und Ausführen von sog. *in silico*¹⁵ Experimenten derart zu vereinfachen, dass mithilfe einer graphischen Benutzeroberfläche der Aufruf von verschiedenen, als Webservice angebotenen Diensten, ermöglicht wird.

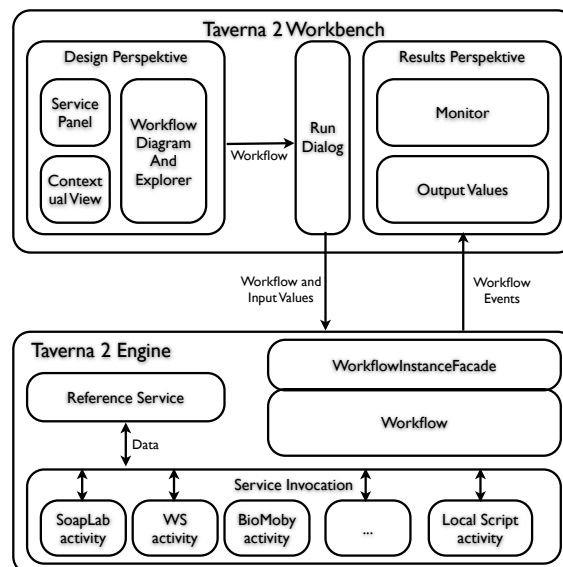


Abbildung 2.5: Architektur von Taverna (nach [Paolo Missier 10])

In Taverna erfolgt die Umsetzung dieses Wunsches so, dass ein Workflow als Graph von sog. Prozessoren dargestellt wird. Jeder dieser Prozessoren verfügt über zumindest eine Eingabe- und Ausgabeschnittstelle, über die die verarbeiteten Daten weitergeleitet werden. Die Zusammenstellung dieses Graphen erfolgt durch Auswahl von bereits existierenden Services in der Design Perspektive (vgl. Abb. 2.5), die untereinander

¹³<http://www.mygrid.org.uk/>, abgerufen am 13. August 2010

¹⁴die jeweils aktuelle Version ist unter <http://www.taverna.org.uk/> (zuletzt abgerufen am 13. August 2010) zu finden

¹⁵Der Kunstbegriff *in silico* entstand zu Beginn der 1990er Jahre in Anlehnung an die etablierten Ausdrücke *in vivo* und *in vitro* zur Beschreibung von Experimenten, die im Computer (deren Mikrochips meist aus Silizium hergestellt sind) ablaufen.

verbunden werden. Nach erfolgter Zusammenstellung wird ein Workflow auf syntaktische Korrektheit überprüft, anschließend erfolgt die Instanziierung und Ausführung in der Taverna Engine. Die Ergebnisse werden wiederum an die Results Perspective zurückgeliefert und können dort auch während der Laufzeit überwacht werden.

Im Hintergrund wird die graphische Darstellung (vgl. Abb. 2.6) der Arbeitsabläufe in der eigens entwickelten Sprache *Scufl* (Simple conceptual unified flow language) abgebildet. So wie alle bisher vorgestellten Sprachen basiert auch Scufl auf XML, die Syntax umfasst unterschiedliche Konstrukte zur Umsetzung der Aufgaben. Anders als bisher wird allerdings in Taverna ein Workflow zu einem Objektmodell kompiliert, indem für jeden Prozessor ein entsprechendes Java-Objekt erzeugt wird. Die Übergabe von Daten von einem Prozessor zum nächsten erfolgt anschließend über lokale Methodenaufrufe. Die Ablauflogik sieht vor, dass ein Prozessor dann gestartet werden kann, wenn alle seine Eingabeschnittstellen mit zumindest einem Datenwert befüllt sind. Darüber hinaus existieren einige implizite Geschwindigkeitsoptimierungen, wie etwa die automatische parallele Bearbeitung innerhalb eines Prozessors, wenn an der Eingabeschnittstelle eine Liste von Werten existiert, obwohl dort nur ein Einzelwert erwartet wird.

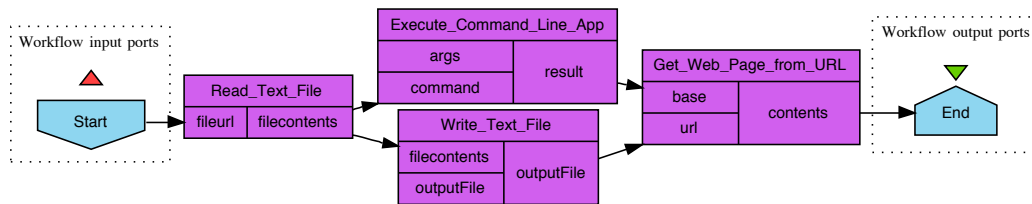


Abbildung 2.6: Beispielworkflow in Taverna

Die drei wichtigsten Konstrukte in Tavernas sind *processor*, *data link* und *coordination constraint*. In ersteren wird der Hauptteil der Arbeit erledigt, indem aus Eingabedaten über einen Transformationsschritt Ausgabedaten produziert werden. Es existiert ein vorgefertigtes Set unterschiedlicher Prozessoren, die verschiedene Aufgaben erledigen. So stehen etwa Typen zum Aufruf von Webservices oder auch zum Ausführen von lokalen Skripten zur Verfügung. Die Verbindung zwischen zwei Prozessoren stellen die data links dar, während coordination constraints dafür zuständig sind, zwei Prozessoren zu verbinden, die keine direkte Datenabhängigkeit haben. Dieser Schritt kann etwa notwendig sein, wenn eine manuelle Priorisierung von Prozessen erforderlich wird [Oinn 04], [Paolo Missier 10].

Trotz der expliziten Ausrichtung von Taverna auf die Konsumation von Webservices ist es relativ einfach möglich, auch Arbeitsabläufe zu erstellen, die etwa lokale Skripte

exekutieren. Durch die breite Verfügbarkeit von unterschiedlichen Services, die auch durch eigene Angebote erweitert werden können, ist Taverna sehr universell einsetzbar. Allerdings ist es wie bei vielen anderen Tools mit ähnlichem Funktionsumfang nicht mehr möglich, neue Workflows ohne Einsatz einer graphischen Benutzeroberfläche zu adaptieren, wenn auch für Taverna ein Modul existiert, dass zumindest die Ausführung bereits existierender Arbeitsablaufsbeschreibungen von der Kommandozeile erlaubt.

2.3.2 Triana

Genauso wie die restlichen hier referenzierten Workflowsysteme wurde *Triana*¹⁶ in der Programmiersprache Java entwickelt und liegt unter einer Open Source Lizenz vor. Triana wurde nicht primär für die Ausführung von Workflows konzipiert, sondern allgemein als „Problem Solving Environment“, das mithilfe verschiedener modularer Komponenten in bestehende Projekte integriert werden kann.

Problem Solving Environments zeichnen sich dadurch aus, dass mit ihrer Hilfe die Zusammenstellung und Ausführung von Applikationen für eine bestimmte Problemstellung ermöglicht wird. Diese Kombination unterschiedlicher Softwareprodukte und Spezialdatenbanken innerhalb eines Forschungsgebietes (und deren Veröffentlichung) ermöglicht es Wissenschaftlern, bereits bestehende Lösungen zu nutzen, anstatt oftmals selbstständig idente Probleme zu lösen. Triana unterscheidet sich insofern von anderen Problem Solving Environments ([Fox 02] bietet eine Übersicht), indem es Unterstützung für die Verteilung der Komponenten in einem Netzwerk anbietet. Die dabei unterstützten Verteilungsmechanismen basieren auf Webservices und P2P Technologien. [Taylor 03]

Die konsequente Unterstützung verteilter Ressourcen wird auch durch die Architektur (vgl. Abb. 2.7) von Triana deutlich: Die Benutzeroberfläche kommuniziert entweder lokal oder über das Netzwerk mit einem *Triana Controlling Service*, das wiederum mit verschiedenen *Triana Engines* oder auch anderen kompatiblen Ausführungsumgebungen verbunden ist. Als weitere Dezentralisierungsmaßnahme können die einzelnen Triana Engines Arbeit an weitere Triana Engines delegieren. [Taylor 04]

Der Aufbau der Benutzeroberfläche von Triana ähnelt dem von Taverna, indem ein graphischer Editor zur Erstellung von Arbeitsabläufen existiert, der eine Auswahl von vor-

¹⁶Die aktuelle Version liegt unter <http://www.trianacode.org/> (zuletzt abgerufen am 13. August 2010) vor.

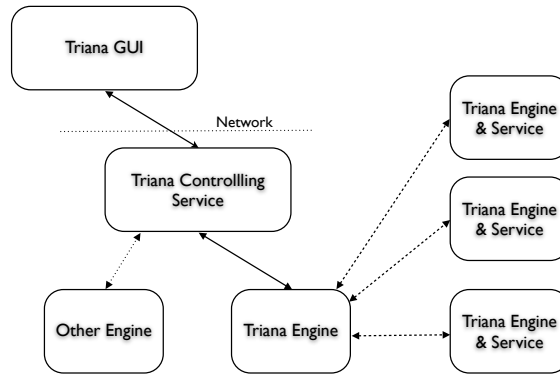


Abbildung 2.7: Architektur von Triana (nach [Taylor 04])

gefertigten Aktivitäten zur Auswahl anbietet. Durch sinnvolle Kombination verschiedener solcher Tasks lassen sich auf einfache Art und Weise komplette Arbeitsabläufe (vgl. Abb. 2.8) abbilden. Die Verbindung zwischen zwei Aktivitäten erfolgt über Konnektoren, wobei bereits während des Erstellens einer solchen Abfolge überprüft wird, ob das Ausgabeformat mit dem Eingabeformat kompatibel ist.

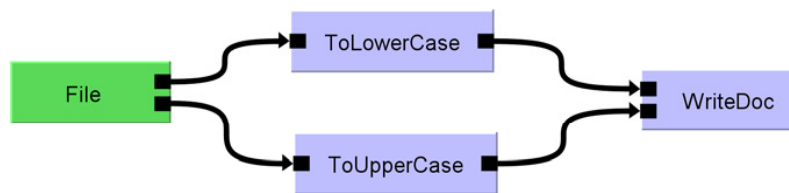


Abbildung 2.8: Beispielworkflow in Triana

Intern werden Arbeitsabläufe in XML abgebildet, wobei jedoch die Lesbarkeit etwa durch notwendige Parameter zur graphischen Darstellung erschwert wird. Obwohl Triana in der Lage ist, andere Formatbeschreibungen wie etwa BPEL4WS zu lesen, wird ein eigenes Format verwendet. Ein Beispiel für einen einzelnen Task des in Abb. 2.8 dargestellten Arbeitsablaufs ist in Listing 1 zu sehen. Durch diese zusätzliche Komplexität wird wie bei den anderen Tools die manuelle Erstellung von Workflows in Triana erschwert.

2.4 Zusammenfassung

In diesem Kapitel wurden zuerst die notwendigen Grundlagen und Begriffsdefinitionen für ein besseres Verständnis dieser Arbeit erläutert. Dabei wurde insbesondere näher

Listing 1 XML-Darstellung eines einzelnen Triana-Tasks

```
<task>
  <toolname>ToLowerCase</toolname>
  <package>Editing.Processing</package>
  <proxy type="Java">
    <param paramname="unitName"><value> triana.tools.ToLowerCase</value></param>
    <param paramname="unitPackage"><value>Editing\Processing</value></param>
  </proxy>
  <renderingHints>
    <renderingHint hint="TaskGraphFactory" proxyDependent="true">
      <param paramname="factory"><value>Default</value></param>
    </renderingHint>
  </renderingHints>
  <inportnum>1</inportnum><outportnum>1</outportnum>
  <inparam /><outparam />
  <input><type>Document</type></input>
  <output><type>Document</type></output>
  <parameters>
    <param name="minOut" type="internal"><value>0</value></param>
    <param name="helpFile" type="internal"><value>DTPTools.html</value></param>
    <param name="popUpDescription" type="internal">
      <value>Converts the input Document to lower case.</value>
    </param>
    <param name="maxIn" type="internal"><value>1</value></param>
    <param name="maxOut" type="internal"><value>2147483647</value></param>
    <param name="guiY" type="gui"><value>4.024390243902439</value></param>
    <param name="guiX" type="gui"><value>1.3008130081300813</value></param>
    <param name="TRIGGER" type="userAccessible"><value>Active</value></param>
    <param name="defaultOut" type="internal"><value>1</value></param>
    <param name="minIn" type="internal"><value>1</value></param>
    <param name="toolVersion" type="internal"><value>3</value></param>
    <param name="defaultIn" type="internal"><value>1</value></param>
  </parameters>
</task>
```

auf Begriffe aus dem Umfeld des Geschäftsprozessmanagements eingegangen.

Darauf folgte ein Überblick über bestehende Systeme zur Be- und Abarbeitung von Arbeitsabläufen, wobei der Fokus zuerst auf Projekten aus dem Umfeld des Geschäftsprozessmanagements gelegen ist, anschließend aber auch der aktuelle Stand aus Sicht der Bioinformatik erläutert wurde. Für alle Beispiele wurde die Architektur dargestellt und ein einführendes Beispiel eines Arbeitsablaufs präsentiert.

Kapitel 3

Architektur

In diesem Kapitel wird sowohl das Systemumfeld, in dem die Applikation vorrangig eingesetzt wird, beschrieben, als auch detaillierter auf die einzuhaltenden Anforderungen eingegangen. Darüber hinaus wird der grundsätzliche Aufbau beschrieben, dessen Umsetzung im nächsten Kapitel dargestellt wird.

3.1 Systemumfeld

Eines der vor der Entwicklung der Applikation festgelegten Ziele war, sich nicht auf eine bestimmte Zielplattform zu beschränken, sondern ein möglichst breites Spektrum von Betriebssystemen zu unterstützen. Dies war ein Mitgrund für die Entscheidung, die Applikation letztendlich in der Programmiersprache Java zu implementieren, da diese auf verschiedensten Plattformen verfügbar ist.

Unabhängig von der möglichen Unterstützung verschiedener Plattformen wird die Applikation zumeist auf diversen Linux-Distributionen zum Einsatz gelangen, da diese in den Labors des Instituts für Bioinformatik an der Boku Wien zur Verfügung stehen.

3.2 Anforderungen

Unabhängig von den impliziten (nichtfunktionalen) Anforderungen wie etwa der Ausführbarkeit des Programms oder der Benutzbarkeit wurden in Vorgesprächen einige weitere Rahmenbedingungen festgelegt, die Auswirkungen auf das Design der Applika-

tion hatten. Dabei ist zu beachten, dass diese impliziten und expliziten Anforderungen das formale Rahmenwerk für die Applikation bilden, jedoch nicht alle Anforderungen mittels überprüfbarer Fakten verifiziert werden können.

3.2.1 Schlankheit und Einsetzbarkeit

Eine der wichtigsten Anforderungen besteht darin, die Forscher nicht durch administrative Tätigkeiten abzulenken, sondern sie in ihrer Arbeit maßgeblich zu unterstützen. Deshalb muss das Tool auch ohne fachlich geschulte Administratoren leicht installierbar, einsetzbar und wartbar sein. Darüber hinaus besteht die Notwendigkeit, dass das Werkzeug aus Akzeptanzgründen die aktuelle Situation maßgeblich vereinfacht, und natürlich bestehende Workflows weiterhin eingesetzt werden können.

3.2.2 Wiederverwendung und Orchestrierung von bestehenden Skripten

Eine Grundvoraussetzung für die Verwendung einer neuen Applikation ist die Möglichkeit der Weiterverwendung bestehender Skripte, da eine vollkommene Neuimplementierung dieser - etwa in einer neuen Programmiersprache - aus zeitlichen Gründen nicht möglich ist. Daraus folgt, dass es ohne Aufwand möglich sein muss, bereits existierende Skripte einzubinden und in unterschiedlichen Konstellationen neu zu kombinieren.

3.2.3 Einbindung externer Applikationen

Die Notwendigkeit, in einem Arbeitsablauf externe Programme ausführen zu können, entsteht direkt aus der vorstehenden Anforderung. Oftmals gelangen in bestehenden Workflows Perl-Skripts zum Einsatz, die auch weiterhin exekutiert werden müssen. Die Entscheidung für eine generische Arbeitsaufgabe (also ein Task, der nicht nur Perl-Skripte abarbeiten kann, sondern etwa auch C-Programme) ergibt sich aus der erweiterten Flexibilität, die aber natürlich verbunden ist mit Einschränkungen hinsichtlich der Auswertung von Rückgabewerten der aufgerufenen Applikationen (diese muss dementsprechend auch generisch erfolgen).

3.2.4 Erweiterbarkeit der Applikation

Die Applikation muss die Möglichkeit bieten, zu einem späteren Zeitpunkt relativ einfach erweitert werden zu können, ohne die Grundfunktionalität anpassen zu müssen. Dies führt zu einem Plugin-Ansatz, wo bestimmte neue Arbeitsaufgaben relativ beliebig zu den Basisfunktionen hinzugefügt werden können.

3.2.5 Versionsmanagement

Um Ergebnisse langfristig speichern zu können und auch Überarbeitungen und Erweiterungen bestehender Arbeitsabläufe automatisiert zu dokumentieren, erfolgt die Übertragung der Daten eines Workflows in ein Versionsmanagementsystem. Aufgrund eines bereits vorhandenen Subversion-Repositories ist dessen Unterstützung der Vorzug vor anderen Datenspeicherlösungen (etwa Datenbanken) zu geben.

Ein zusätzlicher Vorteil, der sich aus dieser Anforderung ergibt, ist eine etwaige Beschleunigung des Ablaufs bei erneuter Abarbeitung eines bestehenden Workflows. Die Wiederverwendung von bereits existierenden (Zwischen-) Ergebnissen ist etwa sinnvoll, da beispielsweise bereits heruntergeladene Datenpakete nicht nochmals übertragen werden müssen, sondern die Bestehenden erneut zum Einsatz gelangen.

3.2.6 Gliederung von Arbeitsabläufen

Sobald Arbeitsabläufe über mehr als einen Task verfügen, besteht die Notwendigkeit der Gliederung sowie der Definition von Abhängigkeiten in der Ausführung. Die Applikation muss dementsprechend in der Lage sein, solche Abhängigkeiten zu definieren und bei der Ausführung der einzelnen Arbeitsschritte zu berücksichtigen. Durch die automatisierte Überprüfung solcher Abhängigkeiten kann sichergestellt werden, dass Arbeitsabläufe einen definierten Start- und Endzustand haben und während der Ausführung nicht in Endlosschleifen geraten (etwa durch den gegenseitigen Aufruf von zwei Arbeitsaufgaben).

Eine weitere Gliederungsebene zur Definition von Parallelitäten ist notwendig, um bestehende optimal Infrastruktur auszunutzen. Die programmgesteuerte Erkennung von Parallelisierungsmöglichkeiten ist in der ersten Version nicht vorgesehen, und muss deshalb manuell bereits bei der Definition des Arbeitsablaufs festgelegt werden.

3.2.7 Parametrisierung von Arbeitsaufgaben

Viele Arbeitsabläufe sind durch Aufgaben gekennzeichnet, die durch Parameter gesteuert werden können. Die Applikation muss diese Möglichkeit unterstützen, indem einzelne Tasks mit Parametern versehen werden können, die bei der Ausführung berücksichtigt werden. Dadurch ergibt sich implizit die Vereinfachung der Definition von wiederkehrenden Tasks mit unterschiedlichen Parametern.

3.3 Architektur

Aus den im vorigen Abschnitt vorgestellten Anforderungen ergibt sich implizit eine logische Aufteilung der Applikation in drei verschiedene Hauptbereiche, die in weiterer Folge vorgestellt werden.

3.3.1 Anwendungsfälle

Die Anwendungsfälle (vgl. Abb. 3.1) unterteilen sich primär in das Erstellen und Überarbeiten von Workflows. Dabei wird entweder eine neue Beschreibung eines Arbeitsablaufs erstellt und anschließend ausgeführt, oder eine bereits vorhandene (und vorher ausgeführte) Version eines Workflows überarbeitet. Während der Ausführung eines Arbeitsablaufs wird zuerst die textuelle Beschreibung eingelesen, und bei jedem Schritt überprüft, ob es bereits wiederverwendbare Zwischenergebnisse gibt. Abschließend werden in der Subversion-Komponente alle neuen Ergebnisse abgelegt.

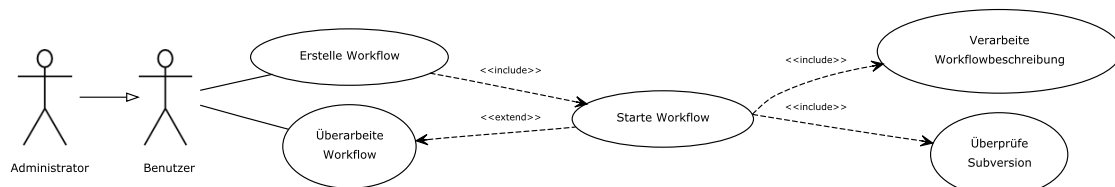


Abbildung 3.1: Anwendungsfalldiagramm

3.3.2 Verarbeitung von Workflowbeschreibungen

Einen wichtigen Aspekt übernimmt in vielen Programmen die Komponente zur Kommunikation mit dem Anwender. Innerhalb der hier vorgestellten Applikation erfolgt diese Interaktion primär über die Beschreibung von Arbeitsabläufen, deren Verarbeitung zum zentralen Bereich der Anwendung wird. Innerhalb dieser Komponente soll sowohl das Auslesen von bestehenden Workflows aus Dateien erfolgen, aber auch die Aufteilung in die verschiedenen Teilaufgaben eines Arbeitsablaufs durchgeführt werden.

3.3.3 Ausführungskomponente

In der Ausführungskomponente erfolgt hauptsächlich die Durchführung der in der Workflowsbeschreibung spezifizierten Aufgaben in der richtigen Reihenfolge. Dabei muss zuerst verifiziert werden, ob die Ausführung überhaupt möglich ist, indem eine Prüfung auf Beginn und Ende der Durchführungskette erfolgt. Anschließend muss die Ausführungsreihenfolge festgelegt werden, indem die Abhängigkeiten innerhalb der Aufgaben aufgelöst werden. Sobald die Sequenz der Aktivitäten vorliegt, kann mit der Ausführung des ersten Task begonnen werden. Abhängig vom Aktivitätstyp müssen in der Ausführungskomponente die richtigen Module gestartet werden. Darüber hinaus muss jeder Ausführungsschritt auf aufgetretene Fehler abgefragt werden und darauf entsprechend reagiert werden.

3.3.4 Subversion

Der dritte und letzte der Hauptaspekte der hier vorgestellten Applikation ist die Integration mit dem bestehenden Subversion-Repository. In dieser Komponente erfolgt gekapselt die gesamte Kommunikation mit dem Repository, wobei unterschiedliche Transaktionstypen (vgl. auch Abschnitt 2.1.5) verfügbar sein müssen:

Überprüfung Bei diesem Transaktionstyp wird ausschließlich lesend auf das Repository zugegriffen, um zu Überprüfen, ob ein Workflow in der aktuellen Form bereits im Repository angelegt wurde. Ist dies nicht der Fall, handelt es sich bei der nachfolgenden Transaktion um eine Neuerstellung („Checkin“) eines Workflows, ansonsten erfolgt ein „Update“.

Checkin Bei dieser Transaktion erfolgt die Speicherung eines neuen Workflows im zentralen Repository. Dabei ist darauf zu achten, dass alle wichtigen Parameter

wie etwa ein eindeutiger Name, bereits errechnete Resultate (sowohl in Dateien als auch ausschließlich in Textform) und die Strukturbeschreibung des Workflows inklusive aller verwendeter Parameter übertragen werden.

Update Dieser Transaktionstyp erfolgt, wenn eine neue Version eines bereits bestehenden Workflows im Repository aktualisiert wird. Dabei wird von der Möglichkeit von Subversion Gebrauch gemacht, neue Revisionen anzulegen, sodass nur geänderte Daten in das zentrale Verzeichnis gespeichert werden müssen.

Checkout Um bereits existierende Arbeitsabläufe aus dem Repository auszulesen, muss es den Transaktionstyp „Checkout“ geben. Hier muss nochmals unterschieden werden zwischen der Möglichkeit, einen kompletten Workflow inklusive bereits errechneter Ergebnisse aus dem Verzeichnis auszulesen, und der Möglichkeit, nur Teile (etwa ein einziges Zwischenergebnis) eines Arbeitsablaufs weiterzuverwenden.

3.4 Zusammenfassung

In diesem Kapitel wurde die Architektur der Anwendung vorgestellt. Dabei wurde in den ersten Abschnitten auf das Systemumfeld und die Anforderungen eingegangen, die der Applikation zugrunde liegen. Diese Rahmenbedingungen wurden dahingehend kombiniert, dass sie im anschließenden Abschnitt zu Anwendungsfällen zusammengefasst wurden. Abschließend wurden drei zentrale Punkte - die Workflowbeschreibung und deren Verarbeitung, die Ausführungskomponente sowie der Zugriff auf Subversion - vorgestellt.

Kapitel 4

Implementierung

In diesem Kapitel wird die konkrete Umsetzung des Programms beschrieben. Dabei wird zuerst näher auf die Implementierungsentscheidungen eingegangen, um danach einen Überblick über die eingesetzten Software-Bibliotheken zu geben. Der Hauptteil des Kapitels widmet sich der Beschreibung der Anwenderschnittstelle und kann so gleichzeitig als Referenz für die Erstellung von Arbeitsablaufsbeschreibungen herangezogen werden. Abschließend erfolgt eine Übersicht über den Zugriff auf das zentrale Repository und mögliche Erweiterungen des Programmumfangs.

4.1 Implementierungsentscheidungen

In diesem Abschnitt werden einige grundlegende Entscheidungen erläutert, die im Vorfeld der Implementierung aufgrund der vorgegebenen Anforderungen getroffen wurden. Darüber hinaus werden zur Anwendung gelangende Programmierparadigmen erklärt und ein kleiner Einblick in die Entwicklungsumgebung gegeben.

4.1.1 Paradigmen

Für das Verständnis des Aufbaus der Applikation ist das wichtigste zum Einsatz kommende Paradigma das Prinzip der *Dependency Injection*. Dabei handelt es sich um eine Anwendung des *Inversion of Control* Musters mit dem Ziel, Abhängigkeiten zwischen Komponenten (etwa Klassen in einem objektorientierten System) zu verringern. Dieser Ansatz wurde 2004 von Martin Fowler in einem Online-Artikel ([Fowler 04]) detailliert

beschrieben, und bildet die Grundlage für viele heute verfügbare Applikationsserver, Frameworks und andere Container.

In der vorliegenden Applikation wird dieses überaus hilfreiche Konzept angewandt, um die Kontrolle über die Art der in Arbeitsabläufen vorkommenden Aktivitäten an das zugrundeliegende Spring Framework (siehe Abschnitt 4.2.1) zu delegieren. Darüber hinaus ermöglicht es die einfache Erweiterbarkeit der Grundfunktionalität, ohne dass dafür bestehender Code zu adaptieren ist. Um einen solchen Ausbau zu realisieren, ist es notwendig, den neuen Task zu implementieren und anschließend die neue Klasse dem Container bekanntzugeben. Dafür trägt man das erstellte Objekt in die bereits vorhandene Liste der Aktivitätstypen ein und kann es sofort in einem Arbeitsablauf verwenden. Ein Beispiel für die bestehenden Aktivitätstypen bietet Listing 9, eine nähere Beschreibung des Dateiinhalts findet sich im Abschnitt 4.4.3.

Natürlich ist Dependency Injection nicht das einzige zum Einsatz kommende Paradigma. Als natives Java-Programm ist die Applikation hierarchisch in Klassen unterteilt, die voneinander abgeleitet werden oder gekapselte Informationen nur über Schnittstellen bereitstellen können. Dies entspricht damit dem Konzept der *Objektorientierung* (vgl. [Ullenboom 03]).

4.1.2 Entwicklungsumgebung

Die hier vorgestellte Applikation „*WorkflowExecutor*“ wurde in Java realisiert, was die Entscheidung für eine frei verfügbare Entwicklungsumgebung deutlich erleichtert hat. Obwohl es am Markt einige unterschiedliche freie Editoren¹ gibt, hat sich in den letzten Jahren *Eclipse* [Foundation 10c] als eine Art Standard etabliert. Die Gründe dafür sind vielfältig, einige davon sind folgende:

Plattformunabhängigkeit Eclipse ist für unterschiedliche Betriebssysteme verfügbar und bietet dadurch plattformübergreifend eine einheitliche Oberfläche.

Grundfunktionalität Eclipse beherrscht die wichtigsten Arbeitserleichterungen für Programmierer wie etwa Syntaxhervorhebung, Debugging oder Wortvervollständigung.

Erweiterbarkeit Die ohnehin schon beträchtlichen Grundfunktionen sind durch

¹vgl. auch <http://www.thefreecountry.com/programming/javaide.shtml>, abgerufen am 13. August 2010

verschiedenste Module erweiterbar, was etwa den Zugriff auf ein Subversion-Repository oder die Integration von Apache Ant (siehe Kapitel 4.2.8) als Werkzeug zur Interaktion mit dem Java-Compiler ermöglicht.

Während der Entwicklung wurde die aktuelle Version 3.5 („Galileo“) unter dem Betriebssystem Mac OS X² eingesetzt (vgl. Abb. 4.1).

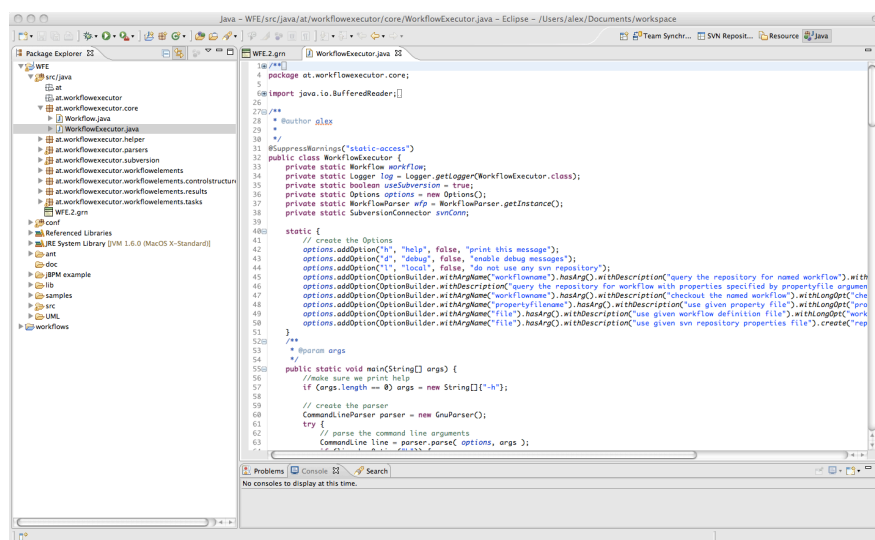


Abbildung 4.1: Screenshot der Entwicklungsumgebung „Eclipse“

4.2 Eingesetzte Bibliotheken

In diesem Abschnitt werden die unterschiedlichen Software-Bibliotheken, die zur Implementierung eingesetzt wurden, kurz vorgestellt. Für detaillierte Informationen wird jeweils zumindest ein Verweis zur Webseite des Herstellers zur Verfügung gestellt.

4.2.1 Spring Framework

Spring ist ein seit mehreren Jahren sehr populäres Open-Source-Framework zur Erstellung von Java-Anwendungen. Es dient im aktuellen Kontext primär zur Bereitstellung des bereits oben angesprochenen *Inversion of Control*-Paradigmas (siehe Kapitel 4.1.1). Durch den Einsatz dieser Technik können auf relativ einfache Art und Weise (durch Menschen lesbare) Arbeitsablaufbeschreibungen erstellt werden, die in weiterer Folge während der Programmausführung durch das Framework in für den Computer

²<http://www.apple.com/macosx/>, abgerufen am 13. August 2010

bearbeitbare Objekte verwandelt werden. Das Spring Framework stellt dafür einen *ApplicationContext* zur Verfügung, mit dessen Hilfe zur Ausführungszeit die benötigten Objekte erzeugt werden können. Dieser Vorgang kann (vereinfacht) so beschrieben werden:

Listing 2 Vereinfachtes Beispiel zur Dependency Injection mit Spring

```
1 ApplicationContext ctx = new
    ClassPathXmlApplicationContext("taskbeans.xml");
2 while (arbeitsablauf.hasNext()) {
3     WorkflowElement task = (WorkflowElement)ctx.getBean(it.next());
4     task.execute();
5 }
```

Zuerst wird ein neuer *ApplicationContext* zur Interaktion mit dem Spring Framework instantiiert (Zeile 1). Anschließend wird über alle Aktivitäten in einem Arbeitsablauf iteriert (Zeile 2) und für jede Aktivität vom Framework die entsprechende Klasse als Objekt erstellt (Zeile 3). Der Trick dabei ist, dass das *WorkflowElement* nicht „weiß“ (und auch nicht wissen muss), von welchem konkreten Typ es ist, sondern das Spring-Framework die Aufgabe übernimmt, aus dem Arbeitsablauf die richtige Aktivität zu laden. In weiterer Folge (Zeile 4) kann mit diesem Objekt wie mit jeder anderen „normalen“ Java-Klasse verfahren werden. Zur weiteren Vertiefung dieser speziellen Funktionalität des Spring-Frameworks sei auf [Walls 07] verwiesen, für Herstellerinformationen kann die Homepage [www.springsource.org 10] herangezogen werden.

4.2.2 SVNkit

Subversion³ ist ein quelloffenes Paket zur Versionskontrolle und wurde entwickelt, um einige Schwächen des weitverbreiteten *Concurrent Versions System*⁴ (*CVS*) zu beheben. Das Framework *SVNkit*[Software 09] ist eine in der Programmiersprache Java geschriebene Schnittstelle zur direkten Interaktion mit so einem zentralen Subversion-Softwareverzeichnis, ohne auf externe Programme zurückgreifen zu müssen. Dabei unterstützt das Programmpaket einen Großteil der Möglichkeiten von Subversion, darunter etwa die Integration von sog. *working copies* sowie die Möglichkeit, auf unterschiedliche Typen von Repositories zuzugreifen [Yan 08].

³<http://subversion.apache.org/>, abgerufen am 13. August 2010

⁴<http://www.nongnu.org/cvs/>, abgerufen am 13. August 2010

4.2.3 Apache Commons CLI

Apache Commons CLI (Command Line Interface) ist eine Bibliothek zum Bearbeiten von Kommandozeilenparametern und bietet verschiedene Methoden, um diese zu lesen, auszuwerten und auch auszugeben. Durch die Verwendung dieser Bibliothek ist es möglich, mit geringem Aufwand auch komplexere Eingabemöglichkeiten (etwa optionale Parameter, Unterstützung von Kurz- und Langschreibweise, etc.) zu erstellen. Ein Beispiel für eine Apache Commons CLI formatierte Ausgabe ist in Abb. 4.2 zu sehen. Weitere Informationen über die Möglichkeiten der Bibliothek sind in [O'Brien 05] zu finden, das gesamte Paket kann in der jeweils aktuellen Version unter [Foundation 10b] heruntergeladen werden.

```
C:\>java -jar workflowexecutor.jar
usage: WorkflowExecutor
  -c,--checkout <workflowname>      checkout the named workflow
  -d,--debug                          enable debug messages
  -h,--help                          print this message
  -l,--local                          do not use any svn repository
  -p,--properties <propertyfilename> use given property file
  -q,--query <workflowname>          query the repository for named workflow
  -qp,--queryproperties               query the repository for workflow with properties
                                      specified by propertyfile argument
  -repprops <file>                   use given svn repository properties file
  -w,--workflow <file>               use given workflow definition file
```

Abbildung 4.2: Beispielausgabe mit Apache Commons CLI formatiert.

4.2.4 Apache Commons Exec

Das Paket Apache Commons Exec ([Foundation 09b]) dient dazu, plattformübergreifend von Java-Applikationen aus externe Programme auszuführen. Durch Verwendung dieser Programmbibliothek ist es ohne größeren Aufwand möglich, auch komplexe Programmaufrufe (etwa mit verschiedenen Parametern oder unter Berücksichtigung von Umgebungsvariablen) zu realisieren. Darüber hinaus können unterschiedliche Rückgabewerte ausgewertet werden oder auch die maximale Ausführungsdauer im vorhinein festgelegt werden.

Im aktuellen Kontext ist die Verwendung dieser Bibliothek äußerst hilfreich, da damit das Ausführen von externen Aktivitäten innerhalb eines Arbeitsablaufs ermöglicht wird, ohne diese jeweils mühsam in die Programmiersprache Java übersetzen zu müssen (siehe Kapitel 4.4.1).

4.2.5 JDOM

Unabhängig von eingesetzten Programmiersprachen gibt es zumindest zwei Methoden, um ein XML-Dokument auszulesen und zu bearbeiten: Die erste Variante ist, die Datei sequentiell einzulesen und zu bearbeiten, die andere Methode ist, die gesamte Struktur in den Speicher zu laden und anschließend zu bearbeiten. Beide Alternativen haben ihre Vor- und Nachteile, etwa Speicherverbrauch oder wahlfreier Zugriff. Für die Programmiersprache Java werden Schnittstellen für jede dieser Varianten bereitgestellt, und jdom [jdom.org 09] ist ein Repräsentant der letzteren Methode. Im Zuge der Bearbeitung einer XML-Datei wird zuerst die gesamte Struktur in den Speicher geladen, anschließend kann - und das ist ein großer Vorteil von jdom - mittels Standard-Java darauf zugegriffen werden. Da die hier zum Einsatz gelangenden Arbeitsablaufbeschreibungen nicht außergewöhnlich groß werden, bleibt auch die Speicherbelastung in einem vertretbaren Rahmen. Eine sehr gute Zusammenfassung der XML-Bearbeitung in Java bietet [McLaughlin 06].

4.2.6 log4j

In den meisten Applikationen besteht die Notwendigkeit, dem Anwender zu verschiedenen Zeitpunkten Statusinformationen zu übermitteln, sei es durch Ausgabe auf der Kommandozeile oder durch extensives Protokollieren in eine Datenbank oder eine Datei. Das von der Apache Foundation veröffentlichte Framework *log4j* ([Foundation 09a]) bietet eine standardisierte und ausgereifte Möglichkeit, alle diese Aufgaben einfach zu erledigen. Die Einbindung des Pakets in eine Applikation erfolgt durch Instanziierung eines sog. *loggers*, mit dessen Hilfe in weiterer Folge Meldungen in verschiedenen „Fehlergrenzstufen“ (beginnend bei *debug* zur Fehlersuche für Entwickler über *info* zur Ausgabe von Informationen bis hin zur Stufe *fatal* zur Protokollierung von schwerwiegenden Fehlern) bereitgestellt werden. Die genaue Konfiguration erfolgt für gewöhnlich in einer Datei (siehe dazu auch Kapitel 4.4.3) und kann bei Einsatz in einem Webcontainer auch zur Laufzeit geändert werden. Die konkreten Einstellungen beim Auslieferungszustand für die hier vorgestellte Applikation sind in Listing 3 zu sehen. Dabei werden zwei verschiedene Ziele für Fehlermeldungen (diese erfordern unter Umständen sofortige Aufmerksamkeit des Benutzers und gelangen deshalb auf die Kommandozeile) und Debug-Meldungen (es wird für jede Ausführung eines Arbeitsablaufs eine Protokolldatei erstellt) definiert. Für weitere Konfigurationsmöglichkeiten sei auf [Gulcu 03] verwiesen.

Listing 3 Konfigurationsbeispiel für log4j.

```
1 log4j.rootLogger=DEBUG, CONSOLE, FILE
2
3 # unwichtige Ausgaben von spring verhindern
4 log4j.logger.org.springframework=WARN
5
6 # CONSOLE gibt Fehlermeldungen auf die Kommandozeile aus
7 log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
8 log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
9 log4j.appender.CONSOLE.threshold=INFO
10 log4j.appender.CONSOLE.layout.ConversionPattern=%d{ISO8601} %-5p
    %c{2}: %m%n
11
12 #FILE protokolliert in Dateien
13 log4j.appender.FILE=org.apache.log4j.DailyRollingFileAppender
14 log4j.appender.FILE.threshold=ALL
15 log4j.appender.FILE.datePattern='.'yyyy-MM-dd_HH-mm
16 log4j.appender.FILE.file=logs/WorkflowExecutor.log
17 log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
18 log4j.appender.FILE.layout.ConversionPattern=%d{ISO8601} %-5p [%t]
    %c: %m%n
```

4.2.7 HyperSQL

HyperSQL ([hsq] Development Group 10)) ist kein grundlegender Bestandteil der Applikation, wird aber aufgrund der einfachen Integration exemplarisch für die Demonstration des direkten Zugriffs auf eine SQL-Datenbank eingesetzt. HyperSQL zeichnet sich unter anderem durch freie Verfügbarkeit, die Unterstützung verschiedener Betriebsmodi (etwa Tabellen, die nur im Arbeitsspeicher existieren, aber auch festplattenbasierte Tabellen) und die durch den Einsatz in verschiedenen Open Source Projekten (etwa OpenOffice 3.0⁵, JBoss⁶ und Mathematica⁷) erlangte Stabilität aus.

4.2.8 Apache Ant

Apache Ant [Foundation 10a] ist ein Werkzeug zur Automatisierung von Aufgaben des Software-Konfigurationsmanagements und bildet genauso wie HyperSQL keinen integralen Bestandteil des Programms, wurde aber extensiv während der Entwicklung eingesetzt. Dabei wurde vor allem auf die Möglichkeit des Erstellens von sog. *jarfiles* zurückgegriffen, um für Endanwender ein einfach auszuführendes Programmpaket zu

⁵<http://www.openoffice.org/>, abgerufen am 13. August 2010

⁶<http://www.jboss.com/>, abgerufen am 13. August 2010

⁷<http://www.wolfram.com/>, abgerufen am 13. August 2010

erstellen.

Ant wird wie viele andere Werkzeuge im Java-Umfeld mit Hilfe von xml-Dateien konfiguriert. Diese bilden im aktuellen Anwendungsfall einen gewissen Arbeitsablauf ab, der mittels sog. *targets* in einzelne Aktivitäten unterteilt wird. Diese Aktivitäten können untereinander abhängig sein und werden beim Ausführen in der richtigen Reihenfolge exekutiert.

Ant ist durch ein offenes Schnittstellensystem mit eigenen Aktivitäten erweiterbar, und kann so perfekt an die eigenen Bedürfnisse angepasst werden. In Listing 4 ist ein Auszug aus dem buildfile der Anwendung zu sehen, der aus dem Quelltext und den zugehörigen externen Programmbibliotheken eine „ausführbare“ jar-Datei erzeugt. Dabei wird automatisch der notwendige Classpath erzeugt und in die Datei *manifest.mf* geschrieben.

Listing 4 Auszug aus dem Apache Ant buildfile.

```
1 <target name="jar" depends="compile" description="generate executable
   jarfile">
2   <!--remove the full path from entries -->
3   <pathconvert property="classpath" pathsep=" " refid="extLibs">
4     <flattenmapper />
5   </pathconvert>
6
7   <!-- copy all external libraries to jar-directory-->
8   <copy todir="${target.jar}" flatten="true">
9     <path refid="extLibs" />
10  </copy>
11
12  <!-- include config files in the jar-->
13  <copy todir="${target.jar}" flatten="true">
14    <fileset refid="configFiles"/>
15  </copy>
16
17  <!-- build jarfile -->
18  <jar destfile="${target.jar}/workflowexecutor.jar">
19    <fileset dir="${target.java}">
20      <include name="**/*.class"/>
21    </fileset>
22    <manifest>
23      <attribute name="Main-Class"
24        value="at.workflowexecutor.core.WorkflowExecutor" />
25      <attribute name="Class-Path" value=". ${classpath}" />
26    </manifest>
27  </jar>
28 </target>
```

4.3 Programmstruktur

Dieser Abschnitt gibt einen Überblick über den internen Aufbau der Applikation und die verschiedenen Komponenten, die darin zum Einsatz kommen. Dabei wird zuerst auf die Paketstruktur eingegangen, und anschließend eine Übersicht über die Klassen und deren Zusammenhänge gegeben. Das Ende dieses Abschnitt bildet die detaillierte Beschreibung der Implementierung dreier wichtiger Klassen der Applikation, *Workflow*, *WorkflowParser* und *SubversionConnector*.

4.3.1 Pakete

Als ein in der Programmiersprache Java implementiertes Projekt und entsprechend der geplanten Architektur ist das Programm hierarchisch in fünf Paketen organisiert, die sich teilweise wiederum in Unterpakete gliedern (siehe auch Abb. 4.3):

core Dieses Paket beinhaltet die wichtigsten grundlegenden Klassen wie den Einstiegspunkt und die Repräsentation eines Arbeitsablaufs.

helper In diesem Paket werden Hilfsklassen abgelegt.

parsers Dieses Paket dient dazu, die verschiedenen Klassen zum Einlesen von Datenstrukturen zu gruppieren.

subversion Hier werden alle zur Kommunikation mit dem zentralen Subversion-Repository notwendigen Dateien abgelegt.

workflowelements Dieses Paket dient als Container für alle unterschiedlichen Klassen, aus denen ein Arbeitsablauf zusammengesetzt werden kann und unterteilt sich wiederum in die Pakete *controlstructures*, *results* und *tasks*.

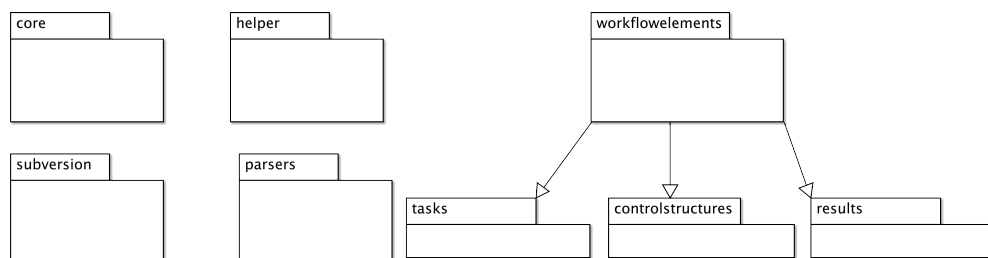


Abbildung 4.3: Darstellung der Paketstruktur.

4.3.2 Klassenübersicht

Innerhalb der Paketstruktur ist die Anwendung in Klassen unterteilt, die die konkrete Umsetzung der Programmlogik beinhalten. Abbildung 4.4 bietet eine Übersicht über diese Klassen.

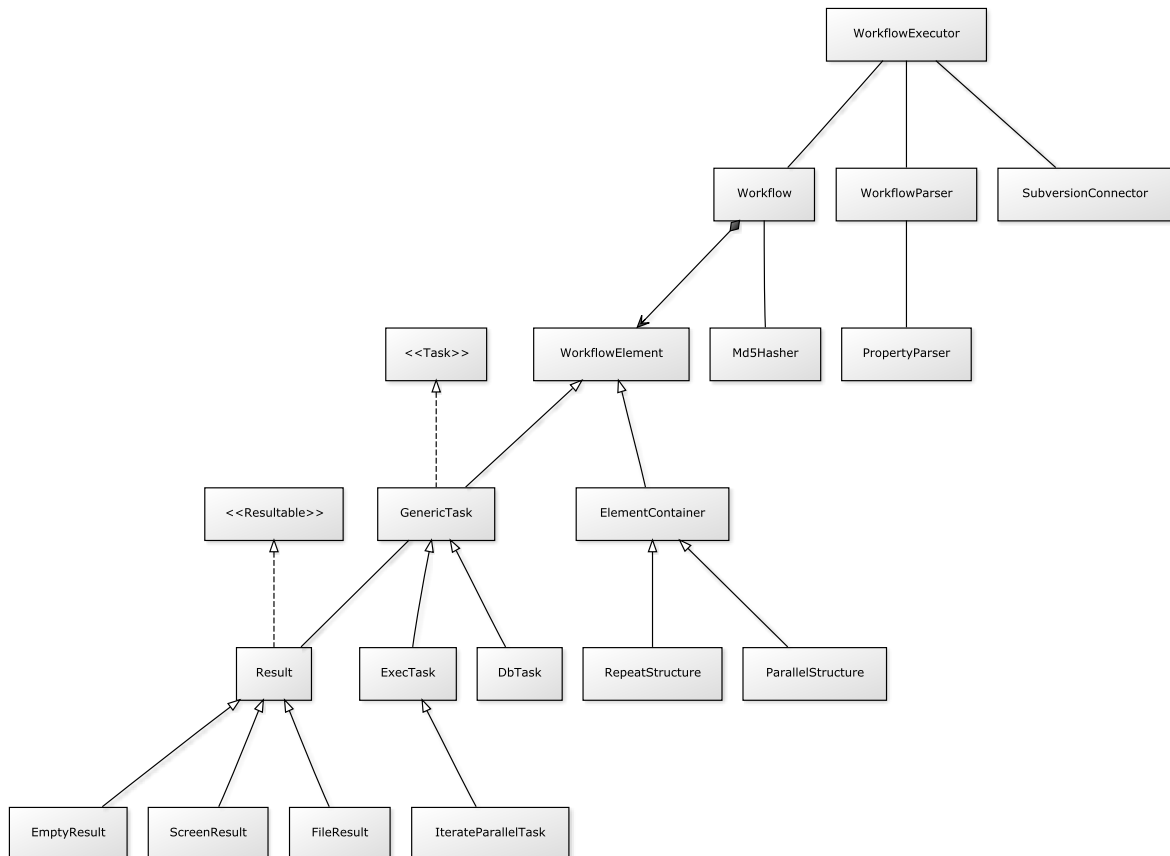


Abbildung 4.4: Klassendiagramm.

Als Haupteinstiegspunkt für die Programmausführung dient dabei die Klasse mit dem Namen *WorkflowExecutor* aus dem Paket *core*. Hier werden die Kommandozeilenparameter (vgl. Abschnitt 4.4.2) ausgewertet und die entsprechenden weiteren Programmteile aufgerufen. Abhängig von den übergebenen Parametern wird im normalen Ablauf eine bestehende Workflowbeschreibung mit Hilfe der Klasse *WorkflowParser* aus dem Paket *parsers* eingelesen und in die verschiedenen Typen (vgl. Abschnitt 4.4) zerlegt. Sobald die Ausführung eines Tasks durchgeführt wurde, wird dessen Ergebnis in der jeweiligen Klasse (abhängig von der Strukturbeschreibung des Workflows, zum aktuellen Entwicklungsstand entweder vom Typ *FileResult* oder *ScreenResult* aus dem Paket *workflovelements.results*) gespeichert. Sofern der Zugang zu einem Subversion-Repository gewünscht und auch konfiguriert ist, wird unter Verwendung des Klasse

SubversionConnector aus dem Paket *subversion* der Zugriff durchgeführt. Abb. 4.5 bietet eine schematische Darstellung dieses Normalablaufs.

4.3.3 Detailbeschreibung der Klasse WorkflowParser

Die Klasse *WorkflowParser* kapselt alle Funktionen zum Einlesen eines Workflows aus einer Datei. Als wichtigste Klassenvariable kommen ein Objekt vom Typ *org.springframework.context.ApplicationContext* (vgl. Abs. 4.2.1) und ein Objekt vom Typ *org.jdom.input.SAXBuilder* (vgl. Abs. 4.2.5) zum Einsatz.

Beim Aufruf des Konstruktors der Klasse mit dem Parameter *xmlFileName* wird zuerst die XML-Struktur aus der Datei geladen. Dabei kommt die Instanz des SAXBuilder zum Einsatz, mit dessen Hilfe das gesamte Dokument eingelesen wird. Anschließend wird ausgehend vom Wurzelement der XML-Repräsentation des Workflows die Liste der einzelnen Arbeitsschritte erstellt.

Bei diesem Schritt wird von der im Abschnitt 4.2.1 vorgestellten Möglichkeit der *Dependency Injection* Gebrauch gemacht, sodass jeder einzelne Task des Arbeitsablaufs korrekt initialisiert wird. Für jedes Element werden schrittweise die einzelnen angegebenen Variablen befüllt, wobei es keinen Unterschied macht, ob die Auszeichnung in der darunterliegenden XML-Struktur als Attribut oder als Kindelement ausgeführt ist. Bei der Einarbeitung des verwendeten Ergebnistyps wird wiederum von der Möglichkeit der *Dependency Injection* Gebrauch gemacht.

Sobald alle Elemente des Arbeitsablaufs aus der Datei eingelesen und in Java-Objekte umgewandelt wurden, liegt der Workflow vollständig vor, und wird als Objekt vom Typ *Workflow* zurückgegeben.

4.3.4 Detailbeschreibung der Klasse Workflow

In dieser Klasse ist die konkrete Ausprägung eines Workflows implementiert. Die wichtigsten Elemente eines Arbeitsablaufs sind die unterschiedlichen *WorkflowElements*, die in einer Liste abgespeichert werden. Beim Aufruf des Konstruktors der Klasse mit dem Namen des Workflows und der Taskliste als Übergabeparameter wird zuerst eine Adjazenzmatrix der einzelnen Aufgaben erstellt. Dafür wird über alle Elemente der Taskliste iteriert und die Vorgänger-Nachfolger-Beziehung in einem zweidimensionalen Array gespeichert.

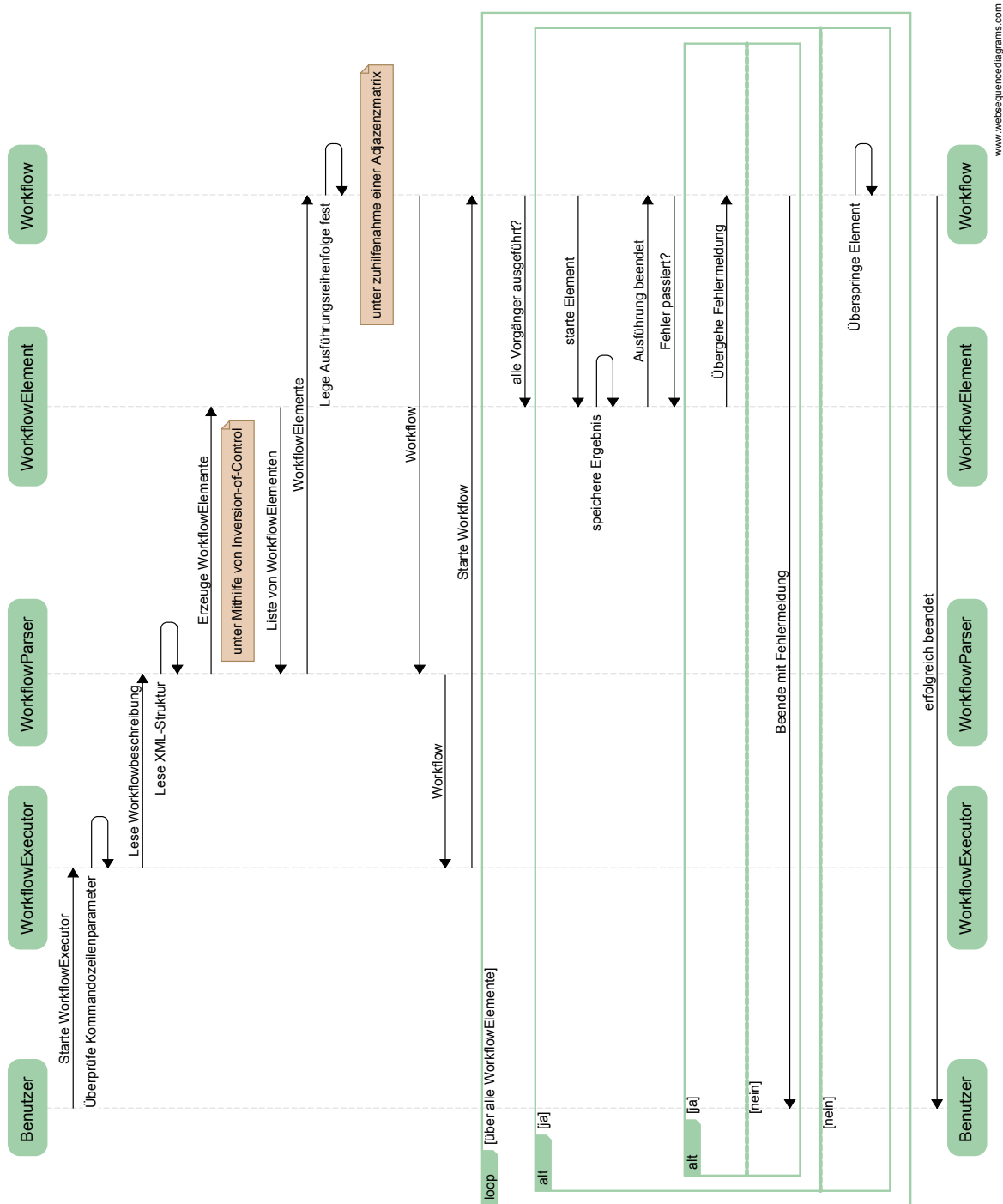


Abbildung 4.5: Sequenzdiagramm.

Als nächster Schritt erfolgt die Reihung der einzelnen Tasks aufgrund ihrer Position in der im vorherigen Schritt erstellten Adjazenzmatrix. Dabei wird ausgehend von den Knoten ohne Nachfolgern („Blätter“, Endknoten) schrittweise Richtung Wurzelement des Workflows vorgegangen. Auf diese Weise entsteht eine sortierte Liste, die die spätere Ablaufreihenfolge festlegt.

Sobald eine neue Instanz der Klasse *Workflow* angelegt wurde, kann der Arbeitsablauf durch Aufruf der öffentlichen Methode *execute()* gestartet werden. Dabei wird eine neue Threadgruppe angelegt, und anschließend über die sortierte Liste der Elemente iteriert. Für jedes Element, dessen Vorgänger bereits erfolgreich ausgeführt wurden, wird innerhalb der Hauptthreadgruppe ein neuer Thread gestartet. Sobald die Ausführung des einzelnen Elements beendet ist, wird überprüft, ob der Rückgabewert einen Fehlerzustand kennzeichnet bzw. ob die gesamte Ausführung unterbrochen werden muss. Falls diese Prüfung erfolgreich beendet wurde, wird der Status des Elements auf *ausgeführt* gesetzt und mit dem nächsten Element weitergemacht. Wenn das letzte Element des Workflows erfolgreich ausgeführt wurde, werden die vollständigen Ergebnisse mit Hilfe der Klasse *SubversionConnector* (vgl. Abs. 4.3.5) im zentralen Software-Repository gespeichert und die Ausführung beendet.

4.3.5 Detailbeschreibung der Klasse *SubversionConnector*

Die Klasse *SubversionConnector* ist für alle Zugriffe auf das zentrale Subversion-Repository zuständig. Beim Aufruf des Konstruktors werden aus einer Konfigurationsdatei (vgl. Abs. 4.4.3) die notwendigen Einstellungen für den Zugriff auf das Verzeichnis geladen. Anschließend wird basierend auf diesen Einstellungen der Zugang auf das Repository validiert und die Membervariablen befüllt. Für die weiteren Zugriffe werden verschiedene Methoden zur Verfügung gestellt, von denen die wichtigsten in Folge vorgestellt werden sollen:

public boolean isUpdate(*Workflow workflow*) Mit Hilfe dieser Methode kann festgestellt werden, ob bereits eine idente Version des übergebenen Workflows im Repository abgelegt wurde. Dafür werden zuerst die verschiedenen Attribute des Workflows extrahiert, und anschließend das gesamte Verzeichnis auf diese Properties überprüft. Dabei wird ausgehend von einem Wurzelverzeichnis im Repository jeder bereits gespeicherte Workflow überprüft, ob er idente Eigenschaften hat. Sobald ein Workflow gefunden wurde, wird der Wert *true* zurückgegeben, ansonsten der Wert *false*.

public Workflow checkoutWorkflow (String workflowName) Diese Methode dient dazu, die aktuellste Version eines bereits im Repository existierenden Workflows auszulesen. Dabei wird zuerst - ähnlich wie bei der Überprüfung, ob es sich um ein Update eines Arbeitsablaufs handelt - das Verzeichnis nach dem entsprechenden Workflow durchsucht. Anschließend wird im Ausführungsverzeichnis ein neues Unterverzeichnis angelegt, und der entsprechende Workflow mit Hilfe der Methode *doExport* aus dem Paket SVNkit (vgl. Abs. 4.2.2) ausgelesen. Abschließend wird das neue Verzeichnis als Arbeitsverzeichnis für den Workflow gesetzt und der Arbeitsablauf retourniert.

public boolean checkinWorkflow (Workflow workflow) Diese Methode ist die Umkehrung der vorhergehenden Methode, hierbei wird eine komplett neue Version eines Workflows im Repository angelegt. Die Vorgehensweise dabei ist, zuerst alle Informationen über den Workflow zu sammeln, diese in ein Objekt vom Typ *SVNProperties* zu packen, und abschließend den gesamten Arbeitsablauf inklusive dessen Ergebnisse in das Verzeichnis zu speichern. Mit diesem Verfahren ist sichergestellt, dass alle wichtigen Attribute des Workflows gespeichert werden und dieser auch wieder durch die Suchmethoden auffindbar ist. Wenn während der Abarbeitung festgestellt wird, dass bereits ein Workflow mit identen Parametern existiert, wird automatisch die Methode *commitWorkflow* aufgerufen.

public boolean commitWorkflow(W Workflow workflow, boolean isUpgrade)
Mit Hilfe dieser Methode wird eine neue Version eines bereits existierenden Workflows im Repository abgelegt. Um nicht für jede Version den gesamten Datenbestand eines Arbeitsablaufs speichern zu müssen, wird durch das Paket SVNkit (vgl. Abs. 4.2.2) ein Delta zur letzten Version berechnet und nur die geänderten Daten übertragen. Dabei wird über alle Dateien des Workflows iteriert, und für jede festgestellt, ob sie sich von der vorhergehenden Version unterscheidet. Abschließend wird die neue Version des Workflows im Repository mit allen notwendigen Parametern versehen, sodass die Suchmethoden in Zukunft auf diese Version zugreifen.

4.4 Anwenderschnittstelle

Dieser Abschnitt beschreibt die Schnittstelle zur Kommunikation des Anwenders mit dem System, die aus drei verschiedenen Komponenten besteht. So muss zumindest

einmalig die grundsätzliche Konfiguration des Systems erfolgen (siehe Kapitel 4.4.3), danach ist es notwendig, den gewünschten Arbeitsablauf in ein standardisiertes Format zu bringen (siehe Kapitel 4.4.1) und die Ausführung des Workflows über diverse Kommandozeilenparameter (siehe Kapitel 4.4.2) zu starten.

4.4.1 Struktur von Arbeitsablaufsbeschreibungen

Das Format einer Arbeitsablaufbeschreibung entspricht einer einfachen XML-Struktur und ist problemlos mit einem beliebigen Texteditor zu erstellen bzw. zu bearbeiten. Der grundsätzliche Aufbau des Dokuments ist bei jedem Workflow ident: Es besteht aus mindestens einem Wurzelement des Typs „process“, das wiederum mindestens ein Element des Typs „task“ beinhaltet. Zum einfacheren Verständnis dient Listing 5 zur Darstellung eines exemplarischen gültigen Arbeitsablaufs:

Listing 5 Exemplarischer Arbeitsablauf

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <process name="simpleprocess">
3   <task name="StartNode">
4     <description>Einstiegsknoten</description>
5     <type>execTask</type>
6     <result>
7       <type>emptyResult</type>
8     </result>
9     <fork>echo "Hello World"</fork>
10    <antecessor></antecessor>
11  </task>
12 </process>
```

Zeile 1 des Arbeitsablaufs deklariert die nachfolgenden Zeilen als XML-Dokument mit der Zeichenkodierung UTF-8 und ist kein zwingender Bestandteil der Arbeitsablaufbeschreibung. Zeile 2 beschreibt das Wurzelement vom Typ „process“ mit dem Namen „simpleprocess“. Als erstes und einziges Kindelement dieses Arbeitsablaufs wird in den Zeilen 3 bis 11 die Aktivität mit dem Namen „StartNode“ beschrieben. Bei der Deklaration von Aktivitäten kann auf ein allgemeines Set von Parametern zurückgegriffen werden, von denen einige zwingend vorhanden sein müssen, andere optional eingesetzt werden können:

Parameter	Beschreibung	Wertebereich	Kardinalität
name	dient der eindeutigen Identifikation einer Aktivität	Freitext	1
type	zur eindeutigen Kennzeichnung der Art der Aktivität	„execTask“, „dbTask“, „iterateParallelTask“	1
description	Beschreibung der Aktivität	Freitext	0 .. 1
antecessor	alle Nachfolger dieser Aktivität	kommaseparierte Liste von Aktivitäts-Namen	0 .. 1
onErrorResumeWithElement	Name der Nachfolgeraktivität im Fehlerfall	Aktivitäts-Name	0 .. 1
resumeOnError	Abbrechen der Ausführung im Fehlerfall	true, false	0 .. 1
timeout	maximale Ausführungszeitspanne in Millisekunden	Ganzzahl	0 .. 1
property	Kennzeichnung („tag“) der Aktivität.	Name=Wert	0 .. n

Tabelle 4.1: Auflistung möglicher genereller Aktivitäts-Parameter

Die drei angeführten unterschiedlichen Aktivitäts-Typen „execTask“, „dbTask“ und „iterateParallelTask“ erfüllen unterschiedliche Funktionen, die nachfolgend kurz beschrieben werden:

execTask Mittels dieses Typs kann ein beliebiges externes Programm aufgerufen werden. Für diesen Typ gibt es die folgenden speziellen Parameter:

Parameter	Beschreibung	Wertebereich	Kardinalität
fork	die auszuführende Prozedur	Freitext	1
exitValues	Liste von gültigen Rückgabewerten der aufgerufenen Prozedur	kommaseparierte Liste von Ganzzahlen	0 .. 1
environment Variable	Umgebungsvariable der Aktivität.	Name=Wert	0 .. n

Tabelle 4.2: Auflistung möglicher Aktivitäts-Parameter für den Task execTask

dbTask Dieser Aktivitätstyp erlaubt das direkte Ausführen von SQL-Abfragen (vgl. Abschnitt 4.4.3 für die Konfiguration des Datenbankzugriffs). Für diesen Typ gibt es die folgenden speziellen Parameter:

Parameter	Beschreibung	Wertebereich	Kardinalität
query	die auszuführende SQL-Abfrage	Freitext	1
dbName	der Name der abzufragenden Datenbank	Freitext	1

Tabelle 4.3: Auflistung möglicher Aktivitäts-Parameter für den Task dbTask

iterateParallelTask Mittels dieses Typs kann eine Sequenz von ExecTasks ausgeführt werden. Dabei wird der aufzurufenden Kommandozeile eine Folge von Werten übergeben. Diese Parameter werden im Feld „parameterValues“ angegeben. Ein Beispiel dafür findet sich in Listing 6.

Parameter	Beschreibung	Wertebereich	Kardinalität
fork	die auszuführende Prozedur	Freitext	1
exitValues	Liste von gültigen Rückgabewerten der aufgerufenen Prozedur	kommaseparierte Liste von Ganzzahlen	0 .. 1
environment Variable	Umgebungsvariable der Aktivität.	Name=Wert	0 .. n
parameterValues	Kommandozeilenparameter	Durch Semikolon getrennte Freitext-Liste bzw. ein Wertebereich der Form a : b	1

Tabelle 4.4: Auflistung möglicher Aktivitäts-Parameter für die Aktivität iterateParallelTask

Diese vorhandenen Aktivitäten werden noch durch zwei „Containeraktivitäten“ erweitert, die einige spezielle Funktionen hinsichtlich des Ablaufs bieten. Die erste Containeraktivität mit dem Namen „repeat“ bietet die Möglichkeit, eine beliebige Anzahl von Aktivitäten mehrmals hintereinander ausführen zu lassen. Die Struktur ist im Listing 7 dargestellt. Die zweite Containeraktivität ergänzt die bisherigen Funktionen um die Möglichkeit, mehrere Tasks parallel ablaufen zu lassen. Ein Beispiel hierzu ist im Listing 8 zu sehen.

Die letzte verbleibende Struktur eines Arbeitsablaufs ist das Konstrukt <result>, mit dessen Hilfe die Ergebnisse bzw. die Ausgabe der einzelnen Aktivitäten verarbeitet werden können. Die möglichen Ausprägungen umfassen:

emptyResult Falls das Ergebnis einer Aktivität verworfen werden soll, wird dies mit-

Listing 6 Arbeitsablauf mit iterativer Parallelausführung

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <process name="sampleworkflow">
3   <task name="iterateParallelDemo">
4     <description>Shows usage 1 of iterateParallel task</description>
5     <type>iterateParallelTask</type>
6     <result>
7       <type>screenResult</type>
8     </result>
9     <fork>echo "Hello " </fork>
10    <parameterValues>World; Vienna; Austria</parameterValues>
11    <antecessor>iterateParallelDemo2</antecessor>
12  </task>
13  <task name="iterateParallelDemo2">
14    <description>Shows usage 2 of iterateParallel task</description>
15    <type>iterateParallelTask</type>
16    <result>
17      <type>screenResult</type>
18    </result>
19    <fork>echo "Execution no: " </fork>
20    <parameterValues>1 : 10</parameterValues>
21    <antecessor></antecessor>
22  </task>
23 </process>
```

Listing 7 Arbeitsablauf mit Aktivitätswiederholung

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <process name="repeatprocess">
3   <repeat name="repeater1" iterations="2">
4     <task name="StartNode">
5       <description>Einstiegsknoten</description>
6       <type>execTask</type>
7       <result>
8         <type>screenResult</type>
9       </result>
10      <antecessor>SecondNode</antecessor>
11      <fork>echo "Hello World" </fork>
12    </task>
13    <task>
14      <name>SecondNode</name>
15      <description>Zweiter Knoten</description>
16      <type>execTask</type>
17      <result type="fileResult">
18        <fileName>secondnode.txt</fileName>
19        <appendFile>true</appendFile>
20      </result>
21      <antecessor></antecessor>
22      <fork>echo "Hello World" </fork>
23    </task>
24  </repeat>
25 </process>
```

Listing 8 Arbeitsablauf mit Parallelaktivitäten

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <process name="parallelProcess">
3   <parallel name="parallel1">
4     <task name="StartNode">
5       <description>Parallel Node 1</description>
6       <type>dbTask</type>
7       <query>select * from customer</query>
8       <dbName>hsqldbDS</dbName>
9       <result>
10        <type>screenResult</type>
11      </result>
12      <antecessor></antecessor>
13      <resumeOnError>false</resumeOnError>
14    </task>
15    <task>
16      <name>Second Node</name>
17      <description>Parallel Node 2</description>
18      <type>execTask</type>
19      <result>
20        <type>fileResult</type>
21        <fileName>pingresult.txt</fileName>
22        <appendFile>true</appendFile>
23      </result>
24      <antecessor></antecessor>
25      <fork>ping localhost</fork>
26      <exitvalues>0, 1, 143</exitvalues>
27    </task>
28  </parallel>
29 </process>
```

tels dieses Typs gekennzeichnet. Dabei können keine weiteren Parameter angegeben werden. Ein Beispiel für diesen Typ ist in Listing 5 ersichtlich.

fileResult Die Ausgabe der Aktivität wird in eine Datei mit dem Namen aus dem entsprechenden Parameter „fileName“ geschrieben. Mittels des optionalen Parameters `appendFile` kann angegeben werden, ob eine bestehende Datei erweitert werden soll oder ob sie überschrieben werden soll. Listing 8 gibt ein Beispiel für diesen Ergebnistyp.

screenResult Das Ergebnis der Aktivität wird am Bildschirm ausgegeben, es können keine weiteren Parameter angegeben werden. Listing 8 gibt ein Beispiel für diesen Ergebnistyp.

4.4.2 Kommandozeilenparameter

Die Detailsteuerung, wie sich ein Arbeitsablauf verhalten soll, ist nicht ausschließlich von der XML-Definition des Ablaufs abhängig, sondern kann auch mittels Kommandozeilenparametern gesteuert werden. So ist es beispielsweise möglich, etwa zu Testzwecken den Zugriff auf ein zentrales Repository zu verhindern oder manche Ablaufparameter in einer externen Datei zu spezifizieren. Um auf der Kommandozeile Hilfe zu den verschiedenen Parametern zu bekommen, muss das Programm entweder ohne weitere Parameter gestartet werden oder explizit mittels des Parameters `-h`. Die weiteren Möglichkeiten sind folgende:

local, -l Es wird keine Verbindung zu einem Repository hergestellt.

query, -q Durchsucht das Repository nach der namentlich angegebenen Arbeitsablaufbeschreibung.

queryproperties, -qp Durchsucht das Repository nach den in der Propertydatei angegebenen Parametern.

checkout, -c Holt die aktuelle Version der Arbeitsablaufbeschreibung aus dem zentralen Repository.

properties, -p Verwendet die Parameter in der angegebenen Propertydatei.

workflow, -w Der auszuführende Arbeitsablauf.

repprops Verwendet die in der angegebenen Datei deklarierten Parameter für den Zugriff auf das Repository.

help, -h Gibt eine Liste der möglichen Kommandozeilenparameter aus.

4.4.3 Konfigurationsdateien

Für die Einstellung von selten wechselnden Parametern gibt es verschiedene Dateien, mittels derer unterschiedliche Aspekte konfiguriert werden können.

taskbeans.xml In dieser Datei werden die notwendigen Parameter für das Spring-Framework (siehe Kapitel 4.2.1) eingestellt, primär handelt es sich dabei um die Zuordnung der unterschiedlichen Aktivitäten (siehe Kapitel 4.4.1) zu den entsprechenden Java-Klassen. Eigene Erweiterungen der Grundfunktionalität erfolgen durch „Registrierung“ des neuen Typs als Bean (siehe Listing 9), ansonsten

sind an dieser Datei keine Änderungen durchzuführen.

Listing 9 Beispiel für taskbeans.xml.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:util="http://www.springframework.org/schema/util"
5     xmlns:context="http://www.springframework.org/schema/context"
6     xsi:schemaLocation="
7         http://www.springframework.org/schema/beans
8         http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
9         http://www.springframework.org/schema/util
10        http://www.springframework.org/schema/util/spring-util-2.5.xsd
11        http://www.springframework.org/schema/context
12        http://www.springframework.org/schema/context/spring-context.xsd">
13 <!-- <context:annotation-config /> -->
14 <bean id="dbTask"
15     class="at.workflowexecutor.workflowelements.tasks.DbTask"
16     scope="prototype" />
17 <bean id="execTask"
18     class="at.workflowexecutor.workflowelements.tasks.ExecTask"
19     scope="prototype" />
20 <bean id="iterateParallelTask"
21     class="at.workflowexecutor.workflowelements.tasks.IterateParallelTask"
22     scope="prototype" />
23 <bean id="parallelStructure"
24     class="at.workflowexecutor.workflowelements.controlstructures.
25         ParallelStructure" scope="prototype" />
26 <bean id="repeatStructure"
27     class="at.workflowexecutor.workflowelements.controlstructures.
28         RepeatStructure" scope="prototype" />
29 <bean id="screenResult"
30     class="at.workflowexecutor.workflowelements.results.ScreenResult"
31     scope="prototype" />
32 <bean id="fileResult"
33     class="at.workflowexecutor.workflowelements.results.FileResult"
34     scope="prototype" />
35 <bean id="emptyResult"
36     class="at.workflowexecutor.workflowelements.results.EmptyResult"
37     scope="prototype" />
38 </beans>
```

repository.properties Diese Datei enthält die Konfigurationsparameter für den Zugriff auf ein Subversion-Repository. Die Einstellungen werden wie unter Java üblich mit einem Schlüssel=Wert - Eintrag pro Zeile vorgenommen. Die drei möglichen Einträge in dieser Datei sind *location* (hier wird das Wurzelverzeichnis des Repositorys angegeben), *user* für den zu verwendenden Benutzernamen und schlussendlich *password* für die Angabe des Passworts.

log4j.properties In dieser Datei werden etwaige Einstellungen für die Protokollierung

des Programms vorgenommen. Im Auslieferungszustand werden auf der Kommandozeile nur die wichtigsten Meldungen (etwa Fehler oder Fortschrittsanzeigen) dargestellt, eine detailliertere Ausgabe mit Meldungen zur Fehlersuche wird in eine separate Datei im Unterverzeichnis *logs* geschrieben. Die Einstellungen entsprechen den Vorgaben des log4j-Frameworks (siehe Kapitel 4.2.6).

jndi.properties Diese Datei muss sich im Classpath der Applikation befinden, um keine Fehlermeldung zu erzeugen. Hier wird konfiguriert, in welchem Verzeichnis nach Datenbank-Parametern gesucht wird. Der erste Eintrag in der Datei (*java.naming.factory.initial=org.osjava.sj.SimpleContextFactory*) ist fix vorgegeben und darf nicht verändert werden. Über die Angabe der Zeile *org.osjava.sj.root=config/* kann ein relativer oder absoluter Pfad (in diesem Fall wird aus *config/* der Wert */config/*) angegeben werden, in dem nach Datenbankkonfigurationen gesucht wird. [OSJava 08a]

db.properties Diese Datei dient als Beispiel-Konfigurationsdatei für die Einrichtung von Datenbankzugriffen. Für jede verschiedene Datenbank, die von einer Arbeitsablaufdatei mittels eines DbTasks (siehe Kapitel 4.4.1) angesprochen werden soll, muss eine entsprechende Datei erstellt werden. In der Datei muss mittels des Schlüssels *type=javax.sql.DataSource* angegeben werden, dass es sich um eine Java-konforme Datenbank handelt. Danach folgen die Einträge für den Datenbanktreiber mittels *driver=org.gjt.mm.mysql.Driver* und die Angabe, wo die entsprechende Datenbank zu finden ist mit *url=jdbc:mysql://localhost/testdb*. Abschließend können mit Zeilen für Benutzernamen (*user=testuser*) und Passwort (*password=testing*) noch die Zugangsdaten angegeben werden. [OSJava 08b]

4.5 Subversion-Integration

Durch den Einsatz des SVNkit-Frameworks (siehe Abs. 4.2.2) besteht die Möglichkeit, Arbeitsabläufe an einem zentralen Aufbewahrungsort zu speichern. Das ermöglicht einerseits die Versionskontrolle zur Nachvollziehbarkeit der Entwicklung eines Workflows, andererseits wird dadurch die Anforderung aus der wissenschaftlichen Praxis erfüllt, dass alle Schritte zur Erzeugung eines Ergebnisses nachvollziehbar sein müssen. Darüber hinaus ist durch die Verwendung von Revisionsproperties die Möglichkeit gegeben, bereits existierende Zwischenergebnisse wiederzuverwenden anstatt diese bei jedem Durchlauf des Arbeitsablaufs neu zu erstellen. Aufgrund der teilweise sehr lan-

gen Rechenzeiten bzw. begrenzter Ressourcen auf Supercomputern ist dies ein nicht zu vernachlässigender Vorteil.

Die Programmlogik bietet verschiedene Möglichkeiten, ein Repository nach existierenden Arbeitsabläufen zu durchsuchen (siehe Kapitel 4.4.2), sodass für Benutzer keine Notwendigkeit besteht, sich in ein anderes Programm einzuarbeiten oder externe Anwendungen zu installieren. Dabei ist zu beachten, dass Arbeitsabläufe programmintern durch sog. *tags* identifiziert werden, die entweder durch die Applikation automatisch vergeben werden (wie etwa der tag *name*), oder durch den Benutzer manuell definiert werden. Dies erfolgt innerhalb der Arbeitsablaufbeschreibung durch Verwendung des Parameters *property* (siehe Kapitel 4.4.1). Damit wird es möglich, unterschiedliche Kennzeichnungen in einem Arbeitsablauf durch den Einsatz verschiedener Parameter zu kennzeichnen. Im Repository entsteht dadurch für jeden Arbeitsablauf eine baumartige Struktur, die ausgehend von einem „Wurzelworkflow“ alle Verzweigungen beinhaltet.

4.6 Erweiterungsmöglichkeiten

Dieser Abschnitt behandelt mögliche Erweiterungen des bisher vorgestellten Konzeptes und soll gleichzeitig einen ersten Ausblick auf Folgeversionen geben. Die meisten der hier angegebenen Punkte stammen aus Entscheidungen im Vorfeld bzw. während der Entwicklung, und sind durch Priorisierungen nicht in die erste hier vorgestellte Version des *WorkflowExecutors* gelangt.

weitere Aktivitätstypen Die erste Auslieferungsversion des Programms enthält primär nur die Umsetzung der wichtigsten Aktivitäten aus der Anforderungsanalyse. Weitere interessante Typen für Folgeversionen wären etwa ein Modul zur Abfrage von Webservices oder die Integration von Schnittstellen zu wissenschaftlichen Rechnerverbünden.

weitere Ergebnistypen Nicht nur die Liste der Aktivitäten ist erweiterbar, auch die möglichen Ergebnistypen können ausgeweitet werden, etwa durch einen Datenbanktyp.

Fortschrittskontrolle In der aktuellen Version wird der Arbeitsfortschritt nur durch Ausgaben zu Beginn und Ende einer Aktivität dargestellt. Wenn ein lang dauernder Task nicht selbständig Nachrichten an den Benutzer sendet, gibt es keinerlei visuelle Anzeige, ob das Programm beschäftigt ist oder vielleicht abgestürzt ist.

Benutzerinteraktion Derzeit ist es für den Anwender nicht möglich, in einen laufenden Prozess manuell einzugreifen, um etwa notwendige Eingaben zu machen (diese müssen im Vorfeld als Parameter übergeben werden).

XML-Schema Eine Möglichkeit, neu erstellte Arbeitsabläufe im Vorfeld auf syntaktische Korrektheit zur überprüfen, könnte durch Bereitstellung eines XML-Schemas erreicht werden.

Spring Framework Obwohl die Applikation auf dem Spring Framework basiert, wäre eine bessere Integration der Möglichkeiten (etwa durch Ausnutzung von *Objekt-Relational-Mapping* im Datenbankmodul) dieses Containers wünschenswert.

4.7 Zusammenfassung

In diesem Kapitel wurde der interne Aufbau der Applikation sowie die verschiedenen zum Einsatz gelangenden Softwarekomponenten behandelt. Darüber hinaus wurden die unterschiedlichen Möglichkeiten zum Erstellen von Arbeitsablaufsbeschreibungen detailliert aufgezeigt und dargestellt, in welcher Form das Programm bestmöglich an die Arbeitsplatsumgebung angepasst werden kann. Zum Abschluss des Kapitels wurde die Zusammenarbeit mit einem zentralen Subversion-Repository beschrieben und Verbesserungsmöglichkeiten aufgezeigt.

Kapitel 5

Validierung

Um die Leistungsfähigkeit der hier vorgestellten Applikation zu überprüfen, wird in diesem Kapitel die Implementierung anhand eines Workflows aus dem täglichen Forschungsbetrieb an der Universität für Bodenkultur Wien, Chair of Bioinformatics getestet. Dabei wird zuerst näher auf den Anwendungsfall und die darin enthaltenen Problemstellungen eingegangen, anschließend erfolgt die Analyse und Bewertung der erzielten Resultate. Den Abschluss des Kapitels bildet ein Ausblick auf mögliche Verbesserungen und notwendige Erweiterungen, die sich im Rahmen der Evaluierung ergeben haben.

5.1 Beschreibung des Anwendungsfalls zur Sequenzierung eines Mausgenoms

Für die Validierung wurde die textuelle Beschreibung (siehe Anhang A ab Seite 66) eines typischen Analyse-Workflows der Boku Wien verwendet. Der vorgegebene Ablauf sieht vor, dass eine große Menge an RNA-Sequenzen von Mäusegehirnen (diese Daten stammen aus einem Hochleistungs-Sequenzierer) gegen ein Referenzdatenset verglichen wird. Gefundene Abweichungen werden auf mit E.coli-Bakterien kontaminierte Stellen überprüft.

Die Abarbeitung dieses prinzipiellen Workflows sieht vor, dass eine Menge von verschiedenen Skripten ausgeführt wird, wobei in der Ablaufbeschreibung spezifiziert werden kann, ob und welche Teile etwa parallel exekutiert werden können. Die zur Anwendung gelangenden Skripte werden extern zur Verfügung gestellt. Eine graphische Darstellung

des schematischen Workflows ist in Abb. 5.1 zu sehen.

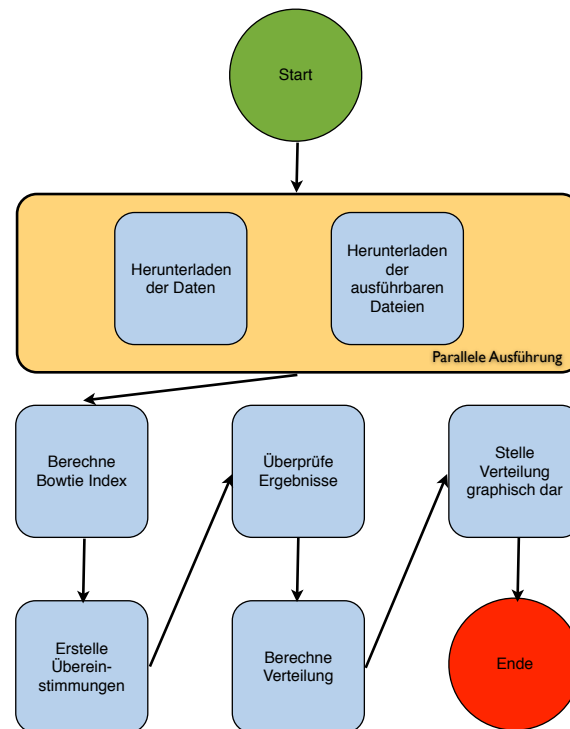


Abbildung 5.1: Schematischer Workflow.

5.2 Umsetzung des Anwendungsfalles

Durch die ausführliche Beschreibung des Anwendungsfalles erfordert die Aufteilung in die verschiedenen Aktivitäten keinen weiteren Aufwand, sondern kann direkt erfolgen. Die Abhängigkeiten der einzelnen Tasks sind ebenfalls bereits vorgegeben, sodass auch diesem Punkt keine gesonderte Aufmerksamkeit gewidmet werden muss.

5.2.1 Vorbereitungsarbeiten

Für die fehlerfreie Ausführung des Arbeitsablaufs sind einige grundlegende Vorbereitungsarbeiten zu tätigen, die nicht im Rahmen des Workflows abgebildet werden. Darunter fallen einerseits die korrekte Installation und Einrichtung des *WorkflowExecutors*, andererseits aber auch die Überprüfung, ob notwendige Skripts aus dem lokalen „Skript Repository“ im Pfad ausführbar sind.

Für die lokalen Tests wurde entschieden, keinen Zugriff auf Subversion vorzunehmen, sondern den Workflow „standalone“ auszuführen. Dies erfordert entweder die Konfiguration über die Parameterdatei *repository.properties* (vgl. Abschnitt 4.4.3 „Konfigurationsdateien“) oder die Kommandozeilenoption „-l“. Des weiteren benötigt der für die Validierung herangezogene Workflow keinen Zugriff auf eine Datenbank, sodass spezielle Einstellungen in den anderen Konfigurationsdateien (*jndi.properties* bzw. *db.properties*) nicht nötig sind.

Aufgrund der großen Datenmenge wurde beschlossen, für die Tests das Herunterladen der Dateien von den angegebenen FTP-Servern nicht in die Workflow-Beschreibung zu integrieren, um bei mehrmaligen Versuchen keine unnötigen Netzwerkbelastungen zu verursachen. Da das Vorhandensein dieser Dateien im lokalen Verzeichnis eine notwendige Voraussetzung für die Ausführung des Workflows ist, wurde sichergestellt, dass die in der Ablaufbeschreibung angegeben Befehle

```
wget 'http://woldlab.caltech.edu/rnaseq/mm9Brain1.comb.fa.gz',  
wget 'http://woldlab.caltech.edu/rnaseq/mm9Brain2.comb.fa.gz',  
wget 'http://hgdownload.cse.ucsc.edu/goldenPath/mm9/bigZips/refMrna.fa.gz',  
und wget 'ftp://ftp.ncbi.nih.gov/genbank/genomes/Bacteria/Escherichia_coli_K_12_substr_MG1566/U00096.fna'
```

vollständig ausgeführt wurden und die heruntergeladenen Dateien korrekt entpackt wurden. Ebenfalls wurde im Zuge der Vorbereitungsarbeiten das Programmpaket zum Vergleich der Sequenzen auf dem lokalen Rechner installiert, wobei die entsprechende Software in der aktuellen Version wie in der Beschreibung angegeben von <http://sourceforge.net/projects/bowtie-bio/files/bowtie/>¹ heruntergeladen wurde.

5.2.2 Erstellung der Workflowbeschreibung

In diesem Testschritt wurde überprüft, ob die XML-Syntax den Anforderungen hinsichtlich der einfachen Bearbeitung mit einem Texteditor genügt. Für die Erstellung wurden keinerlei mündliche Anleitungen gegeben, sondern ausschließlich ein exemplarischer Workflow zur Darstellung der möglichen Aktivitäten zur Verfügung gestellt. Trotz der relativen Komplexität des zu beschreibenden Anwendungsfalles erwies sich die Formatbeschreibung als einfach und intuitiv genug, um innerhalb kurzer Zeit einen vollständigen Workflow zur Verfügung zu haben. Dabei wurden auch komplexere Elemente wie der erste Task zur Überprüfung der Korrektheit der benötigten Dateien (vgl.

¹siehe ebd. für eine Beschreibung; zuletzt abgerufen am 31. Juli 2010

Listing 10) problemlos erstellt.

Listing 10 Einzelner Task aus dem Referenzworkflow

```
1 <task name=" EvalRes">
2   <description>Evaluate the alignment results , splitting data into
      unique- and multi- hit segments</description>
3   <type>iterateParallelTask</type>
4   <result>
5     <type>screenResult</type>
6   </result>
7   <fork>./ eval_bowtie_output . pl</fork>
8   <parameterValues>../ Workflow_data/mm9Brain1 . aligned . bowtie ;
9     ../ Workflow_data/mm9Brain1 . uniq . aligned . bowtie ;
10    ../ Workflow_data/mm9Brain1 . multi . aligned . bowtie ;
11    ../ Workflow_data/mm9Brain2 . aligned . bowtie ;
12    ../ Workflow_data/mm9Brain2 . uniq . aligned . bowtie ;
13    ../ Workflow_data/mm9Brain2 . multi . aligned . bowtie
14  </parameterValues>
15  <antecessor>ReadCalc</ antecessor>
16 </task>
```

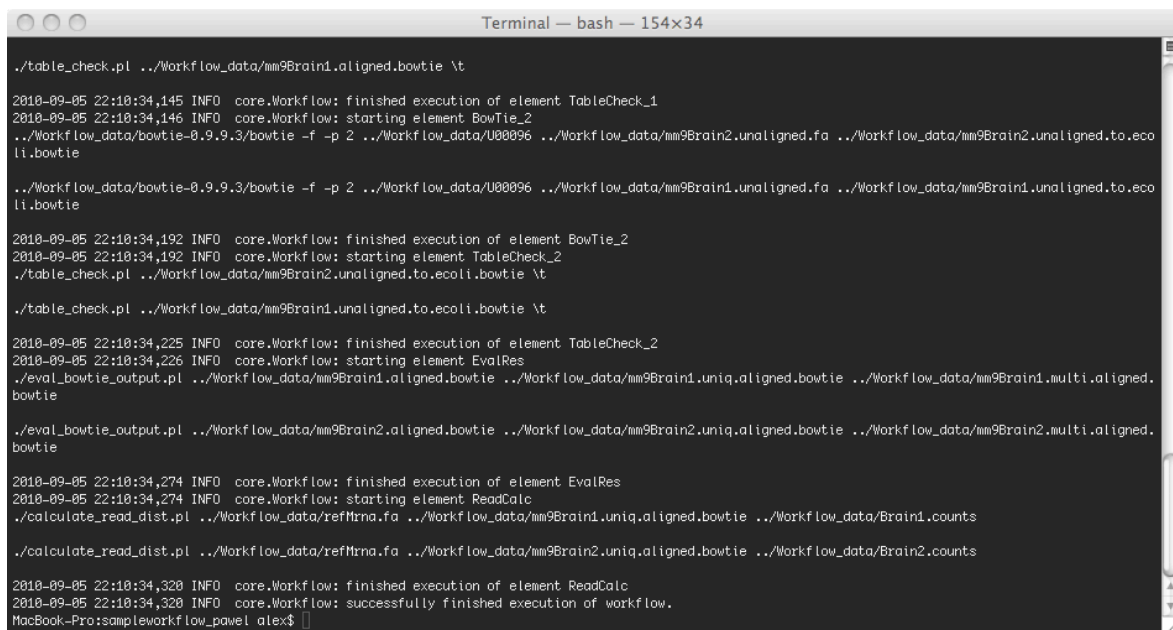
Bei der Umsetzung wurde die Ausgabe der einzelnen Task auf dem Bildschirm konfiguriert (vgl. Zeile 5 in Listing 10), sodass die Überprüfung der Ausführung einfach erfolgen konnte. Für eine vollständige Referenz des erstellten Workflows sei auf Anhang B auf Seite 72 verwiesen.

5.2.3 Ausführen des Arbeitsablaufs

Die Ausführung des Arbeitsablaufs wurde sowohl unter dem Betriebssystem Linux als auch unter Mac OS X auf verschiedenen Computern getestet. Zum Starten des Workflows wurde dabei auf der Kommandozeile der Befehl

```
java -jar WorkflowExecutor.jar -l -w SampleWorkflow.xml
```

verwendet. Dabei ist zu beachten, dass alle notwendigen Dateien zur Ausführung des Workflows im lokalen Verzeichnis auffindbar sein müssen. Die Ausgaben werden während der Ausführung des Programms auf der Kommandozeile ausgegeben (vgl. Abb. 5.2).



```
Terminal — bash — 154x34

./table_check.pl ../Workflow_data/mm9Brain1.aligned.bowtie \t
2010-09-05 22:10:34,145 INFO core.Workflow: finished execution of element TableCheck_1
2010-09-05 22:10:34,146 INFO core.Workflow: starting element BowTie_2
../Workflow_data/bowtie-0.9.9.3/bowtie -f -p 2 ../Workflow_data/U00096 ../Workflow_data/mm9Brain2.unaligned.fa ../Workflow_data/mm9Brain2.unaligned.to.ecoli.bowtie
../Workflow_data/bowtie-0.9.9.3/bowtie -f -p 2 ../Workflow_data/U00096 ../Workflow_data/mm9Brain1.unaligned.fa ../Workflow_data/mm9Brain1.unaligned.to.ecoli.bowtie
2010-09-05 22:10:34,192 INFO core.Workflow: finished execution of element BowTie_2
2010-09-05 22:10:34,192 INFO core.Workflow: starting element TableCheck_2
./table_check.pl ../Workflow_data/mm9Brain2.unaligned.to.ecoli.bowtie \t
./table_check.pl ../Workflow_data/mm9Brain1.unaligned.to.ecoli.bowtie \t
2010-09-05 22:10:34,225 INFO core.Workflow: finished execution of element TableCheck_2
2010-09-05 22:10:34,226 INFO core.Workflow: starting element EvalRes
./eval_bowtie_output.pl ../Workflow_data/mm9Brain1.aligned.bowtie ../Workflow_data/mm9Brain1.uniq.aligned.bowtie ../Workflow_data/mm9Brain1.multi.aligned.bowtie
./eval_bowtie_output.pl ../Workflow_data/mm9Brain2.aligned.bowtie ../Workflow_data/mm9Brain2.uniq.aligned.bowtie ../Workflow_data/mm9Brain2.multi.aligned.bowtie
2010-09-05 22:10:34,274 INFO core.Workflow: finished execution of element EvalRes
2010-09-05 22:10:34,274 INFO core.Workflow: starting element ReadCalc
./calculate_read_dist.pl ../Workflow_data/refMrna.fa ../Workflow_data/mm9Brain1.uniq.aligned.bowtie ../Workflow_data/Brain1.counts
./calculate_read_dist.pl ../Workflow_data/refMrna.fa ../Workflow_data/mm9Brain2.uniq.aligned.bowtie ../Workflow_data/Brain2.counts
2010-09-05 22:10:34,320 INFO core.Workflow: finished execution of element ReadCalc
2010-09-05 22:10:34,320 INFO core.Workflow: successfully finished execution of workflow.
MacBook-Pro:sampleworkflow_pawel alex$
```

Abbildung 5.2: Screenshot des WorkflowExecutor bei der Ausführung

5.3 Resultate

Der Einsatz der im Rahmen dieser Diplomarbeit erstellten Applikation hat sowohl mit selbsterzeugten Testarbeitsabläufen als auch mit dem exemplarischen Referenzwork-flow (siehe Anhang A ab Seite 66 für die textuelle Beschreibung, Anhang B ab Seite 72 für die Umsetzung) interessante Ergebnisse geliefert. So hat sich etwa die Einarbeitungszeit als sehr gering herausgestellt (es reichte die Betrachtung eines Beispiels zur selbständigen Verfassung des Referenzarbeitsablaufs) und die implizite Dokumentation durch die verwendete Beschreibungsstruktur wurde positiv hervorgehoben.

Als wichtiger Beitrag kann auch die Umsetzung von parametrisierbaren Arbeitsaufgaben angesehen werden, mit deren Hilfe der Umfang von Arbeitsabläufen deutlich verringert werden kann. Einzelne sich wiederholende Tasks können natürlich ohne die Angabe von Parametern entweder sequentiell oder auch parallel abgearbeitet werden, müssen dafür aber jedes mal neu modelliert werden. Auch die Verwendung von Parametern innerhalb der aufgerufenen Skripte bietet keine Möglichkeit, diese Variablen explizit zu kennzeichnen.

Ein augenscheinlicher Vergleich für prinzipiell idente Arbeitsabläufe mit und ohne Zuhilfenahme der Möglichkeit der Parametrisierung von Tasks ist in den Listings 11 und 12 zu sehen. Die beiden Workflows überprüfen eine Reihe von Dateien auf ihre Korrektheit, indem sie mehrmals das Skript mit dem Namen „nt_fasta_check.pl“ mit

unterschiedlichen Parametern aufrufen. In Listing 12 erfolgt diese Parametrisierung derart, dass das Skript mehrmals nacheinander aufgerufen wird. Die zweite Version des identen Arbeitsablaufs reduziert sich auf einen einzelnen Arbeitsschritt, der zusätzlich eine Liste der Parameter erhält.

Listing 11 Workflow mit Parametrisierung

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <process name="workflow_non_param">
3   <task name="FastaCheck">
4     <description>Checking if fasta files are correct</description>
5     <type>iterateParallelTask</type>
6     <result>
7       <type>screenResult</type>
8     </result>
9     <fork>./nt_fasta_check.pl</fork>
10    <parameterValues>U00096.fna;refMrna.fa;mm9Brain1.comb.fa 1-90
        10-1e9;mm9Brain2.comb.fa 1-90 10-1e9
11    </parameterValues>
12    <antecessor>BowTieCheck</antecessor>
13  </task>
14 </process>
```

5.4 Verbesserungspotential

Obwohl sich das Tool in den durchgeführten Tests sowohl die ursprünglichen als auch spätere Anforderungen betreffend als ausreichend robust herausgestellt hat, haben sich bereits während der Umsetzungsphase einige Schwachstellen herauskristallisiert, die jedoch nicht mehr im Rahmen der ersten Realisierung behoben werden konnten. Darüber hinaus gibt es zusätzlich zur Grundfunktionalität einige Anforderungen, deren Umsetzung in folgenden Versionen thematisiert werden sollte.

Die erste Gruppe der notwendigen bzw. wünschenswerten Anpassungen umfasst primär sehr „praktische“ Punkte, deren Umsetzung unter Berücksichtigung des bestehenden Programmcodes relativ einfach durchzuführen ist. Für die Realisierung der Wünsche der zweiten Gruppe jedoch sind grundlegende Änderungen an der bestehenden Implementierungslogik notwendig, diese sind somit nicht einfach umzusetzen.

Listing 12 Workflow ohne Parametrisierung

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <process name="workflow_non_param">
3   <task name="NonParamTask1">
4     <description>Checking if fasta files are correct 1</description>
5     <type>execTask</type>
6     <result>
7       <type>screenResult</type>
8     </result>
9     <fork>./ nt_fasta_check.pl U00096.fna</fork>
10    <antecessor>NonParamTask2</antecessor>
11  </task>
12  <task name="NonParamTask2">
13    <description>Checking if fasta files are correct 2</description>
14    <type>execTask</type>
15    <result>
16      <type>screenResult</type>
17    </result>
18    <fork>./ nt_fasta_check.pl refMrna.fa</fork>
19    <antecessor>NonParamTask3</antecessor>
20  </task>
21  <task name="NonParamTask3">
22    <description>Checking if fasta files are correct 3</description>
23    <type>execTask</type>
24    <result>
25      <type>screenResult</type>
26    </result>
27    <fork>./ nt_fasta_check.pl mm9Brain1.comb.fa 1-90 10-1e9</fork>
28    <antecessor>NonParamTask4</antecessor>
29  </task>
30  <task name="NonParamTask3">
31    <description>Checking if fasta files are correct 3</description>
32    <type>execTask</type>
33    <result>
34      <type>screenResult</type>
35    </result>
36    <fork>./ nt_fasta_check.pl mm9Brain2.comb.fa 1-90 10-1e9</fork>
37    <antecessor></antecessor>
38  </task>
39 </process>
```

5.4.1 Löschen aus dem Repository

In der aktuellen Version des Tools ist keine Möglichkeit vorgesehen, Daten programmgesteuert aus dem zentralen Subversion-Repository zu entfernen. Ein manueller Eingriff durch einen Administrator des Repositories ist zwar jederzeit möglich (und würde auch in der Historie aufscheinen), aufgrund der Gefahr, wichtige Workflow-Daten unabsichtlich zu löschen aber nicht empfehlenswert.

Eine erste Umsetzung dieser Anforderung ist relativ einfach möglich, da sie nicht zwingend in die bestehende Programmlogik eingreifen muss, sondern einfach als zusätzliche Kommandozeilenoption angeboten werden könnte. Eine nachhaltigere und aus Sicht der Anwenderfreundlichkeit sicherlich einfachere Möglichkeit wäre aber, zusätzlich für Workflows gewisse Alterungsmechanismen einzubauen. So wäre etwa eine Variante, vor oder nach jedem Lauf des Programms eine Liste von Workflows auszugeben, deren Zwischenergebnisse seit einer konfigurierbar langen Zeitspanne nicht mehr abgerufen worden sind. Der Benutzer sollte dann die Möglichkeit bekommen, die nicht mehr notwendigen Resultate zu entfernen.

Ein anderer Ansatz um dieser Problematik Herr zu werden wäre, nicht die Daten selbst ins Repository zu übertragen, sondern ausschließlich Links auf diese Daten. Auf diese Art und Weise würde das Problem der großen Datenmengen einerseits nicht im zentralen Repository auftreten, andererseits ist die Nachvollziehbarkeit von Workflows im Zeitverlauf nicht ganz sichergestellt. Die Umsetzung dieser Änderung könnte zentral im Code zur Speicherung bzw. Wiederherstellung von Ergebnissen erfolgen.

5.4.2 Überprüfung des freien Arbeitsspeichers

Bei der Ausführung von parallel laufenden Aktivitäten wird aktuell keine Rücksicht darauf genommen, ob diese mit dem verfügbaren Arbeitsspeicher ausgeführt werden können, sondern es wird unabhängig davon die Gesamtanzahl an nebenläufigen Tasks ausgeführt. Dieses Verhalten führt vor allem in Kombination mit speicherintensiven Berechnungen zu unnötigen Verlängerungen der Ausführungszeit, da computerintern ständig zwischen den unterschiedlichen laufenden Threads gewechselt werden muss.

Aus diesem Grund wäre es wünschenswert, wenn neue parallel ablaufende Aktivitäten erst dann gestartet werden, wenn soweit möglich sichergestellt ist, dass ausreichend Arbeitsspeicher zur Verfügung steht. Eine Möglichkeit zur Umsetzung wäre, das Starten von Threads mit einer gewissen Vorlaufzeit zu verzögern, um am Ende dieser „ramp up

time“ den Speicherverbrauch des ersten Threads zu messen und daraufhin zu entscheiden, ob ausreichend Arbeitsspeicher für eine weitere Instanz des Threads zur Verfügung steht.

Eine Umsetzung in dieser Form würde zwar die parallele Ausführung für sehr kurz laufende Aktivitäten etwas verzögern bzw. in manchen Fällen komplett verhindern, wäre aber hinsichtlich der lang laufenden und speicherintensiven Aktivitäten eine Möglichkeit, Engpässe zu verhindern. Um die Vorgehensweise für alle parallelen Aktivitäten zu kapseln wäre es sinnvoll, diese Funktionalität in eine eigene Klasse auszulagern, die sich zentral um die Ablaufsteuerung von Parallelaktivitäten kümmert.

5.4.3 Verifikation von Resultaten

Vor allem bei lange laufenden Workflows wäre es von Vorteil, wenn Zwischenresultate mit einer gewissen Wahrscheinlichkeit als Richtig klassifiziert werden können. Sobald ein frei einstellbarer Grenzwert überschritten wird, wird die aktuelle Ausführung gestoppt, bis der Anwender den Befehl zum Fortfahren gibt.

Die Umsetzung einer solchen Überprüfung wäre aus technischer Sicht nicht sonderlich schwierig, die Probleme liegen dabei eher in der Bewertung der Qualität eines Ergebnisses. Ein möglicher erster Ansatz könnte sein, dass innerhalb der XML-Struktur der Ablaufbeschreibung semantische Informationen hinterlegt werden können, etwa was die Anzahl der erwarteten Ergebnisdateien betrifft. In der Applikation selbst wäre das mit Änderungen an der Basisfunktionalität verbunden, da das Einlesen der XML-Struktur davon betroffen ist. Die Validierung selbst kann an zentraler Stelle beim Speichern des Resultats in das Subversion-Repository erfolgen.

5.4.4 Automatische Fehlerbehandlung

Die konsequente Weiterführung der Verifikation von Resultaten ergibt als nächsten Punkt die automatische Reaktion auf Fehler, sei es in Daten selbst oder in den Ergebnissen. Die adäquate Reaktion auf fehlerbehaftete Informationen setzt ein hohes Maß an semantischem Wissen über die bearbeiteten Daten voraus, sodass ein zu spezifischer Ansatz zur Verifikation mit hoher Wahrscheinlichkeit scheitern wird.

Die erfolgsversprechende Variante ist sicherlich auch hier, Anknüpfungspunkte in die Datenstruktur zu integrieren, um für die jeweilige Aktivität notwendiges semantisches

Wissen hinzuzufügen, und gleichzeitig Fehlerbehandlungsroutinen zu benennen, die nach dem Auftreten eines Fehlers aufgerufen werden sollen. Dabei können Strukturen von Vorteil sein, die bestimmte Fehler in Klassen zusammenfassen, um so den Aufwand für Fehlerbehandlungen möglichst gering zu halten.

Die Implementierung einer solchen Funktionalität erfordert sicherlich eine intensive Auseinandersetzung mit den Problemen im Gebiet der Fehlererkennung, könnte aber im vorgelegten Framework durch Dependency Injection zumindest aus programmier-technischer Sicht gut gelöst werden.

5.4.5 Datenzentrierter Ansatz

Die Änderung der Sichtweise auf einen Workflow weg von der Ablaufbeschreibung hin zu einem datenzentrierten Blickwinkel ist aus Anwendersicht sicherlich ein wünschenswertes Ziel. Die Erleichterungen, die für Benutzer beim Zusammenstellen von Workflows dadurch entstehen, sind nicht von der Hand zu weisen, besteht hier doch nicht mehr die Notwendigkeit, sich Gedanken über die prinzipielle Flusskontrolle zu machen. In dieser Form der Zusammenstellung eines Arbeitsablaufs wird „nur“ noch die gewünschte Transformation von Eingabedaten zu Ausgabedaten modelliert, das Wissen über die konkrete Durchführung dieser Transformationsschritte ist in den Datenstrukturen selbst vorhanden. Obwohl ein solches Vorgehen sicherlich einen gewissen Reiz birgt, ist die Umsetzung nicht trivial und würde vor allem die hier vorgestellten Rahmenbedingungen sprengen.

5.5 Zusammenfassung

In diesem Kapitel wurde die vorgestellte Applikation anhand eines konkreten Workflows getestet. Dafür wurde ein exemplarischer Arbeitsablauf vorgestellt, und anschließend die notwendigen Schritte zur bis zur Ausführung des Workflows beschrieben. Aus den erreichten Resultaten wurden Schlüsse für Verbesserungspotential in zukünftigen Versionen abgeleitet und teilweise Implementierungsvorschläge geliefert.

Kapitel 6

Schlussbetrachtungen

Workflows im Sinne einer koordinierten Abfolge von Tätigkeiten bekommen auch in der relativ neuen Wissenschaft Bioinformatik einen immer höheren Stellenwert. Aktuell existiert eine Vielzahl an Arbeitsumgebungen für unterschiedliche Zielgruppen, die die Erstellung und Ausführung solcher Arbeitsabläufe unterstützen. Eine der augenscheinlichsten Gemeinsamkeiten innerhalb dieser Applikationen - seien sie für die Ausführung von Geschäftsprozessen oder für die Unterstützung von wissenschaftlichen Workflows entwickelt worden - ist die Fokussierung auf graphische Editoren zur Erstellung von Prozessabläufen.

Oftmals ist es jedoch notwendig, einen leichtgewichtigeren Ansatz zu wählen, um nicht mit zusätzlichem Lernaufwand und Verwaltungstätigkeiten belastet zu werden, ohne jedoch auf die Notwendigkeit von konsistenter Datenspeicherung und Nachvollziehbarkeit von Änderungen an Arbeitsabläufen zu verzichten.

In dieser Diplomarbeit wurde ein Schritt in diese Richtung gemacht, indem die Implementierung einer Applikation zur formalen Beschreibung und Ausführung von Workflows vorgestellt wurde. Nach einem einführenden Blick auf den aktuellen Stand der Technik wurde die Architektur und die Funktionsweise des Programms dargestellt.

6.1 Ergebnisse der Arbeit

Zu den wichtigsten Ergebnissen zählen die Umsetzung einer vollständigen Integration für das Versionskontrollsystem Subversion sowie die exemplarische Unterstützung einer Vielzahl von unterschiedlichen Aktivitäten zur Gestaltung von Workflows. So kann etwa

die Ablaufsteuerung mit Tasks zur parallelen Abarbeitung oder mit Verzweigungen abhängig von Ergebnissen auf einer atomaren Ebene gesteuert werden.

Die bestehenden Tasks zur Abfrage von Datenbanken (mit Unterstützung beliebiger Abfragen in SQL), Ausführung von externen Kommandozeilenprogrammen und die parametergesteuerte Iteration über eine Aktivität können beliebig erweitert werden, wenn die Notwendigkeit besteht. Dafür wurde das Konzept der Dependency Injection verwendet, sodass die Grundfunktionalität nicht geändert werden muss.

Die Konfiguration des Tools erfolgt vollständig über Parameter, sodass alle unterschiedlichen Einstellungen ohne Änderungen am Programmcode erfolgen können. Die Möglichkeiten sind dabei nicht nur auf Aufrufparameter beschränkt, es können auch auf Aktivitätsebene innerhalb eines Workflows externe Parameter eingestellt werden.

Die Validierung der erstellten Applikation erfolgte durch den Einsatz des Tools zur Abarbeitung eines exemplarischen Workflows aus der wissenschaftlichen Praxis, der unterschiedliche Aspekte wie die Bearbeitung großer Datenmengen oder die parametergesteuerte Ausführung von einzelnen Tasks beinhaltet. Dabei wurde durch die Anwender vor allem der Vorteil der impliziten Dokumentation innerhalb der formalen Beschreibung des Workflows in der zur Anwendung gelangenden XML-Struktur hervorgehoben, wobei gleichzeitig die prinzipielle Funktionsfähigkeit bestätigt wurde.

Die Integration des Versionskontrollsystems Subversion ermöglicht die Nachvollziehbarkeit von Änderungen an Arbeitsabläufen. Dabei wird automatisch während der Ausführung eines Prozesses überprüft, ob es sich um eine Änderung an einem bereits bestehenden Arbeitsablauf handelt, oder ob ein vollständig neuer Workflow in das Repository eingespielt wird. In ersterem Fall besteht zusätzlich die zeitsparende Möglichkeit, auf bereits bestehende Ergebnisse zurückgreifen zu können, sodass idente Zwischenergebnisse nicht mehrmals berechnet werden müssen.

6.2 Ausblick

Wie aus den vorigen Kapiteln ersichtlich, ist mit der Implementierung der hier vorgestellten Applikation erst ein kleiner Schritt hin zu einer vollständigen Ausführungsumgebung für leichtgewichtige Workflows aus dem Umfeld der Bioinformatik getan.

Wenn sich die Applikation in der täglichen Praxis weiterhin bewährt, wäre ein sinn-

voller nächster Schritt aus Sicht der Anwender die Umsetzung von Mechanismen zum Löschen von temporären Daten. Eine Möglichkeit dafür wäre die Referenzierung von Dateien im zentralen Repository von Subversion anstatt diese vollständig dort abzulegen. Die Realisierung dieser Maßnahme hätte den Rahmen der Umsetzung innerhalb dieser Diplomarbeit gesprengt, wäre aber durch die bereits vorhandenen Mechanismen zur Kommunikation mit Subversion ohne größere Änderungen am Basisframework durchführbar. Ein anderer denkenswerter Ansatz ist das Löschen von Daten durch die Applikation direkt aus dem Repository, was im Sinne der Nachvollziehbarkeit von Änderungen am Datenbestand durchaus sinnvoll wäre.

Weitaus größeren Aufwand was sowohl Spezifikation als auch Implementierung angeht, bildet hingegen der Wunsch nach Mechanismen, um auf Fehler in den Daten adäquat reagieren zu können. Durch die Inhomogenität der Daten ist alleine die Erkennung von Fehlerfällen nicht trivial, die darüber hinausgehende Notwendigkeit, programmgesteuert eine Lösung zu finden, erhöht die Komplexität zusätzlich.

Der Ansatz, einzelne Tasks fehlertolerant zu gestalten, wird mit hoher Wahrscheinlichkeit an der Vielfalt der möglichen auftretenden Fehler scheitern. Darüber hinaus wäre diese Vorgehensweise nicht ohne Änderungen am Grundprogramm umsetzbar, sodass die Flexibilität darunter leidet. Ein wahrscheinlich sinnvoller erscheinendes Unterfangen wäre, grundsätzlich in der Implementierung von einzelnen Aktivitäten Anknüpfungspunkte für Fehlerbehandlungsroutinen hinzuzufügen, sodass individuelle Möglichkeiten geschaffen werden können. Durch den Einsatz von Dependency Injection innerhalb des Spring Frameworks könnte so eine Basis geschaffen werden, die auf viele Fälle - entsprechende Programmierkenntnisse vorausgesetzt - entsprechend reagieren kann.

Eine weitere Möglichkeit um besser auf solche generischen Fehler reagieren zu können, wäre die Umsetzung mithilfe von XML-Strukturen, die für bestimmte Fehlerklassen entsprechende Lösungsmöglichkeiten liefern. Dieser Ansatz hätte den Vorteil, dass auch ohne tiefere Programmierkenntnisse relativ einfach ersichtlich wird, warum das Tool für einen Workflow mit fehlerhaften Daten während der Ausführung die entsprechende Vorgehensweise gewählt hat.

Anhang A

Next Generation Sequencing Analysis Workflow

Dieser Anhang beinhaltet die vollständige Beschreibung des Arbeitsablaufs, so wie er in der wissenschaftlichen Praxis beim Chair of Bioinformatics der Universität für Bodenkultur Wien¹ im Einsatz ist.

¹siehe <http://www.biotec.boku.ac.at/bioinf.html>

Next Generation Sequencing Analysis Workflow

Below is a description of a typical Next Generation Sequencing (NGS) analysis workflow for mRNA based whole shotgun sequencing. This involves taking large sets of short sequence reads generated from a high-throughput parallel sequencer (eg. Illumina, ABI Solid, 454 Roche) and aligning the sequences to a reference set of transcript sequences (or genes or genomes). Here we will use as an example, an analysis workflow with a read data set generated from mouse brain to be aligned against the RefSeq transcripts of mouse. The reads will be aligned to the transcripts using the Bowtie short read aligner, and then the output will be used to get the distributions of reads across transcripts. Short read sequences that were not alignable to transcripts will also be evaluated for possible contaminating sequences, in this case from *E.coli* bacteria.

Provided below are example Query files (the Reads), Target Files (the Transcripts), and Contaminants (the Bacterial sequence) that can be substituted with other data files as desired.

The outlined conceptual workflow assumes that certain functions such as splitting, joining, parallelization, archiving, and execution of command lines will be handled by the generalized infrastructure to be developed by the computer scientists. Specific scripts, such as those involved in parsing, quality checks, and analysis of the data files will be designed by the bioinformatics users, which are stored in a Script Repository. Such scripts may be specific to the scientific application (ie. specific for the analysis of a individual program's output) or general enough to handle various common data formats (ie. Tab delimited files, fasta files). In this example workflow, I have included these specific scripts already.

The current release of the scripts (and necessary Perl package "FALite.pm") is available at: </bi/ala/scratch/plabaj/Workflow>.

Please copy it to your private location (ie. /bi/ala/scratch/user_directory) and then execute steps described in "**Example of how the workflow would work at the command line level with a smaller data set**" section of this document.

Data Set:

Query Files:

(R) Brain Reads, Liver Reads, Muscle Reads (these are static)

<http://woldlab.caltech.edu/rnaseq/mm9Brain1.comb.fa.gz>

<http://woldlab.caltech.edu/rnaseq/mm9Brain2.comb.fa.gz>

Target Files:

(T) UCSC Mouse Transcripts (version archived, static)

<http://hgdownload.cse.ucsc.edu/goldenPath/mm9/bigZips/refMrna.fa.gz>

(X) Contaminants (dated releases, volatile)

ftp://ftp.ncbi.nih.gov/genbank/genomes/Bacteria/Escherichia_coli_K_12_substr_MG1655/U00096.fasta

External Software:

(B) Bowtie software (dated releases, volatile)

<http://sourceforge.net/projects/bowtie-bio/files/bowtie/>

Internal Script Repository:

Q1 [quality check of fasta file] :

- entry header constraints
- character constraints
- length constraints

command line: `./nt_fasta_check.pl <fasta file>`

example: `./nt_fasta_check.pl refMrna.fa`

Q2 [quality check of tab delimited file] :

- header constraints
- column constraints
- record count constraints

command line: `./table_check.pl <tab delimited file> <delimiter>`

example: `./table_check.pl mm9Brain1.aligned.bowtie "\t"`

Q3 [quality check of bowtie install] :

- run tests to check bowtie executables run properly

command line: `./bowtie_check.pl <bowtie installation dir>`

example: `./bowtie_check.pl bowtie-0.9.9.3/`

BI [Bowtie fasta indexing command] :

- run bowtie-build executable on fasta data from command line

command line: `bowtie-build <target fasta file> <output index prefix>`

example: `bowtie-build refMrna.fa refMrna`

BA [Bowtie short read alignment command] :

- run bowtie executable to align query file against target file , from command line

command line: `bowtie -f -unfa <unaligned reads output file> -p <# threads> <targets index> <short reads fasta file> <aligned reads output file>`

example: `bowtie -f --unfa mm9Brain1.unaligned.fa -p 4 refMrna mm9Brain1.comb.fa mm9Brain1.aligned.bowtie &`

EVAL [Bowtie output evaluator] :

- evaluate bowtie output for uniquely matched reads or multiply matched reads and split them

command line: `./eval_bowtie_output.pl <bowtie alignments> <unique reads> <multi reads>`

example: `./eval_bowtie_output.pl mm9Brain1.aligned.bowtie mm9Brain1.uniq.aligned.bowtie mm9Brain1.multi.aligned.bowtie &`

CDISTR [calculation of read distributions] :

will output tab delimited files of the read distributions per target
 command line: ./calculate_read_dist.pl <targets fasta> <unique bowtie> <count output> &
 example: ./calculate_read_dist.pl refMma.fa mm9Brain1.uniq.aligned.bowtie Brain1.counts &

Conceptual Workflow:

- (1) Download Reads and UCSC fasta format data and do quality checks

Download (R, T , X)

|

Q1 (R, T , X)

|

Archive (T , X)

- (2) Download Bowtie executable binaries release from repository and run tests and archive version

Download (B)

|

Q3 (B)

|

Archive

- (3) Bowtie Index processing of the target data fasta files in parallel:

I = parallel: BI (T , X)

- (4) Split (R) data into N chunks, and run these R_N queries with Bowtie against T targets in parallel, and get output for D (reads aligned) and Z (unaligned reads):

D , Z = parallel : BA (R_N , I_T)

|

Join the alignment results files for each D :

For each D_N : A = join (D)

|

Q2 (A)

Run unaligned reads Z against Contaminants X:

For each Z_N : BA (Z , I_X)

|

Join the unaligned reads files for each Z :

For each Z_N : Y = join (Z)

|

Q2 (Y)

(5) Evaluate the alignment results, splitting data into unique- and multi- hit segments (U=unique, M=Multi):

```

For each A: EVAL ( A )
  /      |
  U      M
  |      |
  Q2     Q2

```

Checkpoint: check if $\text{Sum} (U , M , Z) = \text{total records from T}$

(6) Run calculation for each set of N uniquely aligned read distributions:
CDISTR (U_N)

(7) Plot the data for the uniquely aligned read distributions:
PLOT (U_N)

Example of how the workflow would work at the command line level with a smaller data set

Below I have listed the command line steps to execute the workflow, albeit without the splitting, joining, or archiving of data that we hope to have implemented in the generalized infrastructure. This is simply to illustrate how the scripts work and how the flow would be done manually. Ideally, the infrastructure would automatically handle the process of parallelization by splitting of the short reads data files, handling the execution of the scripts for each individual data subset, and joining back the output subsets into complete dataset when appropriate.

(1) Download Reads and UCSC fasta format data and do quality checks

```
wget 'http://woldlab.caltech.edu/rnaseq/mm9Brain1.comb.fa.gz'
```

```
wget 'http://woldlab.caltech.edu/rnaseq/mm9Brain2.comb.fa.gz'
```

```
wget 'http://hgdownload.cse.ucsc.edu/goldenPath/mm9/bigZips/refMrna.fa.gz'
```

```
wget
```

```
'ftp://ftp.ncbi.nih.gov/genbank/genomes/Bacteria/Escherichia_coli_K_12_substr_
MG1655/U00096.fna'
```

```
gunzip all files
```

```
./nt_fasta_check.pl U00096.fna
```

```
./nt_fasta_check.pl refMrna.fa
```

```
./nt_fasta_check.pl mm9Brain1.comb.fa
```

```
./nt_fasta_check.pl mm9Brain2.comb.fa
```

(2) Download Bowtie, install and check installation (here we assume you installed to current directory)

```
./bowtie_check.pl bowtie-0.9.9.3/
```

(3) Bowtie Index processing of the target data fasta files

```
bowtie-0.9.9.3/bowtie-build refMrna.fa refMrna
bowtie-0.9.9.3/bowtie-build U00096.fna U00096
```

steps 1-3: data and software you can also copy from */bi/ala/scratch/plabaj/Workflow_data*

(4) Run reads with Bowtie against targets (ideally in parallel - “-p 4” tells bowtie to run calculations in 4 threads, however we can also run many instances of bowtie {separate for: Brain1, Brain2, etc.}), and get output for reads aligned and unaligned reads. Then Run unaligned reads against the contaminant target sequence:

```
bowtie-0.9.9.3/bowtie -f --unfa mm9Brain1.unaligned.fa -p 4 refMrna mm9Brain1.comb.fa
mm9Brain1.aligned.bowtie &
bowtie-0.9.9.3/bowtie -f --unfa mm9Brain2.unaligned.fa -p 4 refMrna mm9Brain2.comb.fa
mm9Brain2.aligned.bowtie &
```

```
./table_check.pl mm9Brain1.aligned.bowtie "\t"
./table_check.pl mm9Brain2.aligned.bowtie "\t"
```

```
bowtie-0.9.9.3/bowtie -f -p 4 U00096 mm9Brain1.unaligned.fa mm9Brain1.unaligned.to.ecoli.bowtie
&
bowtie-0.9.9.3/bowtie -f -p 4 U00096 mm9Brain2.unaligned.fa mm9Brain2.unaligned.to.ecoli.bowtie
&
```

```
./table_check.pl mm9Brain1.unaligned.to.ecoli.bowtie "\t"
./table_check.pl mm9Brain2.unaligned.to.ecoli.bowtie "\t"
```

(5) Evaluate the alignment results, splitting data into unique- and multi- hit segments

```
./eval_bowtie_output.pl mm9Brain1.aligned.bowtie mm9Brain1.uniq.aligned.bowtie
mm9Brain1.multi.aligned.bowtie &
```

```
./eval_bowtie_output.pl mm9Brain2.aligned.bowtie mm9Brain2.uniq.aligned.bowtie
mm9Brain2.multi.aligned.bowtie &
```

(6) Run calculation of read distributions for each set of N uniquely aligned read distributions

```
./calculate_read_dist.pl refMrna.fa mm9Brain1.uniq.aligned.bowtie Brain1.counts &
./calculate_read_dist.pl refMrna.fa mm9Brain2.uniq.aligned.bowtie Brain2.counts &
```

Anhang B

Umsetzung des Next Generation Sequencing Analysis Workflow

Das nachfolgende Listing stellt die vollständige Umsetzung des in Anhang A textuell beschriebenen Arbeitsablaufs mithilfe der in dieser Arbeit vorgestellten Applikation dar.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <process name="sampleworkflow">
3    <task name="FastaCheck">
4      <description>Checking if fasta files are correct</description>
5      <type>iterateParallelTask</type>
6      <result>
7        <type>screenResult</type>
8      </result>
9      <fork>./ nt_fasta_check.pl</fork>
10     <parameterValues>../ Workflow_data/U00096.fna;../ Workflow_data/
        refMrna.fa;../ Workflow_data/mm9Brain1.comb.fa 1-90 10-1e9;../
        Workflow_data/mm9Brain2.comb.fa 1-90 10-1e9</parameterValues>
11     <antecessor>BowTieCheck</antecessor>
12   </task>
13   <task name="BowTieCheck">
14     <type>execTask</type>
15     <result>
16       <type>screenResult</type>
17     </result>
18     <fork>./ bowtie_check.pl ../ Workflow_data/bowtie-0.9.9.3/</fork>
19     <antecessor>BowTieIndex</antecessor>
20   </task>
```

```

21 <task name="BowTieIndex">
22   <description>Bowtie Index processing of the target data fasta
      files</description>
23   <type>iterateParallelTask</type>
24   <result>
25     <type>screenResult</type>
26   </result>
27   <fork>../ Workflow_data/bowtie-0.9.9.3/bowtie-build</fork>
28   <parameterValues>../ Workflow_data/U00096.fna ../ Workflow_data/
      U00096;../ Workflow_data/refMrna.fa ../ Workflow_data/refMrna</
      parameterValues>
29   <antecessor>BowTie_1</antecessor>
30 </task>
31 <task name="BowTie_1">
32   <description>Run reads with Bowtie against targets</description>
33   <type>iterateParallelTask</type>
34   <result>
35     <type>screenResult</type>
36   </result>
37   <fork>../ Workflow_data/bowtie-0.9.9.3/bowtie -f -p 2 --unfa </fork
      >
38   <parameterValues>../ Workflow_data/mm9Brain1.unaligned.fa ../
      Workflow_data/refMrna ../ Workflow_data/mm9Brain1.comb.fa ../
      Workflow_data/mm9Brain1.aligned.bowtie;../ Workflow_data/
      mm9Brain2.unaligned.fa ../ Workflow_data/refMrna ../
      Workflow_data/mm9Brain2.comb.fa ../ Workflow_data/mm9Brain2.
      aligned.bowtie</parameterValues>
39   <antecessor>TableCheck_1</antecessor>
40 </task>
41 <task name="TableCheck_1">
42   <description>Check BowTie results</description>
43   <type>iterateParallelTask</type>
44   <result>
45     <type>screenResult</type>
46   </result>
47   <fork>./ table_check.pl </fork>
48   <parameterValues>../ Workflow_data/mm9Brain1.aligned.bowtie "\t"
      ;../ Workflow_data/mm9Brain2.aligned.bowtie "\t" </
      parameterValues>
49   <antecessor>BowTie_2</antecessor>
50 </task>
51 <task name="BowTie_2">
52   <description>Run unalign reads with Bowtie against contaminant
      targets</description>

```

```

53     <type>iterateParallelTask</type>
54     <result>
55         <type>screenResult</type>
56     </result>
57     <fork>/bi/ala/scratch/plabaj/Workflow_data/bowtie-0.9.9.3/bowtie -
        f -p 2 ../Workflow_data/U00096 </fork>
58     <parameterValues>../Workflow_data/mm9Brain1.unaligned.fa ../
        Workflow_data/mm9Brain1.unaligned.to.ecoli.bowtie; ../
        Workflow_data/mm9Brain2.unaligned.fa ../Workflow_data/
        mm9Brain2.unaligned.to.ecoli.bowtie</parameterValues>
59     <antecessor>TableCheck_2</antecessor>
60 </task>
61 <task name="TableCheck_2">
62     <description>Check BowTie results</description>
63     <type>iterateParallelTask</type>
64     <result>
65         <type>screenResult</type>
66     </result>
67     <fork>./table_check.pl </fork>
68     <parameterValues>../Workflow_data/mm9Brain1.unaligned.to.ecoli.
        bowtie "\t" ;../Workflow_data/mm9Brain2.unaligned.to.ecoli.
        bowtie "\t" </parameterValues>
69     <antecessor>EvalRes</antecessor>
70 </task>
71 <task name="EvalRes">
72     <description>Evaluate the alignment results , splitting data into
        unique- and multi- hit segments</description>
73     <type>iterateParallelTask</type>
74     <result>
75         <type>screenResult</type>
76     </result>
77     <fork>./eval-bowtie-output.pl</fork>
78     <parameterValues>../Workflow_data/mm9Brain1.aligned.bowtie;
79         ../Workflow_data/mm9Brain1.uniq.aligned.bowtie;
80         ../Workflow_data/mm9Brain1.multi.aligned.bowtie;
81         ../Workflow_data/mm9Brain2.aligned.bowtie;
82         ../Workflow_data/mm9Brain2.uniq.aligned.bowtie;
83         ../Workflow_data/mm9Brain2.multi.aligned.bowtie
84     </parameterValues>
85     <antecessor>ReadCalc</antecessor>
86 </task>
87 <task name="ReadCalc">
88     <description>Run calculation of read distributions for each set of
        N uniquely aligned read distributions</description>

```

```

89     <type>iterateParallelTask</type>
90     <result>
91         <type>screenResult</type>
92     </result>
93     <fork>./ calculate_read_dist.pl ../ Workflow_data/refMrna.fa </fork>
94     <parameterValues>../ Workflow_data/mm9Brain1.uniq.aligned.bowtie
        ../ Workflow_data/Brain1.counts;../ Workflow_data/mm9Brain2.uniq
        .aligned.bowtie ../ Workflow_data/Brain2.counts</
        parameterValues>
95     <antecessor></antecessor>
96 </task>
97 </process>

```

Literatur

- [Brayner 96] A. Brayner, M. Weske. Einsatz von FlowMark in der Molekularbiologie. Tagungsband: EMISA Forum, Band 6/2, Seiten 14–21, 1996.
- [Community 06] JBoss Community. Chapter 1. introduction. Online-Quelle. <http://docs.jboss.com/jbpm/v3.2/userguide/html/ch01.html>, 24. Juli 2010.
- [Foundation 09a] The Apache Software Foundation. Apache log4j 1.2. Online-Quelle. <http://logging.apache.org/log4j/index.html>, 28. Mai 2010.
- [Foundation 09b] The Apache Software Foundation. Jakarta commons exec. Online-Quelle. <http://commons.apache.org/exec/>, 28. Mai 2010.
- [Foundation 10a] The Apache Software Foundation. Apache ant - welcome. Online-Quelle. <http://ant.apache.org/>, 03. Juni 2010.
- [Foundation 10b] The Apache Software Foundation. Commons cli. Online-Quelle. <http://commons.apache.org/cli/>, 28. Mai 2010.
- [Foundation 10c] The Eclipse Foundation. Eclipse.org home. Online-Quelle. <http://www.eclipse.org/>, 03. Juni 2010.
- [Fowler 04] Martin Fowler. Inversion of control containers and the dependency injection pattern. Online-Quelle. <http://martinfowler.com/articles/injection.html>, 04. Juni 2010.
- [Fox 02] Geoffrey C. Fox, Dennis Gannon, Mary Thomas. A summary of grid computing environments. *Concurrency & Computation: Practice & Experience*, 14(13-15):1035–1044, 11/2002 2002.
- [Gadatsch 08] Andreas Gadatsch. Grundkurs Geschäftsprozess-management: Methoden und Werkzeuge für die IT-praxis: Eine Einführung für Studenten und Praktiker. Springer, 2008.

- [Gehring 98] H. Gehring. Betriebliche Anwendungssysteme. Prozessorientierte Gestaltung von Informationssystemen, Hagen, 1998.
- [Gockel 08] Tilo Gockel. Form der wissenschaftlichen Ausarbeitung. Springer-Verlag, Heidelberg, 2008. Begleitende Materialien unter <http://www.formbuch.de>.
- [Gulcu 03] Ceki Gulcu. The complete log4j manual. QOS. ch, 2003.
- [Hollingsworth 95] D. Hollingsworth. Workflow management coalition. the workflow reference model. document number tc00-1003. issue 1.1. jan-95.
- [hsqldb Development Group 10] The hsqldb Development Group. Hsqldb. Online-Quelle. <http://hsqldb.org/>, 28. Mai 2010.
- [Hull 06] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole Goble, Mathew R. Pocock, Peter Li, Tom Oinn. Taverna: a tool for building and running workflows of services. Nucl. Acids Res., 34(suppl₂):W729–732, 2006.
- [jdom.org 09] jdom.org. The jdom project. Online-Quelle. <http://jdom.org/>, 28. Mai 2010.
- [Kreil 01] David Philip Kreil. From General Scientific Workflows to Specific Sequence Analysis Applications: The study of compositionally biased proteins. Dissertation, University of Cambridge, 2001.
- [McLaughlin 06] Brett McLaughlin, Justin Edelson. Java & XML. O'Reilly Media, Inc., 2006.
- [O'Brien 05] Timothy M. O'Brien. Jakarta commons cookbook. O'Reilly Media, Inc., 2005.
- [Oinn 04] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, Peter Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics, 20(17):3045–3054, 2004.
- [OSJava 08a] OSJava. Configuring simple-jndi. Online-Quelle. <http://www.osjava.org/simple-jndi/manual/Configuring.html>, 28. Mai 2010.
- [OSJava 08b] OSJava. Data files simple-jndi. Online-Quelle. <http://www.osjava.org/simple-jndi/manual/DataFiles.html>, 28. Mai 2010.

- [Paolo Missier 10] Stuart Owen Paolo Missier, Stian Soiland-Reyes. Taverna, reloaded. Tagungsband: Scientific and Statistical Database Management, Band 6187 of Lecture Notes in Computer Science, Seiten 471–481. Springer Berlin / Heidelberg, 2010.
- [Rosenkranz 06] Friedrich Rosenkranz. Geschäftsprozesse. Springer-Verlag Berlin Heidelberg, 2006.
- [Scheer 02] August-Wilhelm Scheer. ARIS-vom Geschäftsprozess zum Anwendungssystem. Springer-Verlag Berlin Heidelberg, 2002.
- [Software 09] TMate Software. Svnkit :: Subversion for java. Online-Quelle. <http://www.svnkit.com/index.html>, 28. Mai 2010.
- [Staud 06] J. Staud. Geschäftsprozessanalyse. Springer-Verlag Berlin Heidelberg, 2006.
- [Taylor 03] Ian Taylor, Matthew Shields, Ian Wang, Omer Rana. Triana applications within grid computing and peer to peer environments. Journal of Grid Computing, 1(2):199–217, 06 2003.
- [Taylor 04] Ian Taylor, Matthew Shields, Ian Wang. Resource management for the triana peer-to-peer services. Seiten 451–462, 2004.
- [Ullenboom 03] Christian Ullenboom. Java ist auch eine Insel. Galileo Press, 2003.
- [van Der Aalst 03] W.M.P. van Der Aalst, A.H.M. Ter Hofstede, B. Kiepuszewski, A.P. Barros. Workflow patterns. Distributed and parallel databases, 14(1):5–51, 2003.
- [van der Aalst 04] W.M.P. van der Aalst, L. Aldred, M. Dumas, A.H.M. ter Hofstede. Design and implementation of the YAWL system. Tagungsband: Advanced Information Systems Engineering, Seiten 281–305. Springer, 2004.
- [Van Der Aalst 05] W.M.P. Van Der Aalst, A.H.M. Ter Hofstede. YAWL: yet another workflow language. Information Systems, 30(4):245–275, 2005.
- [Walls 07] Craig Walls, Ryan Breidenbach. Spring in action. Manning Publications Co., Greenwich, CT, USA, 2007.
- [Wohed 09] P. Wohed, N. Russell, A.H.M. ter Hofstede, B. Andersson, W.M.P. van der Aalst. Patterns-based evaluation of open source BPM systems: The cases of jBPM, OpenWFE, and Enhydra Shark. Information and Software Technology, 51(8):1187–1216, 2009.

[www.springsource.org 10] www.springsource.org. Springsource.org. Online-Quelle.
<http://www.springsource.org/>, 28. Mai 2010.

[Yan 08] Zheng Yan. Designing and implementing a system for automating the java project analysis process. Diplomarbeit, V xj  University, School of Mathematics and Systems Engineering, 2008.

