Entwicklung einer browserbasierten Benutzerschnittstelle zur Erstellung von PACT-Programmen

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science (M. Sc.)

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

eingereicht von:	Mathias Nitzsche
geboren am:	9. April 1983
in:	Berlin
Gutachter:	Prof. Johann-Christoph Freytag, PhD
	Prof. Dr. sc. nat. Klaus Bothe
eingereicht am:	31. Mai 2012
verteidigt am:	

People think that computer science is the art of geniuses but the actual reality is the opposite, just many people doing things that build on each other, like a wall of mini stones.

Donald Knuth
(Turing Award Winner, 1974)

Zusammenfassung

„Entwicklung einer browserbasierten Benutzerschnittstelle zur Erstellung von PACT-Programmen"

Innerhalb des Stratosphere-Projekts werden PACT-Programme genutzt, um große Datenmengen verteilt und hochperformant von vielen Computern gleichzeitig verarbeiten zu lassen. Die Erstellung von PACT-Programmen ist bisher jedoch sehr zeitaufwändig, kompliziert und erfordert die Einrichtung einer eigenen Entwicklungsumgebung. Um diesen Prozess zu vereinfachen, wurde im Rahmen der vorliegenden Masterarbeit die PACT-GUI-Webanwendung prototypisch entwickelt, die es erstmals ermöglicht, PACT-Programme ohne Zusatzsoftware graphisch im Browser zu modellieren. Die komplett Java-basierte Browseranwendung wurde in einem evolutionären Vorgehen auf Basis des neuartigen Applikationsframeworks Google Web Toolkit unter Verwendung bewährter Konzepte und Werkzeuge der Softwareentwicklung implementiert. Damit bestätigt die PACT-GUI die konzeptionelle und technische Umsetzbarkeit der visuellen Modellierung von Datenflussgraphen im Browser. Ferner eröffnet sie neuen Benutzerkreisen die Möglichkeit PACT-Prorgamme zu erstellen, was langfristig zur weiteren Verbreitung des Stratosphere-Systems beitragen wird.

Abstract

"Implementation of a browser based user interface to create PACT-programs"

Within the Stratosphere Project PACT-programs are used to process large amounts of data in parallel on a distributed cluster computer architecture. However creating a PACT-program is still very time-consuming, complicated, and requires the installation of a development environment. This master thesis describes the implementation of the PACT-GUI web application, which will simplify this process by allowing to graphically model PACT-Programs without the need for any additional software. The fully Java-based browser application was created in an iterative development process. Well-known software development tools and concepts as well as the new application framework Google Web Toolkit were used. The PACT-GUI confirms the conceptional and technical feasibility of a browser based tool to graphically model data flows. In future, the ease with which users can now generate PACT-programs may promote the wide adoption of Stratosphere System.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Diese Arbeit ist Teil des Stratosphere-Projekts [Str10], einer gemeinsamen Forschungsinitiative der Technischen Universität Berlin, der Humboldt-Universität zu Berlin und des Hasso-Plattner-Instituts Potsdam. Gemeinsam verfolgen die Forscher das Ziel, ein neuartiges, datenbankinspiriertes System[1] zu entwerfen, um sehr große Datenmengen zu analysieren und zu verarbeiten. Die vorliegende Masterarbeit dokumentiert den Erstellungsprozess eines Softwarewerkzeugs namens PACT-GUI[1] – einer Webbrowser[1]-basierten Benutzungsschnittstelle zur Erstellung von PACT-Programmen[1].

1.1 Problemstellung

Es existiert bereits eine Vielzahl von Systemen, um große Datenmengen möglichst effizient zu verarbeiten. Dazu zählen Datenbanksysteme[1] und Programmiermodelle wie Map/Reduce [DG04]. Erstere sind starr an ein vorher definiertes Schema gebunden und skalieren mit sehr großen Datenmengen nur bedingt auf verteilten Architekturen[1] von mehreren hundert oder tausend Rechnern. Letztere skalieren in solchen Umgebungen sehr gut, sind jedoch in ihrer Ausdruckskraft im Hinblick auf komplexe Datenbankoperationen teilweise begrenzt und daher nur umständlich einsetzbar [BEH+10].

Im Rahmen des Stratosphere-Projekts soll ein Mittelweg geschaffen werden, der es einerseits erlaubt, schemafreie Daten durch komplexe datenbankähnliche Operationen zu bearbeiten, und anderseits die Leistung einer Vielzahl von Rechnern fehlertolerant und transparent nutzt. Um dieses Vorhaben zu verwirklichen, wurden ein neues Programmiermodell namens *Parallelization Contracts* (kurz PACTs) [BEH+10] und die dazugehörige Ausführungsumgebung *Nephele* [WK09] entworfen. Beide Techniken sind bereits prototypisch implementiert und erlauben mit Hilfe von PACT-Programmen Datenverarbeitungsaufgaben verteilt auszuführen.

PACT-Programme sind Datenflussgraphen[1]. Sie bestehen aus PACTs, welche die Datenverarbeitung realisieren, und gerichteten Kanten, welche den Datenfluss abbilden. PACT-Programme können derzeit jedoch nicht als Graph modelliert werden, sondern müssen mit

[1] siehe Glossar

Hilfe der Programmiersprache Java[1] in einer Java-Entwicklungsumgebung programmiert werden. Dies ist nicht nur kompliziert, zeitaufwendig und fehleranfällig, sondern erfordert zudem ein hohes Maß an Kenntnis von Interna des Stratosphere-Projekts. Daher ist es in der aktuellen Form nur einem sehr begrenzten Personenkreis möglich PACT-Programme zu erstellen, wodurch die breite Nutzung des Stratosphere-Systems erschwert wird.

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist es, einen Lösungsansatz zu erarbeiten, um den zeitaufwändigen und komplizierten Prozess der Erstellung von PACT-Programmen zu vereinfachen. Es ist ein Werkzeug zu schaffen, welches alle Benutzer des Stratosphere-Systems dazu befähigt, PACT-Programme möglichst einfach zu erstellen. Die Anforderungen an ein solches Werkzeug werden diskutiert und eine Softwarelösung prototypisch implementiert. Um die bisher nötige lokale Softwareinstallation überflüssig zu machen, soll dies mittels einer intuitiv zu bedienenden Benutzungsoberfläche innerhalb des Webbrowsers möglich sein. Da sich aber auch das Stratosphere-Gesamtsystem noch in der Entwicklung befindet, muss die Anwendung über die rein funktionalen Anforderungen hinaus so konzipiert werden, dass sie spätere Weiterentwicklungen und Anpassungen problemlos erlaubt. Diese Arbeit wird dabei weniger auf die Details der Implementierung eingehen, sondern vor allem konzeptionelle Grundlagen und die Vorgehensweise zur Problemlösung beschreiben. Ein weiterer Schwerpunkt der Arbeit liegt auf der Begründung und späteren Evaluation getroffene Entwurfsentscheidungen, um die Überführbarkeit der Ergebnisse auf ähnliche Modellierungsaufgaben zu gewährleisten.

1.3 Aufbau der Arbeit

Im Anschluss an die Einleitung legt Kapitel 2 die Grundlagen zum Verständnis dieser Arbeit. Es werden das Map/Reduce-Modell, das PACT-Programmiermodell, die Technologien und Charakteristika einer verteilten Webanwendung sowie eine Auswahl von Vorgehensmodellen der Softwareentwicklung beschrieben.

Kapitel 3 erläutert im Detail den derzeitigen Prozess der Erstellung von PACT-Programmen sowie dessen Nachteile. Auf dieser Basis wird ein graphisches Modellierungswerkzeug als Lösungsansatz motiviert.

Kapitel 4 beschreibt die verschiedenen Phasen des Entstehungsprozesses des Modellierungswerkzeugs von der Planung und der Analyse über den Entwurf und die Implementierung bis hin zum Test.

Im Kapitel 5, dem Evaluationsteil, werden das gewählte Vorgehen, die getroffenen Entscheidungen sowie die verwendeten Modelle und Werkzeuge bewertet. Des Weiteren werden bestehende Probleme und Möglichkeiten der Weiterentwicklung der Applikation aufgezeigt.

Den Abschluss bildet Kapitel 6 mit einem Fazit der Masterarbeit.

Kapitel 2

Grundlagen

Zum Verständnis dieser Masterarbeit ist die Kenntnis einer Vielzahl von Abkürzungen, Begriffen und Konzepten erforderlich. Alle verwendeten Begriffe, deren Kenntnis nicht vorausgesetzt werden kann, werden bei der erstmaligen Erwähnung erklärt oder gegebenenfalls auf eine Erklärung im Glossar beziehungsweise auf weiterführender Literatur verwiesen. Das Glossar stellt zudem eine Referenz zum späteren Nachschlagen von Begriffsdefinitionen dar. In diesem Kapitel werden die folgenden Themenbereiche vertiefend eingeführt:

2.1 Map/Reduce-Programmiermodell[1]: Es werden die Entstehung, die Arbeitsweise, vorhandene Implementationen und die Grenzen der Anwendbarkeit des Map/ Reduce-Programmiermodells beschrieben.

2.2 Stratosphere-Projekt: Um die Möglichkeiten des Map/Reduce-Modells zu erweitern, entstand das Stratosphere-Projekt. In diesem Unterkapitel werden das PACT-Programmiermodell und die Ausführungsumgebung Nephele als Teile des Projekts sowie verwandte Systeme betrachtet.

2.3 Webanwendungen: Die prototypische Entwicklung der PACT-GUI-Webanwendung ist Teil dieser Masterarbeit. Aus diesem Grund werden die Basiskonzepte und Charakteristika einer Webanwendung vorgestellt.

2.4 Vorgehensmodelle der Softwareentwicklung: Um die Vorgehensweise bei der Anwendungsentwicklung zu begründen, bildet eine Übersicht vorhandener Vorgehensmodelle der Softwareentwicklung den Abschluss dieses Grundlagenkapitels.

2.1 Map/Reduce-Programmiermodell

Das Map/Reduce-Programmiermodell[1] dient zur verteilten Verarbeitung großer Datenmengen. Es stellt die konzeptionelle Basis für PACT-Programme (2.2.1) dar, weshalb es in diesem Unterkapitel im Detail vorgestellt wird.

[1] Eine allgemeine Definition des Begriffs *Modell* findet sich im Glossar

Im ersten Abschnitt wird der Begriff der Datenverarbeitung eingeführt und die Entwicklung von Map/Reduce motiviert. Anschließend werden die Funktionsweise von Map/Reduce sowie vorhandene Ausführungsumgebungen und weiterführende Konzepte vorgestellt. Am Ende dieses Unterkapitels werden die Anwendungsbeispiele und Grenzen des Modells beschrieben, welche zur Entwicklung von Stratosphere geführt haben.

2.1.1 Einführung

Datenverarbeitung bezeichnet den organisierten Umgang mit Datenmengen, um sie zu analysieren oder zu verändern. Im Rahmen der Volkszählung 1890 in den USA wurden erstmalig Daten mechanisch mittels Lochkarten verarbeitet. Seit Mitte des 20. Jahrhunderts hat die elektronische Datenverarbeitung durch den Computer Einzug in alle Bereiche des Lebens gehalten und so die moderne Informationsgesellschaft begründet [FSV05, S. 255]. Über Jahrzehnte hinweg kam dabei Datenbanksystemen[2] die zentrale Rolle bei der Speicherung, Manipulation und Verwaltung großer Datenmengen zu. Durch Datenbankschemata sind die Daten strikt organisiert und mittels Abfragesprachen, wie SQL[2], einfach zugänglich. Transaktionsverwaltungsmechanismen sichern dabei die Datenintegrität auch bei mehreren gleichzeitigen Anfragen.

Aktuelle Anwendungsszenarien der Verarbeitung großer Datenmengen sind beispielsweise Suchmaschinenindexe, Webapplikationen (2.3.1) mit Millionen von Nutzern oder wissenschaftliche Anwendungen in der Klima- oder Genforschung. Die Größenordnung der hier zu verarbeitenden Datenmengen geht in den Bereich mehrerer Tera- oder Petabytes[3]. So hat der Softwarekonzern Google Inc. für die Bereitstellung seiner Dienste im Jahr 2008 täglich mehr als 20 Petabyte Daten innerhalb seiner Rechenzentren verarbeitet [DG08].

Klassische Datenbanksysteme sind für diese Anwendungsszenarien (zusammengefasst unter dem Begriff *Web Scale Data Management* [BAC+09]) vor allem aus den folgenden zwei Gründen nicht oder nur begrenzt geeignet [LLC+12, BEH+10]:

- Zum einen ist die zur Verfügung stehende Hardware zu nennen. Aus Kostengründen wird wenigen leistungsstarken, aber sehr teuren Servern meist eine sehr große Zahl einfacher und günstiger Standardcomputer vorgezogen. Datenbanksysteme sind jedoch nur bedingt fehlertolerant und elastisch genug, um auf solchen Rechnerarchitekturen mit hunderten fehleranfälligen Einzelknoten effizient zu skalieren.

- Zum anderen setzen Datenbanken ein striktes Schema voraus. Sie sind somit nicht optimal geeignet für textuelle oder halbstrukturierte Daten, wie sie in den beschriebenen Anwendungsfällen oft vorkommen.

Um große, schemafreie Datenmengen hochskalierbar auf einer großen verteilten Rechnerarchitektur verarbeiten zu können, wurde das Map/Reduce-Programmiermodell[1] [DG04] entwickelt. Das Modell schafft eine Problemabstraktion, die Funktionalität strikt von der

[2] siehe Glossar

[3] Ein Petabyte entspricht 10^{15} Byte; oder 1024 Terabyte; oder rund 1 Mio. Gigabyte

Parallelisierung trennt, wodurch sich der Programmierer ausschließlich auf die Implementierung der Datenverarbeitungslogik konzentrieren kann. Die Details der parallelen Ausführung, wie die Datenverteilung, die Fehlererholung und die Lastverteilung, werden transparent für den Endnutzer durch das jeweilige Map/Reduce-Framework (2.1.3) realisiert [DG08].

Dabei ist Map/Reduce keine völlig neue Entwicklung, sondern laut der Entwickler: *„Inspired by the map and reduce primitives present in Lisp and many other functional languages*[4] *“* [DG04]. Rückblickend betrachtet scheint diese Anlehnung nur logisch, da bei der *Funktionalen Programmierung* Programmlogik so gekapselt ist, dass Seiteneffekte vermieden werden. Dadurch ist die Programmlogik ausschließlich abhängig von der aufrufenden Instanz und nicht von anderen Umgebungsparametern oder der Ausführungszeit (Referenzielle Transparenz). Dies macht die Bestandteile funktionaler Programme gut verifizierbar, testbar, optimierbar und vor allem auch parallelisierbar. [Rec06, S. 599ff] [Bra05, S. 171ff]

2.1.2 Arbeitsweise

Das Grundkonzept paralleler Datenverarbeitung und Voraussetzung für die Anwendung von Map/Reduce ist *Datenparallelität*. Diese ist gewährleistet, wenn Teile der Daten unabhängig vom Kontext der Gesamtdaten verarbeitet werden können. Solche Einzelteile der Daten werden Parallelisierungseinheiten (englisch: parallelization unit, PU) genannt. Die maximale Anzahl von PUs, in welche die Daten zerlegt werden können, entscheidet über den maximal möglichen Grad der Parallelisierung der Datenverarbeitung. [BEH+10]

Zur Verarbeitung von Daten mittels Map/Reduce müssen diese in PUs geteilt werden können und als Menge von Schlüssel/Wert-Paaren vorliegen. Der Schlüssel ist ein Bezeichner zur eindeutiger Identifikation des Paares. Eine PU kann beispielsweise ein Dokument in einer Menge von Dokumenten (Schlüssel=Dokumentname; Wert=Dokumentinhalt) oder auch eine Zeile in einem Text (Schlüssel=Zeilennummer; Wert=Zeileninhalt) sein.

In seiner Grundform gliedert sich Map/Reduce entsprechend Abbildung 2.1 in die Phasen *Map* und *Reduce*, für die je eine Funktion vom Anwender definiert werden muss.

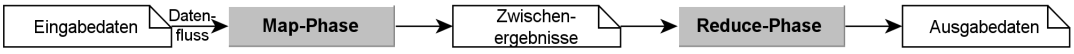


Abbildung 2.1: Vereinfachtes Grundschema von Map/Reduce

Map-Phase: Für jedes Schlüssel/Wert-Paar $(k1, v1)$ der Eingabedaten wird je einmal die anwenderdefinierte Map-Funktion aufgerufen. Die Map-Funktion generiert je Aufruf kein, ein oder mehrere neue Schlüssel/Wert-Paare $(k2, v2)$ als Zwischenergebnismenge[5]. Nach-

[4] deutsch: „Inspiriert von den Map- und Reduce-Funktionen aus Lisp und anderen funktionalen Sprachen“

[5] Die formelle Darstellung der Map- und der Reduce-Funktion ist der Originalquelle entnommen [DG04]. *list* ist jedoch keine (geordnete) Liste, sondern eine Menge.

dem alle Map-Aufrufe abgeschlossen sind, werden die Werte der Zwischenergebnismenge durch das jeweilige Map/Reduce-Framework nach Schlüsseln gruppiert. Jede Gruppe eines Schlüssels mit der Menge aller dazugehörigen Werte erzeugt einen Aufruf der Reduce-Funktion.

$$map(k1, v1) \rightarrow list(k2, v2)$$

Reduce-Phase: Die Reduce-Funktion wird mit einem Schlüssel $k2$ und einer dazugehörigen Menge[5] von Werten $list(v2)$ als Parameter aufgerufen. Das Ergebnis jedes dieser Funktionsaufrufe ist eine meist kleinere, oft nur einen oder keinen Wert umfassende Menge $list(v3)$. Dabei entspricht der Typ von $v3$ dem von $v2$, jedoch nicht unbedingt dem von $v1$. Um auch Datenmengen verarbeiten zu können, die größer sind als der vorhandene Arbeitsspeicher, werden die Daten üblicherweise nicht direkt an die Reduce-Funktion übergeben, sondern ein Zeiger oder Iterator.

$$reduce(k2, list(v2)) \rightarrow list(v3)$$

```
// Eingabe: "Bier Brauer Bauer braut braunes Bier"
            "braunes Bier braut Bier Brauer Baua"
map(String key, String value):
        // key: Zeilennummer
        // value: Inhalt der Zeile
        for each word w in value:
                ZwischenergebnisAusgeben(w, 1);

// kombinierte und sortierte Zwischenergebnisliste der Map-Aufrufe: [("Baua
    ",1),("Bauer",1),("Bier",1),("Bier",1),("Bier",1),("Bier",1),("Brauer",1)
    ,("Brauer",1),("braunes",1),("braunes",1),("braut",1),("braut",1)]
reduce(String key, Iterator values):
        // key: Ein bestimmtes Wort des Textes wie "Bier"
        // values: Eine Liste von Werten (1, 1, 1, 1)
        int wortAnzahl = 0;
        for each value in values:
                wortAnzahl += parseInt(value);
        ErgebnisAusgeben(asString(wortAnzahl));

// Ausgabe: Baua 1, Bauer 1, Bier 4, Brauer 2, braunes 2, braut 2
```

Abbildung 2.2: Map/Reduce: Pseudo-Code des Wordcount-Beispiels

Die Arbeitsweise von Map/Reduce lässt sich anhand des *Wordcount-Beispiels* [DG08] verdeutlichen. Gegeben sei ein Text, in dem Worthäufigkeiten gezählt werden sollen. Der Nutzer schreibt eine Map-Funktion, die, auf jede Zeile des Textes einzeln angewendet, eine „1" für jedes Vorkommen eines Wortes ausgibt. Die Reduce-Funktion wird anschließend je einmal für jedes der gefundenen Wörter aufgerufen und gibt die summierte Anzahl des Auftretens des Wortes aus. Der Ablauf von Map/Reduce im Wordcount-Beispiel wird nochmals durch den Pseudo-Code in Programmbeispiel 2.2 sowie die graphische Darstellung in Abbildung 2.3 veranschaulicht.

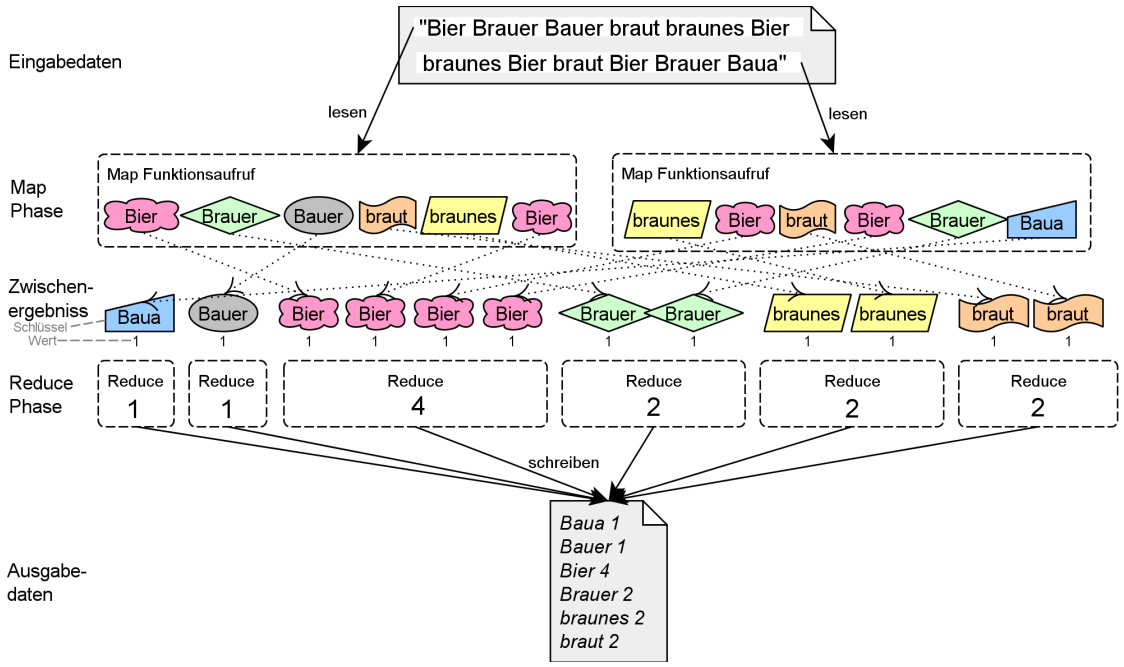


Abbildung 2.3: Map/Reduce: Graphische Darstellung des Wordcount-Beispiels

Um die Anzahl der Datensätze in der Zwischenergebnismenge und somit die Netzwerklast für die Übertragung zu verringern, ist es möglich, eine zusätzliche *Combine-Funktion* zu definieren. Diese wird bereits auf dem Rechenknoten der Map-Funktion ausgeführt und erlaubt eine lokale Voraggregation der Ergebnisse. Die im Pseudo-Code dargestellten Ergebnisse *(“Bier“,1),(“Bier“,1)* können so bereits im Anschluss an die Map-Funktion zu *(“Bier“,2)* reduziert werden.

2.1.3 Frameworks und höhere Programmiersprachen

Aufbauend auf dem Map/Reduce-Programmiermodell wurden sowohl im kommerziellen als auch im Open-Source[2] und wissenschaftlichen Bereich eine Vielzahl ähnlicher und weiterführender Systeme[2] entworfen. Dieser Abschnitt gibt einen allgemeinen Überblick über die verschiedenen Ansätze – für eine detaillierte Einführung sei auf die referenzierte Literatur verwiesen.

Map/Reduce-Frameworks

Als Map/Reduce-Framework[2] wird eine Laufzeitumgebung bezeichnet, auf deren Basis Datenverarbeitungsaufgaben nach dem Map/Reduce-Modell ausgeführt werden können. Die Implementierung des Map/Reduce-Modells hängt stark von der Zielumgebung ab, die aus einem einzelnen Mehrkernrechner oder auch einem großen verteilten Computernetzwerk bestehen kann [DG08]. Unter anderem existieren die folgenden:

- **Googles Map/Reduce-Framework:** Anfang 2003 wurden von Google Inc. das

Map/Reduce-Programmiermodell und die erste Version des eigenen Map/Reduce-Frameworks entwickelt [DG04]. Im Jahr darauf haben die Google-Mitarbeiter Dean und Ghemawat das Modell veröffentlicht, wobei das Framework selbst nach wie vor nur firmenintern verfügbar ist [DG04]. 2010 bekam Google für das Map/Reduce-Modell ein Patent zugesprochen [DG10], wird die freie Verwendung des Modells jedoch nicht einschränken [Met10]. Zur Ausführung eines Map/Reduce-Programms steuert ein Master-Rechenknoten die Worker-Rechenknoten, auf denen die Map- und Reduce-Aufrufe stattfinden. Die Datenhaltung geschieht im Google-eigenen, verteilten Dateisystem GoogleFileSystem [GGL03].

- **Apache Hadoop:** Hadoop [The11a] ist ein Projekt zur gemeinschaftlichen Entwicklung eines quelloffenen[6], frei verfügbaren Frameworks[2] zur Unterstützung datenintensiver, verteilter Rechenaufgaben entsprechend dem Map/Reduce-Modell. Zentrale Bestandteile sind das verteilte, hoch skalierbare Dateisystem HDFS[7] und das Hadoop MapReduce-Framework. Hadoop wurde in Java[2] entwickelt und wird weltweit in vielen freien Softwareprojekten und von Firmen wie Adobe, Amazon, eBay, Facebook, IBM, Twitter und Yahoo eingesetzt [The12d]. Weiterhin ist die nicht-relationale, verteilte Datenbank HBase [The12c], basierend auf dem Konzept der Google BigTable [CDG+08], Teil des Hadoop-Projekts.

- Neben Hadoop gibt es eine Reihe älterer Frameworks, welche das Map/Reduce-Modell auf andere Programmiersprachen oder Serverumgebungen portiert haben, wie MySpace Qizmt (für Windows Server) [MyS09], Disco (für Python und Erlang) [Dis11] und SkyNet (für Ruby) [Sky08].

Höhere Programmiersprache
Es gibt höhere Programmiersprachen, die auf Map/Reduce aufsetzen, um dessen Nutzung zu vereinfachen, indem sie die wiederholte Implementierung von Aufgaben durch Verlagerung in die Laufzeitumgebung vermeiden [PPR+09]:

- **Sawzall** [PDGQ05] ist eine von Google intern verwendete, domänenspezifische Programmiersprache zur Analyse von Log-Files. Sie hat eine einfache, deklarative Syntax, die intern in Map/Reduce-Programme übersetzt wird.

- **Cascading** [Con11] ist eine Java-Bibliothek und Anwendungsschnittstelle, welche Hadoops Map/Reduce für die Ausführung verteilter Datenverarbeitungsaufgaben aus allen JVM[8]-basierten Sprachen wie Java, Jython, JRuby, Groovy und Clojure kapselt.

- **Apache Pig** [GNC+09] ist ein von Yahoo Inc. entwickeltes Framework zur Verarbeitung großer Datenmengen mittels der Programmiersprache Pig Latin [ORS+08]. In dieser Sprache formierte Datenanalyseprogramme werden intern in eine Sequenz von Map/Reduce-Programmen übersetzt und auf dem Hadoop-System ausgeführt.

[6] siehe Glossar unter *Open-Source*
[7] HDFS = Hadoop Distributed File System
[8] JVM = Java Virtual Machine, siehe Glossar

- **Apache Hive** [TSJ+09] ist eine initial von Facebook entwickeltes, verteiltes Data Warehouse Framework. Das System ist in einigen Punkten Pig ähnlich [Geo09, Gat10], jedoch näher an relationale Datenbanksysteme angelehnt. Daten werden in Tabellen mit einem Schema gespeichert und mit HiveQL abgefragt. Die Sprache hat eine auf SQL[2]-basierende Syntax, was die Eingewöhnung für datenbankerfahrene Anwender erleichtert und durch eine JDBC/ODBC-Schnittstelle[2] die Integration mit anderen Anwendungen ermöglicht.
- **Jaql** [IBM11b] ist eine von IBM Research entwickelte lizenzfreie Abfragesprache für JSON[2]-Dokumente. Auch andere Dokumente lassen sich mit Jaql abfragen, wenn sie per JSON-View in dieses Format überführt werden können. In Jaql formulierte, deklarative Anfragen werden intern in Map/Reduce-Aufgaben überführt und mittels Hadoop parallel ausgeführt. Auch Jaql-Abfragen können mit benutzerspezifischem Code erweitert werden.

2.1.4 Anwendungsbeispiele und Grenzen

Trotz seiner konzeptuellen Einfachheit hat das Map/Reduce-Modell bewiesen, dass sich mit ihm vielfältige Probleme effizient lösen lassen [DG08]:

- **Verteilter Grep:** Alle Vorkommen in einer großen Datenmenge finden, die einem bestimmten Muster entsprechen.
- **Log-File-Auswertung:** z.B. Analyse, auf welche Webadressen am meisten zugegriffen wurde oder für welche Webadresse die meisten Fehler aufgetreten sind.
- **Graph-Analyse:** z.B. aus einer großen Menge von Webseiten Aussagen darüber treffen, welche Webseiten auf eine bestimmte andere Webseite verweisen.
- **Verteilte Sortierung:** Sortieren einer großen Datenmenge.

Weitere Anwendungsfelder erstrecken sich von Problemen des Maschinenlernens und Trendanalysen bis hin zur Auswertung von Satellitenbildern und automatischer Sprachübersetzung [DG08].

Es gibt jedoch auch einige Datenverarbeitungsaufgaben, für die Map/Reduce nicht otimal geeignet ist, was sich durch große Leistungseinbußen zeigt [PPR+09]. Eine umfassende Übersicht von problematischen Anwendungen und Lösungsansätzen ist in [LLC+12] aufgeführt. Einige Beispiele sind die folgenden:

Globale Zustände: Mit Map/Reduce können Algorithmen die globale Zustände erfordern, nur schwer implementiert werden. Ein Beispiel ist die wiederholte Verarbeitung der gleichen Eingangsdaten mittels einer Schleife, wie sie oft bei der Graphenanalyse vorkommt. Um die Daten nicht wiederholt einlesen zu müssen, ergänzt unter anderem HaLoop [BHBE10] das Hadoop-Framework (2.1.3) und Twister [ELZ+10] ein eigenes Map/Reduce-Framework um eine Zwischenspeicherlösung.

Echtzeitanalyse: Map und Reduce sind blockierende Operatoren, da die Map- und Reduce-Phase jeweils erst vollständig abgeschlossen sein muss, bevor die nächste Phase oder weitere Berechnung durchgeführt werden können. Es werden daher verschiedene Methoden unter Verwendung von Datenstreams vorgeschlagen, um Echtzeitanalysen von Daten innerhalb von *sliding windows* durchführen zu können [LLC+12].

Join: Das Map/Reduce-Modell arbeitet nur auf einem Quelldatensatz. Das Zusammenführen von Schlüssel/Wert-Paaren aus zwei verschieden Quellen anhand gleicher Schlüssel – aus der relationalen Algebra bekannt als *Join* – ist daher nur umständlich möglich. Um diese Operation dennoch mit Map/Reduce umzusetzen, gibt es verschiedene Vorgehensweisen [LLC+12]. Eine ist beispielsweise die Map-Phase zu nutzen, um eine Vereinigung der beiden Eingabedatensätze zu erzeugen, wobei die einzelnen Ausgabewerte so gekennzeichnet werden müssen, dass ihre Herkunft rekonstruierbar ist. In der Reduce-Phase werden dann alle zu einem Schlüssel gehörigen Werte wieder getrennt und durch einen zu wählenden Join-Algorithmus zusammengeführt [AEH+11].

Dieses umständliche und wenig intuitive Vorgehen ist ein klares Indiz dafür, dass Map/Reduce für das Zusammenführen zweier Datenquellen ungeeignet ist. Besonders problematisch ist, dass der Entwickler schon während der Programmierung Annahmen und Vorbereitungen bezüglich der späteren parallelen Laufzeitumgebung treffen muss. Dies widerspricht dem in der Map/Reduce-Einführung (2.1.1) beschriebenen Grundprinzip der Trennung von Funktionalität und paralleler Ausführung. Zudem versteckt dieses Vorgehen den Fakt, dass zwei verschiedene Datenquellen existieren, die auf Grund ihrer Größe oder vorhandenen Sortierung möglicherweise unterschiedlich behandelt werden sollten. Als Konsequenz ist es schwer, wenn nicht unmöglich, automatisiert Optimierungen vorzunehmen, da dies eine Änderung des Benutzercodes erfordern würde. [PPR+09, BEH+10]

Ein erster Ansatz zur nativen Unterstützung der Zusammenführung mehrerer Datenquellen ist MapReduce-Merge [YDHP07], welches das Map/Reduce-Modell um eine zusätzliche Merge-Phase erweitert. Das im nächsten Unterkapitel beschriebene Stratosphere-Projekt bieten einen generellen Ansatz zur effektive Verarbeitung mehrerer Datenquellen mittels eines neuen Programmiermodells, das weitere komplexe Operatoren ermöglicht.

2.2 Stratosphere-Projekt

Ziel des Stratosphere-Projekts ist es, ein System^2 zur Verarbeitung großer textueller oder semistrukturierter Daten zu schaffen, welches die Rechenleistung einer großen verteilten Rechnerarchitektur von bis zu mehreren tausend Rechnern parallel, fehlertolerant und transparent ausgenutzt. Stratosphere ist ein neuartiger, an Datenbanken^2 und das Map/Reduce-Modell (2.1) angelehnter Ansatz, der speziell die Verarbeitung mehrerer Datenquellen und die kostenbasierte Optimierung der Programmausführung ermöglicht. [Str10]

Wie in Abbildung 2.4 [BEH+10] dargestellt, bilden folgende drei in den nächsten Unterkapiteln behandelte Komponenten den Kern des Stratosphere-Systems:

2.2.1 Das PACT-Programmiermodell ist eine Verallgemeinerung des vorgestellten Map/Reduce-Modells (2.1). Es besteht aus einzelnen *Parallelization Contracts* (kurz PACTs), die mit den Map- und Reduce-Funktionen vergleichbar sind, jedoch weitere, deutlich komplexere Operationen zulassen.

2.2.2 Der PACT-Compiler überführt PACT-Programme in Nephele-Datenflussgraphen[2]. Bei der Übersetzung kann der PACT-Compiler kostenbasiert aus verschiedenen Ausführungsplänen den Optimalen wählen.

2.2.3 Nephele ist eine hoch skalierbare parallele Ausführungsumgebung für Nephele-Datenflussprogramme, die durch den PACT-Compiler aus PACT-Programmen erzeugt werden.

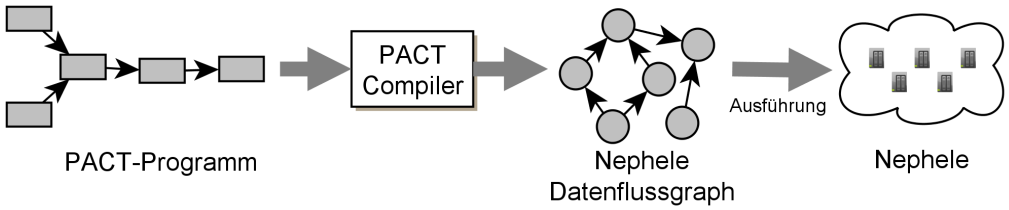


Abbildung 2.4: Kompilierung und Ausführung eines PACT-Programms

Das PACT-Programmiermodell und Nephele sind innerhalb von Stratosphere konzeptionell deutlich voneinander zu trennen. Während mit dem PACT-Programmiermodell implementiert wird, *was* getan werden soll, ist Nephele für das *wie* der parallelen Ausführung zuständig.

2.2.1 PACT-Programmiermodell

Das PACT-Programmiermodell ist eine Verallgemeinerung des Map/Reduce-Modells (2.1) zur verteilten Datenverarbeitung [BEH+10]. PACT-Programme (siehe Abbildung 2.5) sind dem PACT-Programmiermodell entsprechende *gerichtete, azyklische Datenflussgraphen* (DAG[9] - Erklärung auf Seite 12).

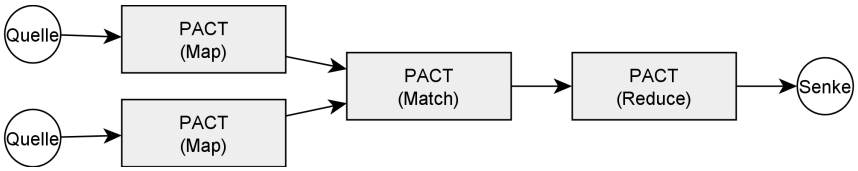


Abbildung 2.5: Ein Beispiel eines PACT-Programms

[9]DAG = Directed Acyclic Graph

Gerichteter, azyklischer Graph
Nach [EP02] besteht ein Graph G aus einer Menge von Knoten (englisch: vertex) V und eine Menge von Kanten (englisch: edges) E zwischen den Knoten. Die Elemente der Menge E sind 2-elementige Teilmengen von V. $G = (V, E)\ mit\ E \subseteq VxV$

Des Weiteren gilt:

- G heißt **ungerichtet**, wenn für alle Knoten v_1, v_2 aus V gilt: $(v_1, v_2) \in E \Rightarrow (v_2, v_1) \in E$. Ansonsten wird der Graph als **gerichtet** bezeichnet.
- Ein nicht-leeres Wort $W = v_1 \dots v_n$ heißt **Weg** (der Länge $n-1$) von v_1 nach v_n, wenn $\forall i < n$ gilt $(v_i, v_{i+1}) \in E$
- Wenn $(v_1, v_2) \in E$ ist, dann ist v_1 **Vater** von v_2 und v_2 **Sohn** von v_1.
- Wenn es einen Weg von $v1$ nach v_2 gibt, dann ist $v1$ **Vorfahre** von v_2 und v_2 **Nachfahre** von $v1$.
- Ein Kreis ist ein Weg $v_1 \dots v_n$ mit $n > 1$ für den gilt $v_1 = v_n$. Ein Graph ohne Kreise heißt **zyklenfrei** und enthält nur gerichtete Kanten.

Kanten sind als Menge von Knotenpaaren definiert. Es sind demnach keine Mehrfachkanten möglich. Eine alternative Definition von Graphen mit Mehrfachkanten bietet [Die00] – für das PACT-Programmiermodell wird diese jedoch nicht gebraucht. Die Knoten eines Graphen werden üblicherweise durch Kreise dargestellt, die Kanten als Linien. Die bildliche Darstellung ist jedoch unabhängig von der formalen Definition und eher eine Frage von Zweckmäßigkeit und Ästhetik [Die00, S. 2].

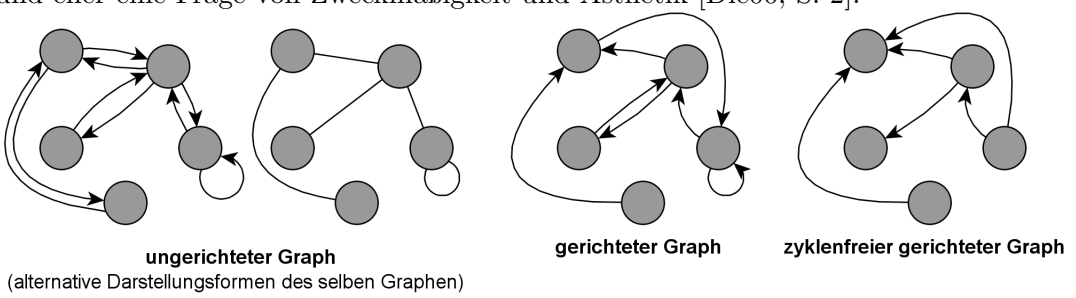


Abbildung 2.6: Ein ungerichteter, ein gerichteter und ein zyklenfreier Graph

PACT-Programme sind nicht strikt in die Phasen Map und Reduce (2.1) gegliedert, sondern erlauben die flexible Formulierung deutlich komplexerer Datenverarbeitungsaufgaben auf mehreren Eingabedatensätzen (siehe Abbildung 2.5). Die gerichteten Kanten verdeutlichen die Übertragung von Daten zwischen den Datenquellen, den Datensenken und den datenverarbeitenden Knoten (PACTs):

- Eine **Datenquelle** liest Daten aus Dateien und überführt diese in PACT-Programm-spezifische Datensätze (siehe Definition des Datenmodells im folgenden Abschnitt).
- Eine **Datensenke** schreibt PACT-Programm-spezifische Datensätze in eine Ausgabedatei.
- Ein **PACT** enthält die in Java[2] implementierte, benutzerdefinierte Verarbeitungslogik der Daten als *User-Defined Function* (UDF[10]). Zusätzlich werden mittels eines *Input Contracts* und optional mehrerer *Output Contracts* Eigenschaften bezüglich der Eingabe- und Ausgabedaten des PACT definiert (siehe Abbildung 2.7).

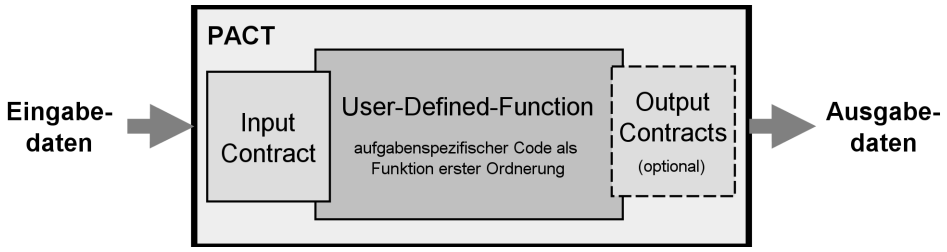


Abbildung 2.7: Aufbau eines PACTs

Input Contracts und User-Defined Functions
Input Contracts sind Funktionen zweiter Ordnung, die mit einer UDF als Funktionen erster Ordnung und den Eingabedaten aufgerufen werden. Input Contracts haben die Aufgabe eine Strategie zu definieren, mit der die Eingabedaten in PUs (2.1.2) aufgeteilt werden. Jede PU wird dann einzeln als Funktionsparameter einem Aufruf der UDF zur Verarbeitung übergeben. Da PUs unabhängig voneinander auf unterschiedlichen Rechenknoten verarbeitet werden können, bestimmt der Input Contract über die Möglichkeiten der parallelen Datenverarbeitung. UDFs werden mit Java implementiert und realisieren die benutzerdefinierte Verarbeitungslogik eines PACTs.

Derzeit bietet das PACT-Programmiermodell fünf verschiedene Input Contracts: Die aus dem Map/Reduce-Modell bekannten *Map* und *Reduce* zur Verarbeitung eines Datensatzes und *Cross*, *Match* und *CoGroup* zur Verarbeitung von zwei Eingabedatensätzen. Da PACTs durch ihre jeweiligen Input Contracts ihre grundlegenden Eigenschaften erhalten, werden sie entsprechend ihrem Input Contract bezeichnet: Ein PACT mit einem Match Input Contract wird demnach auch Match PACT genannt. Die Funktionsweise der einzelnen PACTs wird nachfolgend beschrieben und in Abbildung 2.8 grafisch dargestellt.

Die Eingabedaten sind im PACT-Programmiermodell eine ungeordnete Liste von Datensätzen $R = [r_1, \ldots, r_n]$. Ein Datensatz ist ein geordnetes Tupel mit Werten $r = (v_1, \ldots, v_n)$. Darüber hinaus ist für einige PACTs die Definition eines Schlüssels K für die Eingabedaten nötig. K ist eine Menge von Attributen von R, definiert als $\{k_1, \ldots, k_i\}$.

[10]UDF = User-Defined Function (deusch: benutzerdefinierte Funktion)

Für einen Datensatz r_k ist $K = k$. Eine zweite Dateneingabe wird entsprechend der Definition von R als S definiert; ihr Schlüssel entsprechend der Definition von K als F. Die UDF, bezeichnet durch f, kann beliebig viele Ausgabedatensätze produzieren.

PACTs mit einer Eingabe:

- *Map* definiert jeden Datensatz r der Eingabedaten R als eigene PU, die unabhängig von den anderen PUs an eine Instanz der UDF f übergeben wird. Der Map PACT ist definiert als:

$$\texttt{Map} : R \times f \rightarrow [f(r_1), \ldots, f(r_n)]$$

- *Reduce* partitioniert die Eingangsdaten R anhand ihres Schlüssels K. Alle Datensätze einer PU haben den gleichen Wert der Schlüsselattribute und werden gemeinsam einer UDF f übergeben. Der Reduce PACT ist definiert als:

$$\texttt{Reduce} : R \times f \times \mathsf{K} \rightarrow [f(r_1^{\mathsf{k}_1}, \ldots, r_{n_1}^{\mathsf{k}_1}), \ldots, f(r_1^{\mathsf{k}_l}, \ldots, r_{n_l}^{\mathsf{k}_l})]$$

PACTs mit zwei Eingaben:

- *Cross* erzeugt ein kartesisches Produkt über seine beiden Eingabedatensätze R und S. Dies bedeutet, dass jeder Datensatz der einen Datenquelle r mit jedem Datensatz der zweiten Datenquelle s kombiniert wird. Jedes dieser Kombinationen erzeugt einen eigenen Aufruf der UDF f. Der Cross PACT bildet somit das kartesische Produkt der beiden Eingabedaten und ist definiert als:

$$\texttt{Cross} : R \times S \times f \rightarrow [f(r_1, s_1), f(r_1, s_2), \ldots, f(r_N, s_M)]$$

- *Match* führt Datensätze r und s mit gleichen Werten der Schlüsselattribute $r.K = s.F$ aus zwei verschiedenen Eingaben R und S zusammen. Die UDF f wird mit allen Kombinationen schlüsselgleicher Paare aufgerufen. Der Match PACT eignet sich somit für einen Inner Join und ist definiert als:

$$\texttt{Match} : R \times S \times \mathsf{K} \times \mathsf{F} \times f \rightarrow [\{f(r, s) | r.\mathsf{K} = s.\mathsf{F}\}]$$

- *Co-Group* gruppiert die Datensätze r und s beider Eingabedaten R und S anhand der Werte der Schlüsselattribute K und F in eine PU. Datensätze mit dem gleichen Schlüssel v werden dann gemeinsam an die UDF f übergeben. Es wird auch eine PU erstellt, sollte für einen der Schlüssel nur Werte in einer der Eingaben vorhanden sein. Duplikate werde nicht entfernt, sollte ein Schlüsselwert in beiden Eingaben existieren. Der Co-Group PACT bietet sich daher für einen Outer Join an und ist definiert als:

$$\begin{aligned}&\texttt{CoGroup} : R \times S \times \mathsf{K} \times \mathsf{F} \times f \rightarrow \\ &[f(r_1^{\mathsf{v}_1}, \ldots, r_{n_1}^{\mathsf{v}_1}, s_1^{\mathsf{v}_1}, \ldots, s_{m_1}^{\mathsf{v}_1}), \ldots, f(r_1^{\mathsf{v}_l}, \ldots, r_{n_l}^{\mathsf{v}_l}, s_1^{\mathsf{v}_l}, \ldots, s_{m_l}^{\mathsf{v}_l})]\end{aligned}$$

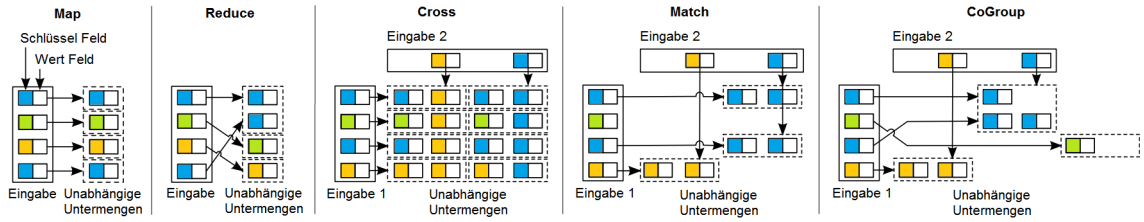


Abbildung 2.8: Graphische Darstellung der Funktionsweise von PACTs

Output Contracts
Während ein Input Contract Pflichtbestandteil eines PACTs ist, sind Output Contracts optional. Mit ihnen kann der Entwickler bestimmte Eigenschaften der durch seine UDF generierten Daten garantieren. Output Contracts haben keinen semantischen Einfluss auf das Ergebnis, sondern helfen dem PACT-Compiler kostenbasierte Optimierungen vorzunehmen. Sie erlauben effizientere parallele Ausführungspläne zu erstellen und so die Laufzeit des Programms signifikant zu verkürzen. [AEH+11]

Derzeit stehen folgende Output Contracts zur Verfügung:

- **Unique-Key:** Produzierte Datensätze haben über alle parallelen Instanzen hinweg einen eindeutigen Wert des Schlüssels. Dieser Output Contract kann auch für Datenquellen genutzt, von denen bekannt ist, dass sie nur Datensätze mit eindeutigen Schlüsselwerten gespeichert haben.
- **Output-Cardinalities:** Die obere und untere Grenze der Anzahl möglicher generierter Datensätze wird definiert.

2.2.2 PACT-Compiler

Bevor ein PACT-Programm mittels der Nephele-Ausführungsumgebung ausgeführt werden kann, muss es durch den PACT-Compiler in einen Nephele-Jobgraphen kompiliert werden. Die Knoten dieses DAGs (2.2.1) repräsentieren die Operatoren auf den Daten, die Kanten sind Kommunikationskanäle zwischen den Operatoren. Drei Varianten der Kommunikation stehen zur Verfügung: *Netzwerk-, In-Memory- und Datei-Kanäle*. Die beiden ersteren ermöglichen, dass ein Rechenknoten direkt und fast verzögerungsfrei die Ausgabe eines anderen Knotens weiter verarbeitet. Die Kommunikation zwischen Knoten über eine Datei ist deutlich langsamer, da erst die komplette Datei fertig geschrieben werden muss, bevor ein Nachfolgeknoten mit der Weiterverarbeitung anfangen kann. Da an dieser Stelle jedoch Zwischenergebnisse materialisiert werden, sind Dateikanäle hilfreich bei der Fehlererholung. [ABE+10]

Die Kapselung des Benutzercodes in UDFs innerhalb der PACTs verleiht dem PACT-Programmiermodell ganzheitlich betrachtet einen deklarativen Charakter. Der PACT-

Compiler kann somit bei der Kompilierung aus verschiedenen Ausführungsplänen kostenbasiert den Effektivsten auswählen. Dabei muss der Compiler mit beliebigem Benutzercode mit unbekannter Semantik sowie variierendem Parallelisierungsgrad umgehen. Um das PACT-Programm in einen möglichst optimalen Nephele-Jobgraphen zu überführen, müssen neben der Verteilung der zu analysierenden Daten auch die vorhandenen Output Contracts einbezogen werden. Diese geben dem Compiler Hinweise bezüglich der Eigenschaften der Daten, wie Kardinalität oder Einzigartigkeit der Schlüssel, welche bei der Optimierung ausgenutzt werden können. [ABE+10]

2.2.3 Nephele

Nephele ist ein Datenverarbeitungsframework[2] zur verteilten Ausführung von Datenflussprogrammen in Form von Nephele-Jobgraphen. Dabei verantwortet Nephele alle Details der parallelen Programmausführung, wie Ressourcenplanung, Aufgabenverteilung an die Einzelknoten, Synchronisation und zukünftig auch die Fehlererholung. Eingaben und Ausgaben der Verarbeitung durch Nephele können auf der lokalen Festplatte oder dem verteilten Datensystem HDFS 2.1.3 gespeichert werden. [WK09]

Innerhalb des Statosphere-Projekts ist Nephele entsprechend Abbildung 2.9 die Ausführungsumgebung für PACT-Programme, nachdem sie zu Nephele-Jobgraphen kompiliert wurden. Obwohl es durchaus denkbar wäre, PACT-Programme auch auf anderen graphbasierten Ausführungsumgebungen wie Dryad (2.2.4) laufen zu lassen, wird aus folgenden Gründen Nephele vorgezogen [BEH+10]:

- Nephele bietet weitreichende Möglichkeiten durch Parametrisierung die physikalische Ausführung eines Datenflussprogrammes zu steuern. So ist es zum Beispiel möglich, den Grad der Parallelität für jede Einzelaufgabe separat zu steuern oder explizit die Art des Kommunikationskanales zu definieren.

- Nephele ist speziell dafür ausgelegt, dynamisch weitere Ressourcen, auch von kommerziellen externen Rechnernetzwerken wie Amazons Elastic Cloud 2 [Ama11], einbeziehen zu können, wenn es die Auslastung erfordert.

- Es ist geplant, dass Nephele zukünftig während der Ausführung kontinuierlich Lastverteilungen und Optimierungen vornimmt.

Das Nephele-System führt übergebene DAGs jedoch nicht sofort aus, sondern muss sie erst in einen parallelen Datenflussgraphen *aufspannen*. Dies geschieht, indem die Knoten und entsprechend auch die Kommunikationskanäle vervielfacht werden. Die Anzahl paralleler Knoten hängt von einem Parallelisierungsparameter ab, der je Knoten während der Kompilierung gesetzt wurde. Dabei ist es möglich, dass entsprechend Abbildung 2.9 [BEH+10] für verschiedene Knoten, also verschiedene Programmteile, unterschiedliche Grade der Parallelisierung entstehen.

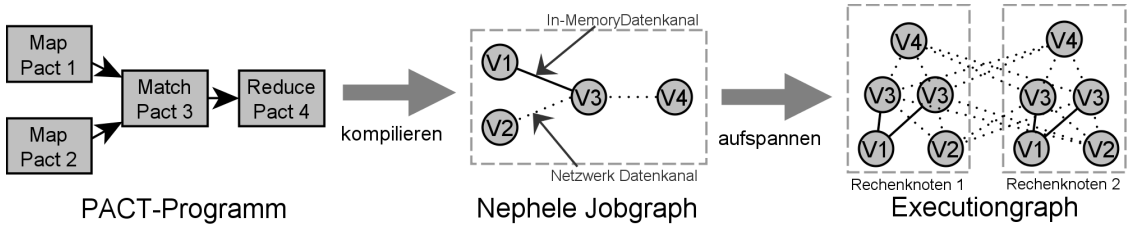


Abbildung 2.9: Aufspannen eines PACT-Programms

2.2.4 Verwandte Systeme

Ähnlich Stratosphere verfolgen auch andere Systeme den Ansatz verteilte Datenverarbeitungsaufgaben als Datenflussgraphen zu formulieren. Dafür implementieren diese Systeme eigene Ausführungsumgebungen, eigene Programmiermodelle und eigene Datenabfragesprachen. Für eine detaillierte Einführung sei auf die referenzierte Literatur verwiesen.

Microsoft's Dryad [IBY+07] ist eine verteilte parallele Ausführungsumgebung für als gerichtete Datenflussgraphen modellierte Dryad-Programme. Höhere Programmiersprachen, die Dryad-Programme zur Ausführung nutzen, sind *Dryad LINQ* [YIF+08] (zur Entwicklung innerhalb des .Net Frameworks [Mic12a]) und *SCOPE* [CJL+08] (mit einer SQL^2-basierten Syntax).

ASTERIX [BBC+11] ist ein Gemeinschaftsprojekt verschiedener Universitäten in den USA. Die Forscher haben die parallele Ausführungsumgebung für Datenflussprogramme *Hyracks* [BCG+11] entwickelt. Analyseaufgaben können direkt als *Hyracks-Job* oder auch deklarativ mittels der Abfragesprache *Asterix Query Language* (AQL) formuliert werden. Um eine Vielzahl halbstrukturierter Datenformate zu unterstützen, basiert der *Asterix Data Store* auf einem eigenen typisierten und selbstbeschreibenden Datenmodell, dem *Asterix Data Model* (ADM). Weitere Teile des Projekts sind ein *Hadoop Compatibility Layer* und ein *HiveQL (2.1.3) Query Plugin* .

2.3 Webanwendungen

Als Teil dieser Masterarbeit soll die PACT-GUI zur Erstellung von PACT-Programmen als Webanwendung umgesetzt werden. Nachfolgend wird der Begriff *Webanwendung* und damit verbundene Technologien und Konzepte eingeführt. Es werden die Vor- und Nachteile dieser Systemarchitektur zusammengefasst und einige weitere Charakteristika von Webanwendungen beschrieben, die für die Technologieentscheidungen (4.3) und die Implementierung (4.4) eine wichtige Rolle spielen.

2.3.1 Begriffsklärung

Eine Webanwendung ist eine Software, auf die mittels eines Web-Clients (beispielsweise ein Webbrowser - 2.3.2) zugegriffen wird. Sie basiert auf dem *Client/Server-Modell* [DEF+08, S. 21ff] – einem grundlegenden Konzept zur verteilten Ausführung von Software auf verschiedenen Computern[11]. In dem Modell stellt ein Anbieter (Server) einen Dienst zur Verfügung, welcher durch einen Konsument (Client) genutzt wird. Ein Server teilt durch die Erbringung seines Dienstes Ressourcen mit vielen über ein Netzwerk verbundenen Clients, welche diese ausschließlich konsumieren. Ein solcher Dienst ist eine über eine klar definierte Schnittstelle angebotene Funktionalität wie die Beantwortung einer Datenbankabfrage durch einen Datenbankserver (beispielsweise MySQL-Server [Ora12c]) oder die Bereitstellung einer Webseite durch einen Webserver (beispielsweise Apache Tomcat [The11b]). Von einem Webserver ausgelieferte Webseiten können bereits statisch als Datei vorliegen oder durch eine auf dem Webserver ausgeführte Webanwendung dynamisch erzeugt werden. Somit wird es erst durch Webanwendungen möglich, dynamisch auf Benutzereingaben zu reagieren und umfangreiche Funktionalitäten wie einen Onlineshop bereitzustellen [Moc05, S. 117f].

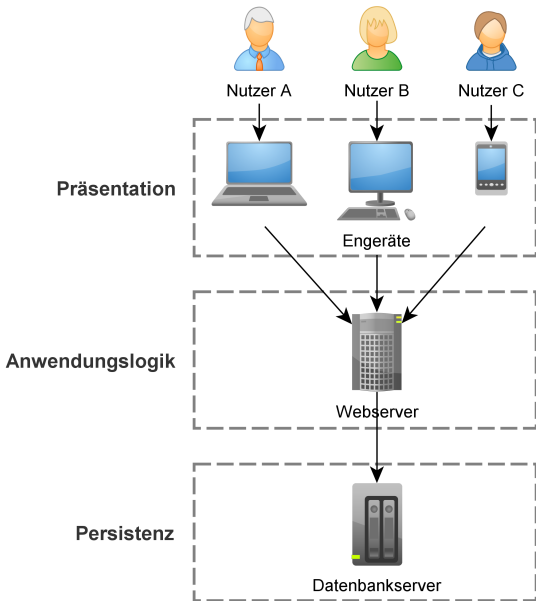


Abbildung 2.10: 3-Tier-Architektur von Webanwendungen

Wenn ein Server nicht nur einen Dienst erbringt, sondern gleichzeitig die Dienste anderer Server nutzt, spricht man von einer mehrschichtigen (englisch: multi-tier) Server-Architektur [DEF+08, S. 41ff]. In einer solchen Architektur können Rechner einer Schicht nur die Dienste der jeweilig darunter liegenden Schicht nutzen. Auf Webservern ausgeführte Webanwendungen können beispielsweise einen Datenbankserver nutzen, um Anwendungs-

[11] In hier nicht betrachteten Spezialfällen kann eine Webanwendung auch auf einem Computer gleichzeitig ausgeführt und genutzt werden.

daten *persistent* zu speichern – der Datenbankserver kann den Webserver jedoch nicht nutzen. Unter zusätzlicher Einbeziehung des Webbrowsers, als dem Teil der Webanwendung, der für die *Präsentation* der Benutzungsoberfläche verantwortlich ist, sind Webanwendungen entsprechend Abbildung 2.10 meist in drei Schichten (3-Tier) geteilt[12]: Präsentation, Anwendungslogik und Persistenz. Durch diese Schichtenabstraktion auf logischer und physischer Ebene entsteht eine klare Aufgabenteilung, womit sich die Wartbarkeit und Skalierbarkeit einer Anwendung erhöht.

Bei der Betrachtung von Webanwendungen wird jedoch auf Grund der sehr unterschiedlichen Technologien und Aufgaben der verschiedenen Schichten oft nur in Front-End (Webbrowser des Endnutzer) und Back-End (gesamte Serverumgebung) unterschieden. Abbildung 2.11 stellt in Anlehnung an [Bes02, S. 14f] gängige Aufgabenverteilungen zwischen Front- und Back-End schematisch dar.

1. **verteilte Präsentation:** In diesem bisher ausschließlich betrachteten Fall wird die Präsentation vom Server erzeugt und im Browser nur dargestellt.
2. **entfernte Präsentation:** Der Browser realisiert keine Anwendungslogik, generiert aber durch Ausführung eigener Präsentationslogik die dargestellte Oberfläche.
3. **kooperative Verarbeitung:** Neben der gesamten Präsentationslogik wird auch ein Teil der Anwendungslogik im Browser realisiert.

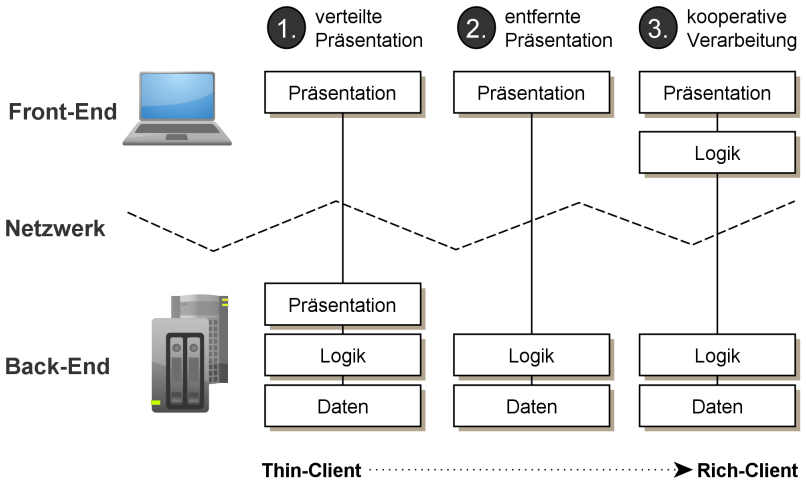


Abbildung 2.11: Mögliche Aufgabenverteilungen einer Client/Server-Webanwendung

Mit steigendem Umfang von im Webbrowser ausgeführter Anwendungs- und Präsentationslogik entwickelt sich eine Webanwendung von einer eher einfachen Webseite mit

[12] Je nach Literaturquelle [DEF+08, Bal99, For03, MBB07, RAS] wird der Presentation-Tier unterschiedlich definiert. Teilweise ist er auf Technologien wie JSP (4.3.1) beschränkt und um einen vierten Browser-Tier ergänzt.

geringer Benutzerinteraktion (einer *Thin-Client-Webanwendung*) hin zu einer komplexen Anwendungsoberfläche (einer *Rich-Client-Webanwendung* oder auch *Rich Internet Application - RIA*). Der Übergang ist dabei fließend, was sich in der unscharfen Definition der Begrifflichkeiten in der Literatur [CC08, Ker10, Wä10, Bau08] widerspiegelt.

Rich-Client-Webanwendungen stellen browserbasiert eine desktopähnliche graphische Benutzungsoberfläche (auch Benutzeroberfläche; englisch: Graphical User Interface; GUI) bereit. Die Befehlseingabe erfolgt mittels eines Steuergeräts wie der Maus über graphische Elemente, wie Menüs, Eingabefelder, Knöpfe, Listen und Fenster. Dabei soll die GUI in Aussehen und Bedienung den Interaktionparadigmen von Desktopapplikationen möglichst ähnlich sein, um den Lern- und Einarbeitungsaufwand der Endnutzer zu reduzieren [vL10].

Desktopanwendungen bieten durch die lokale Ausführung und Nutzung dennoch meist eine bessere Benutzbarkeit, Geschwindigkeit und sind weniger Sicherheitsrisiken ausgesetzt als Webanwendungen. Der Vorzug von Webanwendungen liegt in der einfachen und kostengünstigen Softwareverteilung durch die zentrale Ausführung und dezentrale Nutzung. Weitere Vorteile von Webanwendungen sind [Lin07]:

- Keine lokale Installation oder Administrationsrechte nötig
- Betriebssystem- und Hardware-unabhängig – jederzeit und überall auch mobil nutzbar
- Wartung und Weiterentwicklung sind zentral jederzeit möglich
- Nutzerdaten sind sicherer zentral verwahrt
- Das Nutzerverhalten ist sehr gut analysierbar
- Keine Viren, keine Softwareraubkopien

Gängige serverseitige Programmiersprachen für die Erstellung von Webanwendungen sind Java [Ora12a], PHP [Sch05, S. 471ff], Perl [Sch05, S. 371ff], Python [Sch05, S. 397ff], Ruby [Rub12] oder die im Microsoft .Net-Framework [Mic12a] angebotenen Sprachen Visual Basic sowie C#. Dazu existiert eine fast ebenso große Anzahl von Webservern zur Ausführung der Anwendungen, wie der Apache HTTP Server [The12a], der Jetty-Server [Mor11], der JBoss Application Server [Red11] oder Microsofts Internet Information Server [Mic12a]. Welche Technologie die passendste ist, hängt neben der grundsätzlich Eignung für den vorliegenden Anwendungsfall von verschiedenen anderen Aspekten ab. Dies können unter anderem die entstehenden Kosten, Vorwissen der Projektbeteiligten, Kundenvorgaben, allgemein Verbreitung der Technologie, existierende Hardware oder vorhandene Entwicklerwerkzeuge sein [GB09, S. 249].

2.3.2 Benutzungsschnittstelle Webbrowser

Als Benutzungsschnittstelle einer Webanwendung dient ein Webbrowser. Entsprechend dem Client/Server-Modell (2.3.1) erfüllt diese Software die Aufgabe, eine Anfrage (HTTP[2]-Request) in Form einer URL[2] an den Webserver zu senden und die Antwort (HTTP[2]-Response) zu verarbeiten. Angefragte Ressourcen können ganze Webseiten, aber beispielsweise auch Bild-, Video- oder Audio-Formate sein.

Innerhalb des Webbrowsers werden für die Umsetzung von Webanwendungen drei Basistechnologien verwendet: *HTML* dient dazu, den statischen Inhalt einer Webseite zu beschreiben. Durch die Verwendung von *CSS* lässt sich dieser Inhalt graphisch formatieren. Mit Hilfe von *JavaScript* können Elemente einer Webseite in Abhängigkeit von Benutzerinteraktionen oder anderen Ereignissen dynamisch verändert werden.

HTML[13] [Sch05, S. 7ff] ist eine textbasierte Auszeichnungssprache zur semantischen Strukturierung der Inhalte einer Webseite. Sie wird seit Anfang der 90-er Jahre immer weiter entwickelt und befindet sich derzeit kurz vor Verabschiedung der Version 5 durch die Standardisierungsorganisation W3C. Sie basiert auf SGML und ist damit von ihrer Syntax her mit XML[2] verwandt. Elemente der Webseite werden durch HTML-Tags definiert.

```
<p id="p1">Beispieltext in einem HTML-Paragraph-Element</p>
```

CSS[14] [Sch05, S. 163ff] ist eine ebenfalls von der W3C-Organisation standardisierte, deklarative Sprache zur Formatierung von HTML-Dokumenten. Mittels CSS-Definitionen lässt sich die Darstellung (zum Beispiel Farben, Anordnungen, Schrifteigenschaften, Rahmen) der Elemente eines HTML-Dokuments in Abhängigkeit des Ausgabemediums beeinflussen. Dies geschieht üblicherweise in einer separaten Datei, wodurch eine einfache Wiederverwendung für mehrere HTML-Dokumente eines Webauftritts ermöglicht wird.

```
#p1 {color: red; font-size: 5cm;}
```

JavaScript [Sch05, S. 263ff] ist eine dynamisch typisierte Scriptsprache, welche außer einigen syntaktischen Ähnlichkeiten nicht mit Java[2] verwandt ist. Sie kann sowohl objektorientiert als auch prozedural oder funktional verwendet werden. JavaScript wird im Browser ausgeführt und dient zur Implementierung der gesamten clientseitigen Anwendungslogik. Typische Anwendungsgebiete innerhalb von Webseiten sind die dynamische Manipulation von Webseiteninhalten oder die Validierung von Eingaben in Textfeldern. Aber auch komplette moderne Rich-Client-Webanwendungen (2.3.1) können mit JavaScript realisiert werden.

```
document.getElementById('p1').innerHTML += '!!!';
```

Asynchrone Serverinteraktion durch Ajax

Webanwendungen unterscheiden sich auf Grund der verteilten Architektur (2.3.1) in ihrem Interaktionsparadigma von GUIs lokal installierter Desktopsoftware. Die Benutzung einer Webanwendung unterliegt mit den bisher vorgestellten Techniken dem Paradigma einzelner Seiten, zwischen denen der Benutzer wechselt [San11b]. Für jede Interaktion mit dem Server, wie dem Löschen eines Datenbankeintrages, muss die gesamte dargestellte Webseite neu geladen werden. Dies erhöht nicht nur deutlich den Datenverkehr, sondern verringert zudem die Benutzerfreundlichkeit, da die Antwortzeit einer Nutzeraktion recht

[13] HTML = Hypertext Markup Language
[14] CSS = Cascading Style Sheets

hoch ist. Durch die Verwendung von Ajax können Informationen mit dem Server ausgetauscht werden, ohne stets die gesamte Webseite neu zu laden.

Ajax[15] [GU10, S. 56] ist eine weitere von der W3C standardisierte Webtechnik. Es ist keine eigene Programmiersprache, sondern beschreibt ein Konzept der Verwendung von JavaScript zur asynchronen Datenübertragung zwischen Browser und Server. Mit Ajax ist es möglich, eine HTTP[2]-Anfrage im Hintergrund an den Server zu stellen, ohne die derzeit angezeigte Webseite neu zu laden. Übliche Anwendungsfälle von Ajax sind das dynamische Nachladen von Webseiteninhalten, Daten oder sogar Applikationslogik. Erst durch die Verwendung von Ajax ist es möglich, Webanwendungen mit desktopähnlichem Verhalten zu implementieren.

2.3.3 Weitere Konzepte

Im Verlauf dieser Masterarbeit werden weitere grundlegende Konzepte der GUI-Entwicklung genutzt, die im Folgenden vorgestellt werden.

Model-View-Controller (MVC) [Ker10, S. 56f] ist ein Architekturmuster, das seit Anfang der 80-er Jahre für die Erstellung von graphischen Benutzungsoberflächen eingesetzt wird. MVC verfolgt das Ziel, die Verantwortlichkeiten innerhalb der Anwendung klar zu trennen[16]. Dadurch kann sich jeder Entwickler auf seinen Aufgabenbereich konzentrieren, spätere Anpassungen werden erleichtert und die Wiederverwendbarkeit und Testbarkeit der einzelnen Komponenten wird erst ermöglicht [Met02, S. 79].

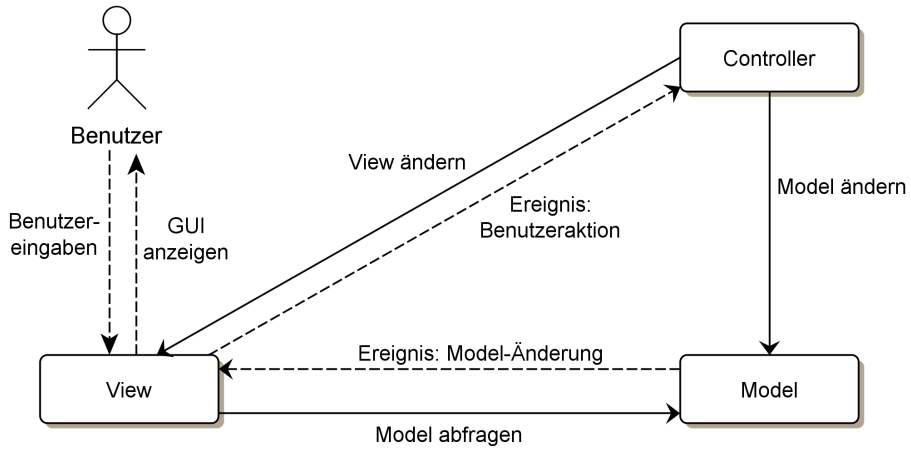


Abbildung 2.12: Model-View-Controller

MVC besteht entsprechend Abbildung 2.12 aus den folgenden drei Teilen:

- **Model** (Modell): Umfasst alle Daten der Anwendung; ist aber unabhängig von deren

[15] Ajax = Asynchronous JavaScript and XML
[16] bekannt als *Separation of concerns*

Darstellung oder Verarbeitung. Nur Model-Änderungen werden nach außen bekannt gegeben.

- **View** (Präsentation): Umfasst die Programmoberfläche. Sie besteht beispielsweise aus Eingabefeldern, Knöpfen, Listen und Menus. Dabei kennt die View das darzustellende Model und gibt Benutzerinteraktion an den Controller weiter.
- **Controller** (Steuerung): Enthält die gesamte Verarbeitungslogik der Anwendung. Der Controller ändert das Model und reagiert auf Benutzerinteraktion mit der View.

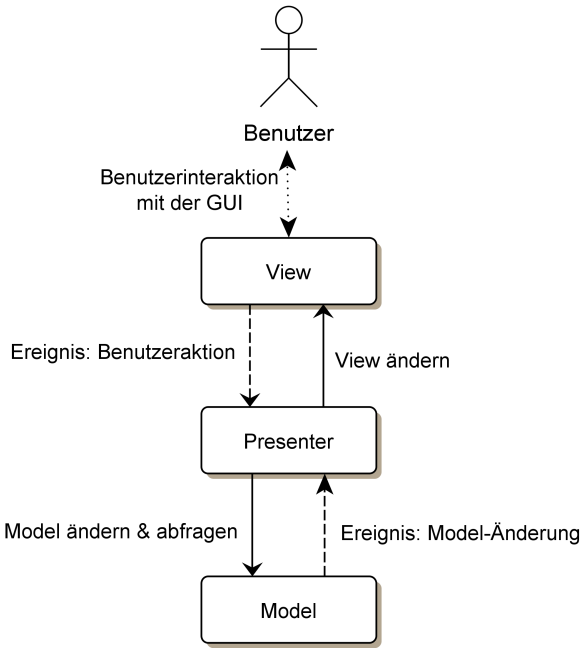


Abbildung 2.13: Model-View-Presenter

Model-View-Presenter (MVP) [Ker10, S. 57f] ist ein auf Basis von MVC entstandenes Architekturmuster entsprechend Abbildung 2.13. Es gibt folgende Unterschiede:

- Die Aufgabenverteilung ist in MVP deutlich strikter festgelegt als bei MVC. Je nach Programmiersprache und Framework[2] war teils unscharf definiert, wo beispielsweise die Validierung von Formulareingaben oder die Formatierung von Daten zu realisieren ist [MBB07, S. 163ff]. In MVP gehört jegliche Logik in den Presenter.
- Der Presenter fungiert als zentrale Schnittstelle zwischen den nun strikt voneinander getrennten View und Model. Dies kann sinnvoll sein, wenn entsprechend Abbildung 2.11 die View und das Model nicht innerhalb der gleichen Ausführungsumgebung implementiert sind (2.3.1 - entfernte Präsentation, kooperative Verarbeitung) [Ker10, S. 57].

- Die Rolle der View ist in MVP auf ein Minimum reduziert. Der komplette Verzicht auf Logik soll die View soweit vereinfachen, dass explizite Softwaretests der View überflüssig werden.

Der Aufbau von MVP erinnert an die beschriebene 3-Tier-Architektur von Webanwendungen (2.3.1). Die Konzepte haben durchaus Berührungspunkte, sollten aber dennoch nicht verwechselt werden. Mit n-Tier wird die physikalische Schichtenaufteilung der Gesamtapplikation auf verschiedene durch ein Netzwerk verbundene Rechner beschrieben[17]. MVP (oder auch MVC) sind Architekturmuster zur Aufteilung der verschiedenen Teile des Programmcodes einer Anwendung, wobei der physikalische Ort keine Rolle spielt.

Publish/Subscribe

Wenn alle graphischen Elemente einer ereignisgesteuerten Benutzungsoberfläche jeweils nach dem MVC- oder MVP-Muster implementiert sind, stellt sich die Frage, wie sich eine Änderung in einem Steuerelement[2] auf ein oder mehrere andere auswirken kann.

Mit direkten Referenzen zwischen den Elementen der GUI würden viele Abhängigkeiten zwischen Komponenten entstehen. Dadurch würden die mit MVC/MVP angestrebten Vorteile der besseren Test-, Austausch- und Wiederverwendbarkeit zunichte gemacht.

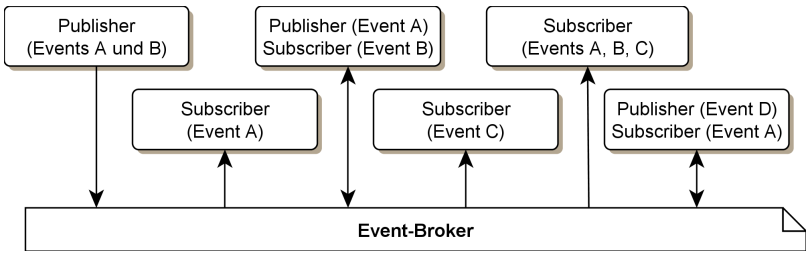


Abbildung 2.14: Einsatz des Event-Bus in der PACT-GUI

Die Lösung dieses Problems stellt das 1987 [BJ87] erstmals beschriebene *Publish/Subscribe-Entwurfsmuster*[18] [Gra01, S. 175] bereit. Entsprechend Abbildung 2.14 hält jedes GUI-Objekt nur noch eine Referenz auf ein globales Nachrichtenverteilungssystem (Event-Broker). Über diese Verbindung kann ein Objekt Ereignisse global bekannt geben (Publish) oder sich registrieren über bestimmte Ereignisse informieren zu werden (Subscribe). Die verschiedenen Steuerelemente[2] der Anwendung müssen somit keine Kenntnis voneinander haben, können sich aber dennoch gegenseitig beeinflussen. Dieses Prinzip der Unabhängigkeit von Systemkomponenten ist in der Softwareentwicklung bekannt als *Loose Coupling* [Rya09] und sollte im Hinblick auf die Austausch- und Testbarkeit der Einzelteile stets angestrebt werden.

[17] Natürlich ist auch denkbar die verschiedenen Tiers auf einem Rechner zu auszuführen, wobei die strenge Trennung zumindest auf Softwareebene dann weiter besteht

[18] auch bekannt als „Observer" Muster

2.4 Vorgehensmodelle der Softwareentwicklung

Nach Balzert [Bal08, S. 446, 515] soll jede Softwareentwicklung in einem festgelegten organisatorischen Rahmen erfolgen. Prozessmodelle[1] – auch Vorgehensmodelle genannt – bilden einen solchen Rahmen. Sie definieren innerhalb des Software-Projekts die Reihenfolge der Arbeitsabläufe, durchzuführende Aktivitäten sowie zu erstellende Teilprodukte.

Dieses Unterkapitel stellt die bekanntesten klassischen Basismodelle sowie auf deren Grundlage entstandene weiterführende Modelle vor. Es werden jeweils Stärken und Schwächen erörtert, um darauf aufbauend die Wahl des eigenen Vorgehens und damit die Gliederung dieser Arbeit zu begründen.

2.4.1 Basismodelle

Basismodelle [Bal08, S. 515] sind auf Projektebene entstandene rudimentäre Modelle, welche Softwareentwicklungsprojekte grobgranular in Phasen gliedern. Sie definieren deren zeitlichen Reihenfolge sowie Qualitäts- und Rückkopplungsschritte, welche in den einzelnen Phasen durchgeführt werden.

Sequenzielles Modell
Die ersten und einfachsten Vorgehensmodelle in der Softwareentwicklung entsprechen dem sequenziellen Modell (auch Phasenmodell) [Bal08, S. 519ff]. Zeitgleich mit der Entwicklung der ersten Hochsprache wurde 1956 von Benington das *stagewise model* [Hof08, S. 493] als eine erste sequenzielle, phasenorientierte Vorgehensweise vorgeschlagen. Von Royce [Roy70] wurde diese Idee später weiterentwickelt zum Wasserfallmodell entsprechend Abbildung 2.15. Es sieht sieben Phasen vor, die sequenziell nacheinander durchlaufen werden. Eine Phase muss vollständig und stabil abgeschlossen sein, bevor die nächste beginnen kann. Wenn nötig erlauben zusätzliche Rückkopplungsschleifen zur vorherigen Phase zurückzuspringen. Spätere Weiterentwicklungen des Modells wandeln die Anzahl, Aufteilungen und Aufgaben der Phasen ab [PP04, Bal00, S. 17] oder erlauben die teilweise zeitliche Überlappung (nebenläufiges Modell [Bal08, S. 519ff]), um die gesamte Entwicklungszeit zu verkürzen.

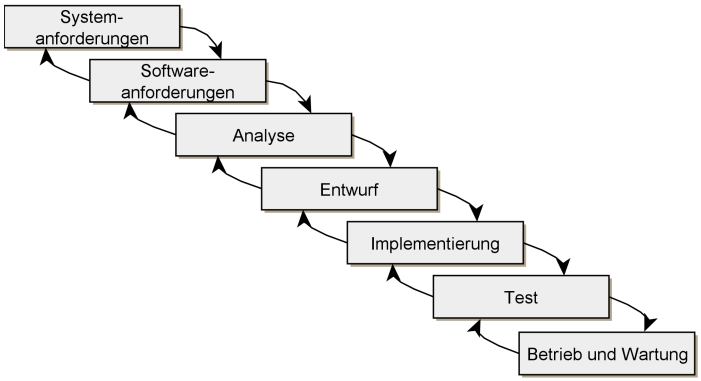


Abbildung 2.15: Wasserfallmodell nach Royce

Das sequenzielle Vorgehen ist leicht verständlich und erfordert nur einen geringen Managementaufwand. Dennoch besitzt es in seiner Reinform heute keinerlei praktische Relevanz mehr [Hof08, S. 494ff]. Ein Grund ist, dass zu Beginn eines Projekts der Auftraggeber selten alle Anforderungen an das Produkt vollständig und endgültig kennt und formulieren kann. Außerdem sind die wenigsten Entwicklerteams in der Lage, auf Basis von Anforderungen ein komplexes Gesamtsystem bis ins Detail zu spezifizieren und zu entwerfen. Dies trifft insbesondere zu, wenn neuartige, den Entwicklern noch nicht bis ins Detail vertraute Techniken zum Einsatz kommen.

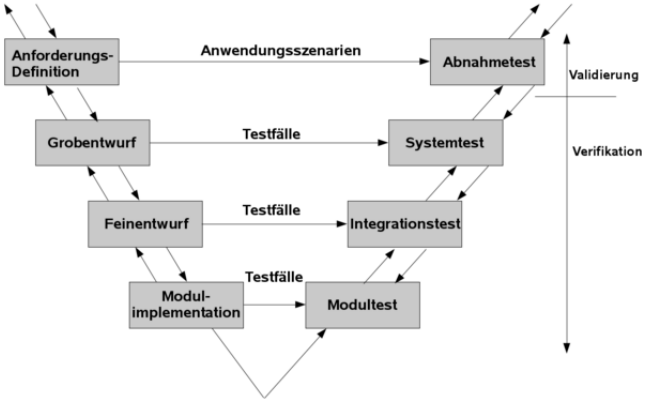


Abbildung 2.16: V-Modell nach Boehm

Historisch gesehen ist das **V-Modell** [Boe81] entsprechend Abbildung 2.16 als eine Erweiterung des sequenziellen Modells zu verstehen [Bal08, S. 533ff]. Den bekannten Phasen des Wasserfallmodells auf der linken Seite der Grafik werden jeweils nachgelagerte testende Phasen gegenübergestellt. Dadurch ist in jeder Phase die Planung der Qualitätssicherung inbegriffen und somit die strikte Trennung von Implementierung und Test, die der gängigen Praxis heutiger Softwareentwicklung widerspricht, aufgebrochen. Eine höhere Testabdeckung und allgemein höhere Softwarequalität sollen so erreicht werden.

Inkrementelles Modell

Bei der Softwareentwicklung nach dem inkrementellen Modell [Bal08, S. 526f] wird versucht, frühestmöglich Teilprodukte der Software an den Auftraggeber auszuliefern, damit dieser Erfahrungen mit der Software sammeln kann. Sein Feedback fließt in die weiteren Teilprodukte ein und hilft Fehlentwicklungen frühzeitig zu vermeiden. Voraussetzung für dieses Vorgehen ist, dass sich das Softwareprodukt in einzelne voneinander unabhängige Bestandteile auseinandernehmen und später wieder zusammenfügen lässt. Wie in Abbildung 2.17 [Bal08, S. 527] dargestellt, kann dies horizontal in nebeneinander stehenden Teilprodukten, vertikal in aufeinander aufbauenden Schalen oder in einer Mischform der beiden geschehen. Um die spätere Kompatibilität der Einzelteile zu gewährleisten, müssen auch beim inkrementellen Vorgehen die Anforderungen an die Software schon bei Projektbeginn vollständig erfasst werden, was wie beim sequenziellen Vorgehen erwähnt, oftmals

jedoch nicht möglich ist.

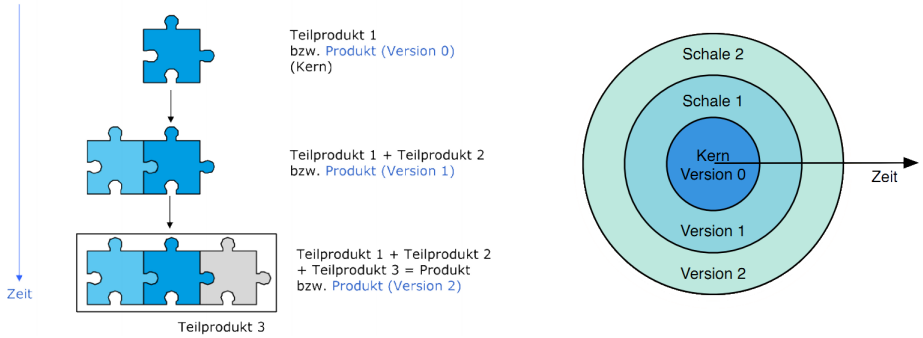


Abbildung 2.17: Softwarezerlegung im inkrementellen Modell

Evolutionäres/iteratives Modell
Für Projekte, in denen der Auftraggeber seine Anforderungen noch nicht vollständig kennt, eignet sich das evolutionäre Modell [Bal08, S. 529ff]. Bei diesem Ansatz wird zuerst nur der Produktkern anhand der absoluten Mussanforderungen des Auftraggebers in einer sogenannten *Nullversion* entworfen und implementiert. Mit dieser sammelt der Auftraggeber Erfahrungen, um weitere Produktanforderungen zu definieren, welche dann in die nächste Produktversion einfließen. Bei dieser Vorgehensweise kann zu jedem Zeitpunkt des Projektes flexibel auf Änderungswünsche und neue Anforderungen reagiert werden. Der Auftraggeber kann durch die kleinen Arbeitsschritte die Entwicklung mit geringem Managementaufwand gut steuern und kontrollieren. Die nötigen zeitlichen und finanziellen Ressourcen für das gesamte Projekt sind hingegen durch die Natur des Modells kaum planbar. Eine weitere Gefahr liegt in einer nicht ausreichend flexiblen Systemarchitektur früher Phasen, um auch noch nicht bekannte Anforderungen späterer Phasen abzudecken. Frühe Entwurfsentscheidungen müssen demnach ähnlich fundiert durchdacht sein wie bei anderen Vorgehensweisen.

In den Bereich der iterativen Vorgehensmodelle fällt auch das **Spiralmodel** [Boe88]. Bei seiner Anwendung sind für jede Projektphase entsprechend Abbildung 2.18 vier Phasen (zusammen ein Iterationszyklus) zu durchlaufen. Da der Inhalt der Entwicklungsphase (3. Phase) nicht genau spezifiziert ist, kann das Spiralmodell als sogenanntes generisches Metamodell auch für die Auswahl des je Iteration passenden Vorgehensmodells genutzt werden. Eine Iteration kann einem Teilprodukt im inkrementellen Vorgehen, einer Produktgeneration im evolutionären Vorgehen oder auch einer Phase des Wasserfallmodells entsprechen. Das Spiralmodell ist mit einem hohen Managementaufwand verbunden und daher eher für große Projekte geeignet. [Bal08, S. 556ff]

Weitere Basismodelle
Andere Basismodelle sind nicht als vollständige Vorgehensbeschreibungen zu verstehen, sondern decken nur Teilaspekte eines Projekts ab. Sie beschreiben spezielle Probleme wie

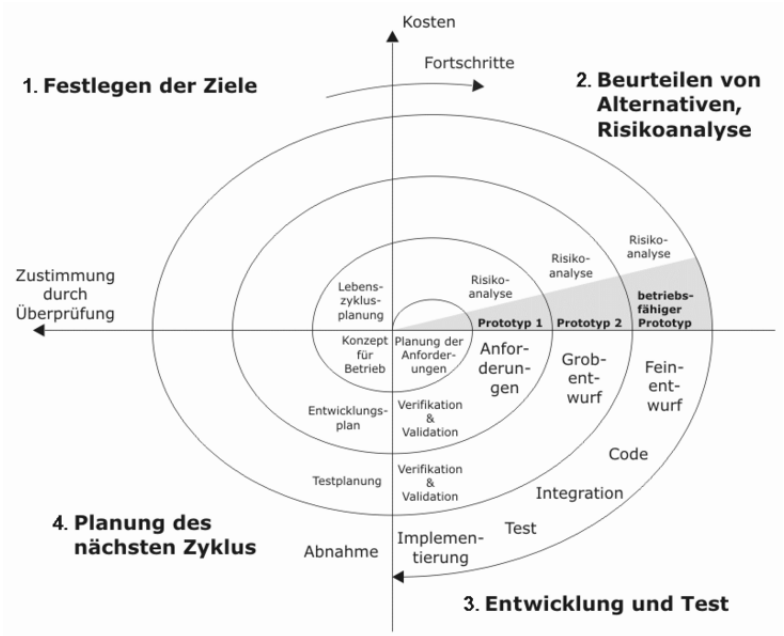


Abbildung 2.18: Spiralmodell nach Boehm

die geographisch oder zeitlich verteilte Softwareentwicklung oder den korrekten Umgang mit Softwareproduktfamilien und -linien. Beides hat im Rahmen der vorliegenden Masterarbeit keine Relevanz. Von besonderer Bedeutung sind jedoch das Prototypenmodell und das komponentenbasierte Modell.

Prototypenmodell: [Bal08, S. 537ff] Mit einem Software-Prototypen lassen sich ausgewählte Eigenschaften des zukünftigen Systems zu einem sehr frühen Projektzeitpunkt testen und simulieren. Horizontale Prototypen decken eine bestimmte Ebene des Systems, wie Benutzungsoberfläche oder die Datenbanktransaktionen, ab. Vertikale Prototypen implementieren ausgewählte Systemteile über alle Ebenen. Auch hinsichtlich ihrer Verwendung lassen sich Prototypen unterscheiden. Demonstrationsprototypen helfen vor allem der Kommunikation zwischen Projektbeteiligten. Laborprototypen oder Pilotsysteme sind eher technisch ausgerichtet. Sie dienen als Diskussionsgrundlage für Entwurfsentscheidungen oder sichern experimentell die Realisierbarkeit bestimmter Anforderungen.

In Kapitel 4.2.1 wird die Verwendung von Prototypen im Rahmen der Entwicklung der PACT-GUI näher beschrieben.

Komponentenbasiertes Modell: [Bal08, S. 532ff] In der industriellen Produktion werden Halbfabrikate (Komponenten) verwendet, um die Wirtschaftlichkeit der Produktion zu erhöhen. Das komponentenbasierte Modell[19] verfolgt in der Softwareentwicklung das gleiche Ziel. Bei jedem Entwicklungsschritt ist zu prüfen, ob bereits fertige, für die Wie-

[19] bekannt als „Component Based System Development"

derverwendung optimierte Softwarebausteine existieren. Dies können Funktionsbibliotheken, Programmierschnittstellen (sogenannte APIs), Komponenten und Frameworks sein[20] [Nas03]. Durch Softwarewiederverwendung ergeben sich die folgenden Vorteile [Bal08, S. 532ff]:

- Zeitersparnis und Produktivitätssteigerung
- Verbesserung der Softwarequalität, da die Komponenten durch ihren großen Nutzer- und besonders bei freier Software auch Entwicklerkreis sehr ausgereift sind
- Implizite Adaption von Konzepten und Best-Practices aus einer Vielzahl ähnlicher Projekte
- Schnelle Einarbeitung für neue Projektmitglieder, da viele Komponenten weithin bekannt sind und meist sehr umfangreiche Dokumentation bieten
- Fokussierung auf Kernelemente der Eigenentwicklung

Als nachteilig kann sich die Abhängigkeit von einem Komponentenhersteller, die Inkompatibilität zwischen Komponenten oder die nicht ausreichende Erweiterbarkeit herausstellen. Somit ist auch die Auswahl externer Komponenten als eine wichtige Architekturentscheidung sorgfältig durchzuführen.

2.4.2 Weiterentwickelte Modelle

Auf Grundlage der vorgestellten Basismodelle erfolgte zunächst eine Weiterentwicklung in zwei Richtungen:

(a) **Rahmen- oder Referenzmodelle** [Bal08, S. 565ff] definieren Ziele, aber enthalten keine Aussagen dazu, wie diese operational erreicht werden sollen. Diese Referenzmodelle strukturieren Prozesse in Qualitätskategorien und beschreiben die jeweils enthaltenen Tätigkeiten unter Einbeziehung der Unternehmensebene. Beispiele sind CMMI, SPICE/ISO 15504, ISO9000, TQM.

(b) **To-do-Modelle** [Bal08, S. 619ff] führen alle nötigen Aktivitäten des gesamten Projektes für alle Beteiligten auf allen Unternehmensebenen auf. Die Tätigkeitsbeschreibungen sind oftmals so feingranular, dass sie mehrere Hundert oder Tausend Aktivitäten detailliert festlegen und beschreiben. Deswegen werden diese Modelle auch schwergewichtige oder monumentale Modelle genannt. Beispiele sind das V-Modell XT, RUP-, PSP- oder TSP-Modell.

Die Modellbeschreibungen der genannten Modellarten füllen ganze Bücher und erfordern auf Grund ihrer Komplexität meist die Einbeziehung eines Prozessexperten in das Projektteam. Teilweise ist ihre Anwendung auch für kleinere Projekte durch sogenanntes

[20] StringUtils ist beispielsweise eine Funktionsbibliothek. JavaMail ist eine API (siehe Glossar). JavaBeans ist ein Komponentenstandard (4.4.2). Spring ist ein Framework (siehe Glossar).

Tailoring (zuschneiden) möglich, was dennoch die Kenntnis des gesamten Modells erfordert. Monumentale Modelle sollten daher erst ab einer Projektgröße von 10-20 Mitarbeitern in Erwägung gezogen werden [Bal08, S. 619, 686].

Agile Modelle
Als Gegenbewegung zu den immer schwieriger zu verstehenden monumentalen Modellen entstanden in den letzten Jahren die leichtgewichtigen, agilen[21] Modelle [Bal08, S. 651ff]. Der agile Ansatz versteht die Softwareerstellung wieder primär als kreativen Prozess, der nicht unnötig bürokratisiert werden sollte. Es sollten genau so viel Prozess und Dokumentation vorhanden sein, dass sich der Aufwand lohnt.

Die Befürworter agiler Entwicklung haben ihre Ideen im agilen Manifest aus dem Jahr 2001 [Bec01] in folgenden vier Punkten zusammengefasst:

- Einzelpersonen und Interaktionen sind wichtiger als Prozesse und Werkzeuge.
- Laufende Systeme sind wichtiger als umfassende Dokumente.
- Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen.
- Reaktion auf Änderungen ist wichtiger als das Verfolgen eines Plans.

Um diese Ideen auf die Projektebene zu übertragen, entstanden auch im Bereich agiler Modelle verschiedene Ansätze, wie eXtreme Programming (kurz XP) [Hof08, S. 506], Feature Driven Development, Scrum oder Crystal. Diese sind aber meist keine detaillierten Handlungsanweisungen, sondern als Best-Practice-Sammlungen zu verstehen. XP als einer der bekanntesten Vertreter stellt fünf Werte in den Mittelpunkt, um die Philosophie des Modells zusammenzufassen: Kommunikation, Einfachheit, Rückmeldung, Mut und Respekt. Zusätzlich werden diverse Vorschläge zur Umsetzung dieser Werte in Softwareentwicklungsprojekten gemacht, wie Programmierung in Paaren, tägliche sehr kurze Meetings im Stehen, Kurze Release-Zyklen und Retrospektiven.

Softwareentwicklung nach dem agilen Ansatz ist sehr flexibel veränderten Kundenwünschen gegenüber, erhöht die Zufriedenheit der Projektbeteiligten und steigert oft deutlich die Produktivität. Auf der anderen Seite setzt der agile Ansatz eine ganze Reihe von Rahmenbedingungen für seine Anwendung voraus. Die enge Zusammenarbeit mit dem Kunden, die Art der Software, die Teamgröße sowie das Persönlichkeits- und Qualifikationsprofil der Teammitglieder sind nur einige der wichtigen Faktoren, damit agile Softwareentwicklung die Erfolge bringen kann, die es verspricht. [Bal08, S. 654ff]

[21] agil = flink, wendig, beweglich

Kapitel 3

Problemstellung

Dieses Kapitel beschreibt die Problemstellung, welche dieser Masterarbeit zugrunde liegt. Einleitend wird der aktuelle Prozess zur Implementierung eines PACT-Programms beschrieben (3.1). Anschließend werden dabei existierende Probleme benannt (3.2) und ein Lösungsansatz definiert (3.3).

3.1 Erstellungsprozess von PACT-Programmen

PACT-Programme (2.2.1) werden komplett in der Programmiersprache Java[1] innerhalb einer Java-Entwicklungsumgebung implementiert. Folgende Schritte sind nötig zur Erstellung eines PACT-Programms:

1. Für die Entwicklung benötigte Software muss installiert und konfiguriert werden. Üblicherweise umfasst dies mindestens die Entwicklungsumgebung Eclipse IDE, die Versionsverwaltungssoftware Git und die Build Management Software Maven (alle siehe 4.3.4).

2. Der gesamte Quellcode des Stratosphere-Projekts (2.2) muss mit Git aus einem zentralen Softwarearchiv heruntergeladen und innerhalb der Eclipse IDE als Projekt angelegt werden.

3. Um ein PACT-Programm zu entwickeln, muss ein neues Programmpaket mit korrekter Struktur von Unterverzeichnissen (siehe Abbildung 4.6) erstellt werden.

4. Benötigte externe Bibliotheken werden als Java-Archiv (.jar-Datei[1]) im *lib*-Unterverzeichnis abgelegt.

5. Eine Manifest-Datei zur Konfiguration des PACT-Programms ist anschließend im *META-INF*-Unterverzeichnis anzulegen.

6. Die Verarbeitungslogik des PACT-Programms muss mittels einer Vielzahl von Java-Klassen mit allen nötigen Methoden, Parametern und Annotationen programmiert

[1] siehe Glossar

werden. Je PACT ist eine PACT-spezifische Klasse erforderlich. Alle PACT-Objekte und die Datenflüsse zwischen ihnen werden mit einer zusätzlichen PACT-Plan-Klasse (Programmbeispiel 3.1) erstellt.

7. Abschließend müssen die Java-Quellen des PACT-Programms mittels des Kommandozeilen-Programms *javac* kompiliert werden. Dem javac-Programmaufruf werden zusätzlich Pfade zum Manifest (5.) und allen benötigten externen Bibliotheken (4.) mittels Parametern übergeben. Nur fehlerfreie Quellen können erfolgreich kompiliert werden, wodurch für jede .java-Datei eine gleichnamige .class-Datei erzeugt wird.

8. Es wäre unpraktisch PACT-Programme als Sammlung von .class-, Manifest- und .jar-Dateien an andere Benutzer oder den PACT-Compiler zu übergeben. Aus diesem Grund werden in einem letzten Schritt allen nötigen Ressourcen mit dem Kommandozeilen-Programm *jar* zu einem Java-Archiv gepackt.

Nach korrekter Ausführung dieser Schritte ist es möglich, das erstellte PACT-Programm in Form einer JAR-Datei an den PACT-Compiler (2.2.2) zur Überführung in einen Nephele-Jobgraphen und Ausführung auf Nephele (2.2.3) zu übergeben. Dies geschieht zusammen mit den Eingabedaten mittels einer bereits existierenden Webanwendung (dem *Nephele and PACTs Query Interface*).

```
class PACT-Plan {
        Datenquelle D1 = new Datenquelle(dateipfad1)
        Datenquelle D2 = new Datenquelle(dateipfad2)
        PACT P1 = new Map1PACT(D1)
        PACT P2 = new Map2PACT(D2)
        PACT P3 = new MatchPACT(P1, P2)
        PACT P4 = new ReducePACT(P3)
        Datensenke D3 = new Datensenke(P4, dateipfad3)
}

class Datenquelle {
        [Code zum Lesen der Datenquellen]
}

class Map1PACT {
        [UDF des 1. Map-PACTs]
}

// ... weitere Klassen für Map2PACT, MatchPACT, ReducePACT, Datensenke
```

Abbildung 3.1: Vereinfachter Pseudo-Code eines PACT-Programms

3.2 Konflikt zwischen mentalem und technischem Modell

Der in 3.1 beschriebene Prozess zur Erstellung von PACT-Programmen ist zeitaufwändig, kompliziert und fehleranfällig. In seiner jetzigen Form ist ein hohes Maß an Verständnis von Systeminterna des Stratosphere-Projekts erforderlich, weshalb die Erstellung von

PACT-Programmen nur einem sehr begrenzten Personenkreis möglich ist.

Was darüber hinaus den Prozess selbst für erfahrene Entwickler umständlich macht, ist der Bruch zwischen dem *mentalen Modell* des PACT-Programms im Bewusstsein des Entwicklers und der eigentlichen *Implementierung*.

Ein **mentales Modell**[2] [Dut94] ist eine kognitive Repräsentation realer, hypothetischer oder imaginärer Sachverhalte, welche Menschen zum Verständnis eines Phänomens nutzen. Das mentale Modell eines PACT-Programms ist stark von der bildlichen Veranschaulichung als Graph entsprechend Abbildung 2.5 geprägt. Es besteht aus gerichteten Verbindungen zum Datentransport und den PACTs (2.2.1). Diese haben bestimmte Eigenschaften und enthalten Benutzercode (UDF) zur Verarbeitung der Daten.

Die **Implementierung** eines PACT-Programms erfolgt mittels der Programmiersprache Java. Programmbeispiel 3.1 zeigt den Pseudo-Code des PACT-Programms aus Abbildung 2.5. Für den Graphen des PACT-Programms (PACT-Plan) sowie Datenquellen, Datensenken und PACTs müssen Java-Klassen erstellt werden. Der Datenfluss wird implizit definiert durch Übergabe eines Objekts als Parameter für die Erstellung eines anderen Objekts. Bereits in der dargestellten vereinfachten und verkürzten Form (es werden keine UDFs, keine Output Contracts oder weitere Eigenschaften wie der Parallelisierungsgrad zugewiesen) ist das PACT-Programm für den Leser nur verständlich, wenn er es in sein jeweiliges mentales Modell überführt.

Bei der Erstellung eines PACT-Programms kann sich ein Entwickler demnach nicht ausschließlich auf die Modellierung des Datenflussgraphen und die Implementierung der UDFs konzentrieren, sondern muss zudem sein mentales Modell eines Datenflussgraphen in eine Java-spezifische sequenzielle Form überführen. Der Entwickler muss also sein eigenes Denkmodell verlassen, wodurch die kognitive Belastung, also der Grad der nötigen mentalen Anstrengung zur Ausführung einer Aufgabe [Dut94], erhöht wird.

Wie [Dut94] anhand referenzierter Beispiele[3] und Untersuchungen belegt, fällt Anwendern die Bedienung eines Systems umso leichter, je mehr *ihr eigenes mentales Modell der technischen Implementierung entspricht*. Demnach kann die Bedienung eines Systems vereinfacht werden, indem sich die reale Systembedienung und das mentales Modell der Bedienung im Bewusstsein des Anwenders soweit wie möglich annähern. Dafür kann entweder das mentale Modell des Benutzers beispielsweise durch eine Schulung dem technischen System angepasst werden oder die Systembedienung kann in Richtung des mentale Modells des Nutzers verändert werden.

Auf die Implementierung eines PACT-Programms übertragen, bedeutet das: Nutzern würde die Implementierung einfacher fallen, wenn sie anfangen PACT-Programme mental als Java-Code zu betrachten. Dies scheint kaum vorstellbar und würde zudem den Verlust der implementationsabhängigen Abstraktionsebene bedeuten. Sinnvoller ist es, die technische Schnittstelle zum Erstellen eines PACT-Programms in Richtung des Denkmodells

[2] Eine allgemeine Definition des Begriffs *Modell* findet sich im Glossar

[3] Bedienung eines Taschenrechners; Metapher des Schreibtischs bei der Computerbedienung

des Entwicklers anzupassen.

3.3 Graphisches Modellierungswerkzeug als Lösungsansatz

Um die in 3.2 beschriebene Diskrepanz zwischen dem mentalem Modell und der Java-basierten Implementierung zu minimieren, ist ein Werkzeug erforderlich, welches erlaubt, PACT-Programme als Datenflussgraph visuell zu modellieren. Mittels des gleichen Werkzeugs muss es für einzelne PACTs möglich sein, verschiedene Eigenschaften (wie zum Beispiel Output Contracts oder den Parallelisierungsgrad) zu definieren und UDFs mittels Java zu implementieren. Fertig modellierte PACT-Programme müssen dem PACT-Programmiermodell entsprechen und automatisch in ein für den PACT-Compiler verständliches Format überführt werden können. All dies sollte ohne die zeitaufwändige Einrichtung einer lokalen Entwicklungsumgebung (3.1 - Schritt 1) mittels eines Webbrowsers (2.3.2) möglich sein. Abbildung 3.2 stellt die vorgeschlagene graphische Werkzeugunterstützung bei der Erstellung eines PACT-Programms nochmals graphisch dar.

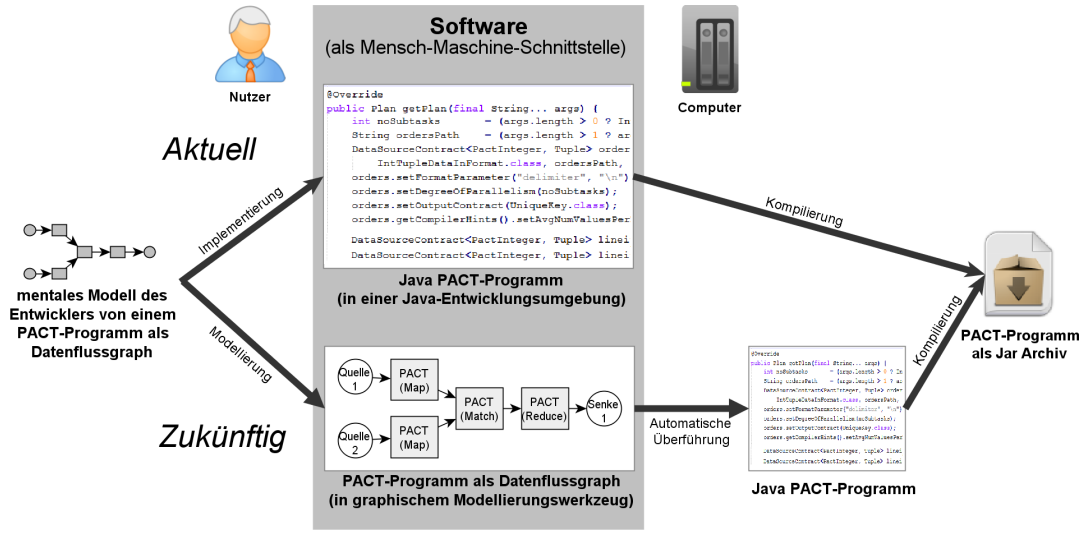


Abbildung 3.2: Erstellung eines PACT-Programms aktuell und zukünftig

Basierend auf diesen Grundanforderungen wird im nächsten Kapitel die Umsetzbarkeit des beschriebenen Werkzeugs evaluiert. Die Anforderungen werden detaillierter spezifiziert und konzeptionell analysiert. Unter Nutzung geeigneter Hilfsmittel der Softwareentwicklung wird abschließend die Problemlösung prototypisch implementiert. Sämtliche Schritte werden am Beispiel der Erstellung von PACT-Programmen beschrieben, lassen sich aber als generischer Lösungsansatz auf ähnliche Problemstellungen der visuellen Modellierung eines Datenflussgraphen übertragen.

Kapitel 4

Phasen der Softwareentwicklung

Das vorherige Kapitel hat die Problematik der Erstellung von PACT-Programmen erläutert. Es wurde ein Werkzeug zur visuellen Modellierung eines Datenflussgraphen sowie anschließenden automatischen Generierung von PACT-Programmen als Lösungsansatz motiviert. Dieses Kapitel beschreibt die prototypische Entwicklung dieses Werkzeugs in Form der Browser-basierten PACT-GUI-Webanwendung (2.3). Dabei ist nicht das Ziel den Projektverlauf oder die Implementierung im Detail darzustellen, sondern möglichst generische und auf ähnliche Anwendungsfälle übertragbare Lösungsansätze für die Problemstellung der visuellen, werkzeuggestützten Modellierung eines Datenflussgraphen zu entwickeln.

Dieses Kapitel ist entsprechend dem Wasserfallmodell[1] (2.4.1) gegliedert, da dieses sich auf Grund seines bekannten linearen Aufbaus gut zur Dokumentation eignet:

4.1 Planung: Auswahl eines geeigneten Vorgehensmodells zur Implementierung der PACT-GUI sowie die Definition von Anforderungen

4.2 Analyse: Implementierungsunabhängige, konzeptionelle Vorüberlegungen auf Basis der Anforderungen

4.3 Entwurf: Auswahl und Beschreibung geeigneter Technologien und Werkzeuge zur Umsetzung der Anforderungen

4.4 Implementierung: Beschreibung der Gesamtarchitektur sowie ausgewählter technischer Aspekte der PACT-GUI

4.5 Test: Qualitätssicherung durch automatisierte Softwaretests

4.1 Planung

Übliche Aktivitäten der Planungsphase, wie die Prüfung der fachlichen, personellen und ökonomisch sinnvollen Durchführbarkeit der Softwareentwicklung, sind bereits mit der

[1] Die ersten zwei Phasen wurden nach Balzert [Bal09] zu einer generelleren Planungsphase zusammengefasst. Die letzte Phase *Betrieb und Wartung* ist für den PACT-GUI-Prototypen nicht relevant.

Vergabe der Masterarbeit gegeben. Auch die Rollenverteilung im Projekt sowie ein Zeit- und Budgetplan sind somit vordefiniert. Offene Punkte, welche im Folgenden erarbeitet werden, sind die Wahl einer geeigneten Vorgehensweise zur Implementierung (4.1.1) und die Spezifikation von Anforderungen (4.1.2) an die PACT-GUI.

4.1.1 Wahl einer geeigneten Vorgehensweise

Die Basisvorgehensmodelle der Softwareentwicklung (2.4.1) beschreiben einen idealtypischen Projektverlauf und sind alleinstehend in ihrer Reinform nur in Ausnahmefällen anwendbar [Hof08, S. 491]. Das zu wählende Vorgehensmodell muss zu den mitwirkenden Personen sowie zur Art und Größe des Projekts passen. Vorrangiges Ziel muss sein, Prozesse zu schaffen, welche nicht nur ihrem Selbstzweck dienen, sondern von den Projektbeteiligten verstanden und *„gelebt"* werden. Aus diesem Grund setzt sich das gewählte Vorgehen für die Entwicklung der PACT-GUI aus verschiedenen Ansätzen zusammen.

Rahmenbedingungen
Von besonderer Bedeutung bei der Auswahl eines Vorgehensmodells zur Implementierung der PACT-GUI ist es, dass die Anforderungen an das zukünftige System nicht vollständig und detailliert erarbeitet werden können. Zum einen ist davon auszugehen, dass es über den Projektverlauf zu Änderungen im technischen Umfeld der PACT-GUI kommen wird, da sich verschiedene andere Teile des Stratosphere-Projekts (2.2) weiterhin in der Entwicklung befinden. Zum anderen existieren keinerlei Erfahrungswerte für den speziellen Anwendungsfall. Bisher werden PACT-Programme, wie in der Problemdefinition (3.1) geschildert, ausschließlich händisch programmiert. Wie dies graphisch umsetzbar und technisch realisierbar ist, kann im Vorfeld nur schwer abgeschätzt werden. *„I can't tell you what I want, but I'll know it when I see it"*[2] [Bal08] ist eine oft zitierte Aussage im Softwareumfeld, welche das Problem unbekannter oder nicht formulierbarer Anforderungen aus Auftraggebersicht verdeutlicht.

Vorgehensbeschreibung
Um flexibel auf neue Ideen und Änderungen der Projektumgebung reagieren zu können, ist das evolutionäre Vorgehen (2.4.1) mit Anleihen bei den agilen Methoden (2.4.2) geeignet.

Das folgende Unterkapitel 4.1.2 umfasst eine initiale Anforderungsdefinition, welche in der Analysephase weiter spezifiziert wird und als Ausgangspunkt zur Entwicklung der Nullversion (2.4.1 - evolutionäres Modell) dient. Der Schwerpunkt liegt über den gesamten Projektverlauf hinweg auf lauffähigen Softwareversionen. Mit ihnen können alle Projektbeteiligten bereits früh Erfahrungen sammeln und diese in die weitere Entwicklung einfließen lassen. Durch diese codegetriebene Entwicklung müssen alle Projektbeteiligten zwangsläufig enger als in anderen Modellen zusammenarbeiten und können mit jeder neuen Produktversion voneinander lernen. Um die in Unterkapitel 2.4.1 beschriebene Gefahr einer unzureichenden Systemarchitektur[3] zu minimieren, kommt ein Framework[3] entsprechend

[2] zu deutsch: Ich kann dir nicht sagen was ich will, aber ich werde es wissen, wenn ich es sehe.
[3] siehe Glossar

dem komponentenbasierten Modell (2.4.1) zum Einsatz. Zusätzlich wird so die Softwarequalität gesteigert und die Gesamtentwicklungszeit verringert. Weiterhin ist die Nutzung von Prototypen (2.4.1) zur technischen Evaluierung der Frameworks sowie zur frühen Demonstration der Benutzungsoberfläche geplant. Nachteile dieser Vorgehensweise wie die schlechte Planbarkeit des Gesamtprojektaufwands spielen im wissenschaftlichen Umfeld dieser Masterarbeit eine untergeordnete Rolle.

4.1.2 Anforderungsspezifikation

Eine Anforderung ist eine Beschaffenheit oder Fähigkeit, welche ein Softwaresystem erfüllen oder besitzen muss, um ein bestimmtes Problem lösen zu können [IEE90, S. 62]. Die Anforderungsspezifikation stellt die Gesamtheit der Anforderungen dar.

Im Wasserfallmodell wird die initiale Anforderungsspezifikation (=Lastenheft) [Bal09, S. 447] während der Ausschreibungs- oder Planungsphase erstellt. Sie beschreibt die fachlichen Basisanforderungen grob aus Auftraggebersicht. Primäres Ziel ist es, eine allgemeine Vorstellung des zu entwickelnden Systems und seiner Umgebung schriftlich auf einem hohen Abstraktionsniveau zu definieren. Detaillierte Anforderungen werden entsprechend dem evolutionären Vorgehensmodell (4.1.1) gemeinschaftlich mit allen Projektbeteiligten in Gruppendiskussionen auf Basis lauffähiger Produktversionen erarbeitet und sind in der Analyse (4.2) beschrieben.

Die Anforderungsspezifikation muss keiner bestimmten Form entsprechen, sollte aber inhaltlich mindestens die Visionen und Ziele, die Rahmenbedingungen, die Systemumgebung sowie die Softwareeigenschaften, getrennt in funktionale und nichtfunktionale Anforderungen, umfassen [Bal09, S. 456].

Visionen und Ziele
Bezogen auf die Softwareentwicklung ist eine Vision eine realitätsnahe Beschreibung der gewünschten zukünftigen Softwarelösung. Sie dient allen Projektbeteiligten während der Entwicklung als Leitgedanke. Ziele verfeinern und operationalisieren diese Vision anschließend. [Bal09, S. 457]

Vision: Um die Nutzung des Stratosphere-Systems zu vereinfachen, soll im Verlauf dieser Masterarbeit prototypisch ein Werkzeug namens PACT-GUI geschaffen werden, welches die effiziente Erstellung von PACT-Programmen ermöglicht.

Ziel 1: Der Benutzer kann PACT-Programme erstellen, speichern, laden, verändern, visualisieren, validieren, generieren und mit anderen austauschen.

Ziel 2: Sämtliche Funktionen der PACT-GUI sind als Webanwendung über den Browser zugänglich, um die Installation und Wartung weiterer Software (3.2 - Schritt 1) zu vermeiden.

Ziel 3: Für vorgebildete Benutzer soll die Programmbedienung selbsterklärend, intuitiv und effizient sein.

Ziel 4: Die Softwarearchitektur und die Implementierung des PACT-GUI-Prototypen sind nachhaltig und flexibel genug, um für zukünftige Anforderungen angepasst werden zu können.

Rahmenbedingungen
Rahmenbedingungen stellen technische oder organisatorische Beschränkungen für das Softwaresystem oder den Entwicklungsprozess dar [Bal09, S. 459]. Nachfolgend werden diese für die PACT-GUI aufgelistet.

Organisatorische Rahmenbedingungen:

R1 Der Anwendungsbereich der Software liegt vorerst im wissenschaftlichen, universitären Umfeld.

R2 Die Zielgruppe für die Verwendung der PACT-GUI sind Nutzer, die über Vorwissen bezüglich des Stratosphere-Projekts verfügen. Die Kenntnis des PACT-Programmiermodells und seine Anwendung wird ausdrücklich vorausgesetzt (Ziel 3).

R3 Die Softwarenutzung (oder auch Betriebsbedingung) erfolgt per Desktop-Computer. Die Benutzungsoberfläche speziell für kleine, berührungsempfindliche Bildschirme mobiler Endgeräte zu optimieren, ist nicht geplant.

R4 Eine Mehrsprachigkeit der Benutzungsoberfläche ist nicht erforderlich, da innerhalb von Statosphere Englisch als alleinige Projektsprache definiert ist.

R5 Die PACT-GUI kann von mehreren Benutzern gleichzeitig verwendet werden. Die synchrone Bearbeitung von PACTs ist jedoch nicht vorgesehen.

Technische Rahmenbedingungen:

R6 Zielumgebung für die PACT-GUI sind alle standardkonformen Desktop-Webbrowser (2.3.2). Für die derzeit von den Projektbeteiligten vor allem verwendeten Browser *Mozilla Firefox* und *Google Chrome* sollte die Anwendung wenn nötig speziell optimiert werden.

R7 Entsprechend Ziel 2 ist keine weitere Installation oder Wartung von zusätzlicher Software auf dem Computer und innerhalb des Webbrowsers wie Java Runtime Environment [Ora12a], Adobe Flash [Ado12] oder Microsoft Silverlight [Mic12b] nötig.

R8 Die PACT-GUI ist betriebssystemunabhängig, also lauffähig auf Linux-, Windows- und MacOS-Plattformen.

R9 Die Anwendung soll auf jedem handelsüblichen Computer ausführbar sein, also keine speziellen Anforderungen an die Hardware im Bezug auf Speicher oder Rechenleistung stellten.

Systemumgebung
Die materielle und immaterielle Umgebung des zukünftigen Softwaresystemes hat maßgeblichen Einfluss auf die Umsetzbarkeit der Anforderungen. Daher müssen die Systemgrenze klar festgelegt sein und Schnittstellen mit der Umgebung definiert werden. [Bal09, S. 461]

- Die PACT-GUI wird vorerst keine direkten Schnittstellen zu anderen Systemen innerhalb oder außerhalb des Stratosphere-Projekts erhalten. Eine Einbindung externer Systeme beispielsweise zur Authentifizierung, Ausführung von PACTs oder Datenspeicherung sollte zukünftig jedoch nicht ausgeschlossen sein (Ziel 4).

- Als Datenquelle und -senke während des Ladens und Speicherns von PACTs (Ziel 1) dient die lokale Festplatte.

Funktionale Anforderungen
Funktionale Anforderungen [Bal09, S. 456, 465] beschreiben die vom zukünftigen Softwaresystem bereitzustellenden Funktionen oder Dienste. Sie legen also fest, *was* ein System leisten soll. Dagegen beschreiben die nichtfunktionalen Anforderungen des nächsten Abschnitts, *wie gut* das System seine Funktionen mindestens erbringen soll.

In der agilen Softwareentwicklung werden bei Projektbeginn die Grundanforderungen möglichst knapp als sogenannte *User Stories*[4] dokumentiert. Eine User Story formuliert Wünsche an das zu entwickelnde System aus Nutzersicht - in der Form *„Ich wünsche..."* oder *„Ich brauche..."*. Dabei wird ein natürlichsprachlicher Ausdruck in der Terminologie des jeweiligen Anwendungsbereiches gewählt. Um die Qualität und Verständlichkeit der User Story nochmals zu erhöhen, empfiehlt sich wie bei der klassischen Formulierung von Anforderungen die Verwendung von Anforderungsschablonen. Dies sind Baupläne, welche die syntaktische Struktur einer Anforderung festlegen. [Rup07, S. 225ff]

Klassisch sollen Anforderungen so formuliert sein, dass sie vollständig, korrekt, konsistent, prüfbar, eindeutig, verfolgbar und bewertbar sind [IEE98]. User Stories hingegen setzten einen anderen Schwerpunkt. Nach dem *INVEST-Modell* sollen sie die folgenden Eigenschaften besitzen [Wat08]:

- **I**ndependent: Anforderungen sollen soweit wie möglich unabhängig voneinander sein, um sie getrennt bearbeiten, priorisieren oder verwerfen zu können.

- **N**egotiable: Eine User Story ist kein Vertrag und keine feste Spezifikation. Sie ist vielmehr eine Erinnerung für den Entwickler und den Kunden, die Details der Story zum Entwicklungszeitpunkt zu besprechen oder zu verhandeln.

- **V**aluable: Eine User Story ist wertvoll für den Auftraggeber. Sie stellt eine spätere Funktion dar und nicht Systeminterna oder Entwicklungsaufgaben.

- **E**stimatable: Der Implementierungsaufwand sollte grob abschätzbar sein.

- **S**mall: Die Anforderung sollte nicht zu umfangreich, sondern innerhalb weniger Wochen implementierbar sein.

- **T**estable: Die User Story sollte soweit konkretisiert sein, dass sie später testbar ist.

[4] zu deutsch: „Benutzergeschichten"

Für die PACT-GUI sind folgende User Stories definiert:

User Story 1: Als Benutzer muss ich vollständige, dem PACT-Programmiermodell entsprechende PACT-Programme erstellen können.

a) Das heißt, ich muss visuell einen PACT-Graphen modellieren können. Dazu gehört das Anlegen verschiedener Arten von Kommunikationskanälen, PACTs, Datenquellen und -senken.

b) Das heißt, ich muss Eigenschaften der PACTs wie den Namen und den Parallelisierungsgrad bearbeiten können.

c) Das heißt, ich muss UDFs (2.2.1) programmieren können.

d) Wünschenswert wäre, wenn externe Java-Bibliotheken und eigene Java-Hilfsklassen bei der Implementierung der UDFs genutzt werden können.

e) Wünschenswert wäre, wenn mehrere PACT-Programme gleichzeitig geöffnet und bearbeitet werden können, um die Wiederverwendung bereits erstellter UDFs zu erleichtern.

User Story 2: Als Benutzer muss ich PACT-Programme speichern und laden können.

Dafür muss das System die Möglichkeit bieten, ein erstelltes PACT-Programm persistente in eine Datei zu überführen und zu einem späteren Zeitpunkt das gleiche PACT-Programm aus dieser Datei wieder herzustellen.

User Story 3: Als Benutzer muss ich *„ausführbare"* PACT-Programme generieren können.

Das heißt, ich muss mit der PACT-GUI erstellte PACT-Programme validieren und in ein für den PACT-Compiler verständliches Format überführen können.

Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen (auch Qualitätsanforderungen) [Bal09, S. 463] legen qualitative oder quantitative Eigenschaften an eine Software fest. Sie beschreiben typischerweise Aspekte der Software, die mehrere oder alle funktionalen Anforderungen betreffen. Sie stehen somit orthogonal zu den funktionalen Anforderungen und haben daher großen Einfluss auf die Softwarearchitektur[3]. Es ist zu beachten, dass Qualitätsanforderungen oft in Konkurrenz oder Konflikt miteinander stehen. Anforderungen an die Sicherheit können sich beispielsweise negativ auf die Benutzbarkeit oder die Laufzeiteffizienz auswirken.

Die ISO Norm 9126 (=DIN 66272) kategorisiert Qualitätsmerkmale in 6 Qualitätskategorien [Rup07, S. 273ff]. Da die PACT-GUI keine speziellen Qualitätsanforderungen hat, priorisiert und kommentiert Tabelle 4.1 die Qualitätskategorien allgemein.

Qualitätskategorie (Teilmerkmale)	Relevanz
Funktionalität (Angemessenheit, Genauigkeit, Interoperabilität, Sicherheit) Die prototypische Implementierung der PACT-GUI soll vor allem die Umsetzbarkeit von Konzepten und die Eignung von Technologien prüfen. Der Funktionsumfang oder die Sicherheit spielen vorerst eine untergeordnete Rolle.	niedrig
Zuverlässigkeit (Reife, Fehlertoleranz, Wiederherstellbarkeit) Ein normales Level ist ausreichend, da es sich bei der PACT-GUI nicht um eine kritische Software handelt.	normal
Benutzbarkeit (Verständlichkeit, Erlernbarkeit, Bedienbarkeit, Attraktivität) Ein durchschnittlicher Grad an Benutzerfreundlichkeit ist ausreichend, da sich die PACT-GUI ausschließlich an fachkundige, IT-affine Benutzer richtet. (Rahmenbedingung R2)	normal
Effizienz (Zeitverhalten, Verbrauchsverhalten) Um ein flüssiges Arbeiten zu ermöglichen, sollte die Anwendung möglichst schnelle Reaktionszeiten auf Nutzereingaben bei geringem Verbrauch von Systemressourcen bieten. (Ziel 3; Rahmenbedingung 9)	hoch
Wartbarkeit (Analysierbarkeit, Änderbarkeit, Stabilität, Testbarkeit) Von besonders hoher Bedeutung, da die Weiterentwicklung der PACT-GUI über den Zeitraum dieser Masterarbeit hinaus geplant ist. (Ziel 4)	hoch
Übertragbarkeit (Anpassbarkeit, Installierbarkeit, Koexistenz, Austauschbarkeit) Keine besondere Relevanz, da die standardkonforme Implementierung der Applikation sowie die Nutzung des Browsers als Benutzungsschnittstelle bereits eine einfache Übertragbarkeit in andere Hardware- und Betriebssystem-Umgebungen erlaubt. (Ziel 2; Rahmenbedingung 6 und 8)	niedrig

Abbildung 4.1: Priorisierung von Qualitätsanforderungen nach ISO 9126

4.2 Analyse

In der Analysephase (oder auch Definitionsphase) wird auf Basis der Anforderungsdefinition (4.1.2) die fachliche Problemlösung aus Auftragnehmersicht (=Produktdefinition) erarbeitet [Bal09, S. 433ff]. Sie enthält das Pflichtenheft, welches bei der Produktabnahme zum Projektende als Referenz des vereinbarten Leistungsumfangs gilt. Nach Balzert [Bal09, S. 446] kann auf die Zweiteilung in Lasten- und Pflichtenheft verzichtet werden, wenn wie im Fall der PACT-GUI keine explizite Auftragnehmer- und Auftraggebersicht existiert.

Ziel dieses Kapitels ist daher nicht die Erstellung bestimmter Artefakte, sondern auf konzeptioneller Ebene Möglichkeiten zur Umsetzung der Anforderungen aufzuzeigen. Im Folgenden werden die drei definierten User Stories in je einem Unterkapitel detailliert analysiert.

4.2.1 User Story 1: GUI zur Erstellung von PACT-Programmen

Nach User Story 1 der Funktionalen Anforderungen (4.1.2) muss ein Benutzer mittels der Anwendungsoberfläche (GUI, 2.3.1) vollständige, dem PACT-Programmiermodel entsprechende PACT-Programme erstellen können. Die dazu nötige GUI wird in diesem Unterkapitel softwareergonomisch untersucht und mittels eines Prototyps konzipiert.

Softwareergonomische Betrachtung
Um die kognitive Belastung (3.3) bei der Programmbedienung so gering wie möglich zu halten, muss die GUI entsprechend Ziel 3 (4.1.2) selbsterklärend, intuitiv und effizient zu bedienen sein.

Softwareergonomie [Bal99, S. 653] heißt das Teilgebiet der Informatik, welches sich mit der „Usability" – zu deutsch Benutzbarkeit oder Gebrauchstauglichkeit – von Software als Schnittstelle zwischen Benutzer und Computer befasst. Nach der Industrienorm DIN EN ISO 9241 *„Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten"* ist ein System *gebrauchstauglich*, wenn ein Benutzer sein Ziel effektiv, effizient und zufriedenstellend erreicht. Das bedeutet, Benutzern soll die Bedienung möglichst einfach oder genauer gesagt mit möglichst geringer mentaler Anstrengung möglich sein [Nie03]. In Teil 110 der Norm „Grundlagen der Dialoggestaltung" werden sieben Kriterien definiert, um eine hohe Gebrauchstauglichkeit zu gewährleisten: 1) Aufgabenangemessenheit, 2) Selbstbeschreibungsfähigkeit, 3) Erwartungskonformität, 4) Fehlertoleranz, 5) Steuerbarkeit, 6) Individualisierbarkeit und 7) Lernförderlichkeit. Vergleichbar mit diesen Kriterien, jedoch konkreter formuliert, sind die 10 Usability-Heuristiken von Nielsen [Nie94]. Auf die Entwicklung der PACT-GUI bezogen, sind die folgenden Punkte zu beachten:

1. **Rückmeldungen:** Jede Aktion des Benutzers soll in angemessener Zeit zu einer Reaktion führen. Zu jedem Zeitpunkt soll der Benutzer wissen, was das System gerade macht.

2. **Ausdrucksweisen des Anwenders:** Die Anwendung sollte die Terminologie der Anwendungsdomäne verwenden. Im Fall der PACT-GUI sollten also die Fachbegriffe des PACT-Programmiermodells verwendet werden (Rahmenbedingung 2).

3. **Fehlertoleranz:** Anwender lernen auch durch falsche Bedienung. Sie sollten keine Angst vor Fehlbedienung haben und stets unerwünschte Aktionen rückgängig machen können.

4. **Gute Fehlermeldungen:** Fehlermeldungen sollten klar das Problem beschreiben und wenn möglich immer einen Lösungsvorschlag anbieten.

5. **Fehlervermeidung:** Noch besser als eine gute Fehlermeldung ist es, Fehler gar nicht erst zuzulassen. So sollte die Bearbeitung von PACTs nur entsprechend dem PACT-Programmiermodell möglich sein (Ziel 1).

6. **Konsistenz:** Die gesamte PACT-GUI sollte einheitlichen Standards folgen. Die GUI-Bedienung sollte erwartungskonform sein und möglichst auf bereits von anderen

Systemen bekannte Bedienkonzepte zurückgreifen. So sollten Beschriftungen stets das Gleiche bedeuten und Steuerelemente[3] sich stets gleich verhalten (Ziel 3).

7. **Minimale mentale Belastung:** Alle nötigen Objekte, Aktionen und Optionen sollten, wann immer nötig, sichtbar oder zugänglich sein. Der Benutzer sollte nicht gezwungen sein, sich Informationen und Instruktionen zu merken (Ziel 3).

8. **Sonderfunktionen für erfahrende Benutzer:** Die PACT-GUI sollte möglichst mehrere Wege einer Interaktion bieten, um ebenso erfahrene und unerfahrene Nutzer zu unterstützen. Beispielsweise können Tasten-Kombinationen die Interaktion für erfahrene Benutzer beschleunigen, ohne dass Neulinge davon anfangs verwirrt werden.

9. **Ästhetik und minimalistisches Design:** Ein minimalistisches, klar strukturiertes auf die nötigen Informationen beschränktes Design hilft die Bedienung zu vereinfachen und die Aufmerksamkeit des Nutzers komplett auf die Erledigung der jeweiligen Aufgabe zu fokussieren (Ziel 3).

10. **Hilfe und Dokumentation:** Eine gute GUI kann ohne Hilfen und Dokumentation genutzt werden. Im Bedarfsfall sollten sie aber einfach zugänglich sein und dem Anwender durch kurze, aber konkrete Hinweise helfen (Rahmenbedingung 2).

Konzeption der GUI

Abbildung 4.2 stellt in Form eines UML-Anwendungsfalldiagramm[3] die in User Story 1 enthaltenen Aktivitäten dar.

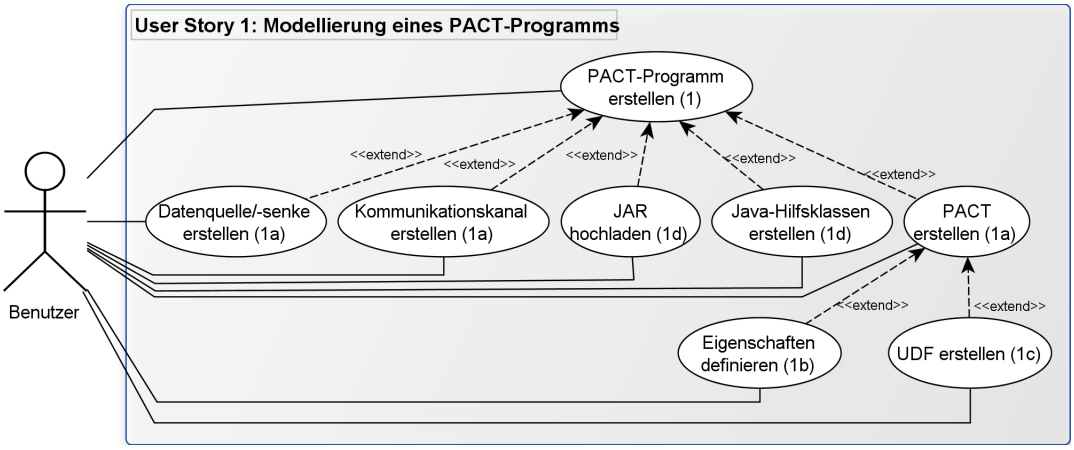


Abbildung 4.2: UML-Anwendungsfalldiagramm der User Story 1

Um eine graphischen Benutzungsschnittstelle zu konzipieren, welche diesen Anforderungen entspricht, empfiehlt Balzert ein zweistufiges Vorgehen [Bal99, S. 687].

Schritt 1: In einem ersten Schritt sollten die Anforderungen von der allgemeinsten zur speziellsten[5] analysiert werden, um eine *Grobskizze* der Anwendungsoberfläche zu erstellen. Diese enthält Vorgaben bezüglich der Aufteilung der Arbeitsfläche sowie der zu verwendenden Steuerelemente[3] und beschreibt den Wechsel zwischen verschiedenen Dialogen.

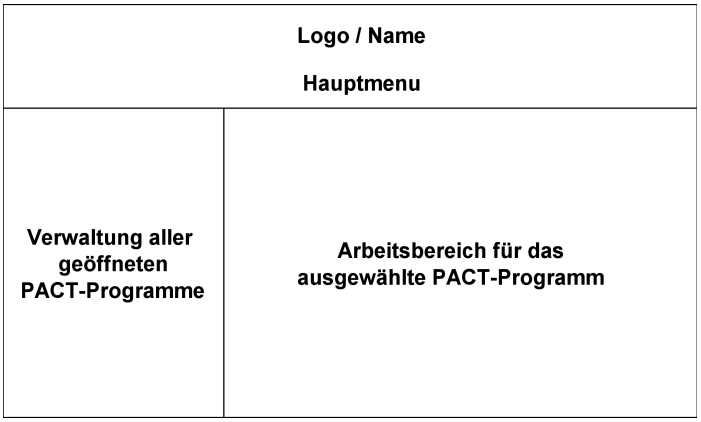


Abbildung 4.3: Entwurf der GUI-Strukturierung

Entsprechend Abbildung 4.3 wird die Anwendungsoberfläche dreigeteilt:

- *Der Kopfbereich* enthält den Programmnamen und ein aufklappbares (und dadurch platzsparendes und zukünftig beliebig erweiterbares) Hauptmenü zur Programmsteuerung. Das Menü enthält Einträge um PACT-Programme zu erstellen und zu schließen (User Story 1), zu speichern und zu laden (User Story 2) sowie zu generieren (User Story 3).

- Nach User Story 1e soll es möglich sein, gleichzeitig mehrerer PACT-Programme zu bearbeiten. Denkbar wäre den Wechsel zwischen verschiedenen PACT-Programmen per Hauptmenü zu realisieren. Wenn aber gleichzeitig alle geöffneten PACT-Programme und deren Eigenschaften dargestellt werden sollen, dann bieten sich *ein Verwaltungsbereich für PACT-Programme* (links) in Form eines Reiter-, Baum- oder Akkordeon-Steuerelements (Anhang A.5) an.

- Der zentrale Arbeitsbereich erlaubt ein ausgewähltes PACT-Programm entsprechend User Story 1 (Abbildung 4.2) zu bearbeiten.

Die Aufteilung der Oberfläche in einen Kopf-, einen Verwaltungs- und einen Arbeitsbereich ist bewusst an bekannte Applikationen wie den Microsoft Windows Explorer[6], die VmWare Workstation [VMw12], die Eclipse IDE (4.3.4) oder Datenbankverwaltungssoftware wie das IBM Data Studio [Int12] angelehnt, um die Einarbeitung zu erleichtern.

[5] bekannt als Top-Down-Methode [Bal09, S. 55]

[6] Datei- und Verzeichnisverwaltungsprogramm von Microsoft Windows http://windows.microsoft.com

Schritt 2: Die Details der Benutzungsoberfläche sollen nach Balzert anschließend in einem zweiten Schritt mit Hilfe eines lauffähigen Front-End-Prototypen iterativ verfeinert werden. Ein solcher horizontalen Prototyp (2.4.1, auch *Click-Dummy* genannt) bildet die Benutzungsoberfläche im Hinblick auf Aussehen, Interaktionsgeschwindigkeit und Bedienung ab. Funktionalitäten hinter den Bedienelementen werden nicht oder nur ansatzweise implementiert.

Für die Erstellung des Prototypen wurde *Ext.js* [San12a] verwendet. Diese JavaScript[3]-Programmbibliothek (2.3.2, 2.4.1) bietet eine Vielzahl von Steuerelementen[3], um modular eine Webbrowser-basierte GUI zu entwickeln. Ein Webserver, eine Entwicklungsumgebung oder die Kompilierung von Quellcode sind nicht erforderlich. Durch kontinuierliches Einarbeiten der Vorschläge aller Projektbeteiligten wurde ein Prototyp der PACT-GUI entwickelt. Dieser besteht aus nur einer, leicht verteilbaren HTML-Datei mit rund 1100 Zeilen (30KB) HTML, JavaScript und CSS (2.3.2) und kann so ohne weiteres auf jedem Computer getestet werden.

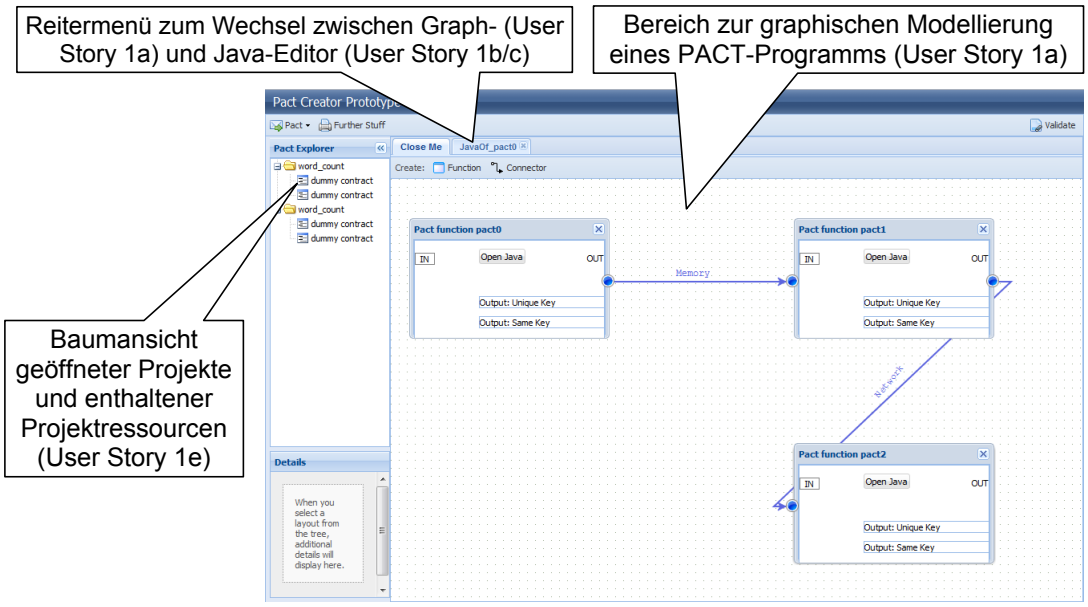


Abbildung 4.4: GUI-Prototyp mit Graph-Editor-Ansicht

Abbildung 4.4 zeigt die Oberfläche des JavaScript-Prototypen mit einem Verwaltungsbereich für alle geöffneten PACT-Programme links sowie dem graphischen Modellierungsbereich (Graph-Editor) für das geöffnete PACT-Programm rechts. Datenkanäle, PACTs, Datenquellen und -senken können mittels einer Symbolleiste erstellt werden. Per Drag & Drop[7] werden die Elemente auf der Arbeitsfläche frei angeordnet und PACTs mittels Datenkanälen verbunden. In der fertigen PACT-GUI ist bei jedem Modellierungs- und Editierungsschritt zu prüfen, ob diese entsprechend dem PACT-Programmiermodell erlaubt

[7]Dieser Fachbegriff bezeichnet das Aufnehmen und Fallenlassen von GUI-Objekten mittels eines Zeigegeräts wir der Maus.

sind (4.2.1 - Nielsen Usabilty Heuristik 5). Folgende Punkte sind zu prüfen:

- Beim Verknüpfen von zwei PACTs durch einen Datenkanal dürfen keine Kreise (Zyklenfreiheit 2.2.1) oder doppelte Verbindungen entstehen. Keiner der beiden PACTs darf also bereits ein Nachfolger (2.2.1) des anderen sein.
- Datenquellen können keine eingehenden Verbindungen haben; Datensenken keine ausgehenden.
- PACT-Programm-Namen müssen mit einem Großbuchstaben beginnen.
- PACT-Programm-Namen und PACT-Namen dürfen keine Sonderzeichen enthalten.

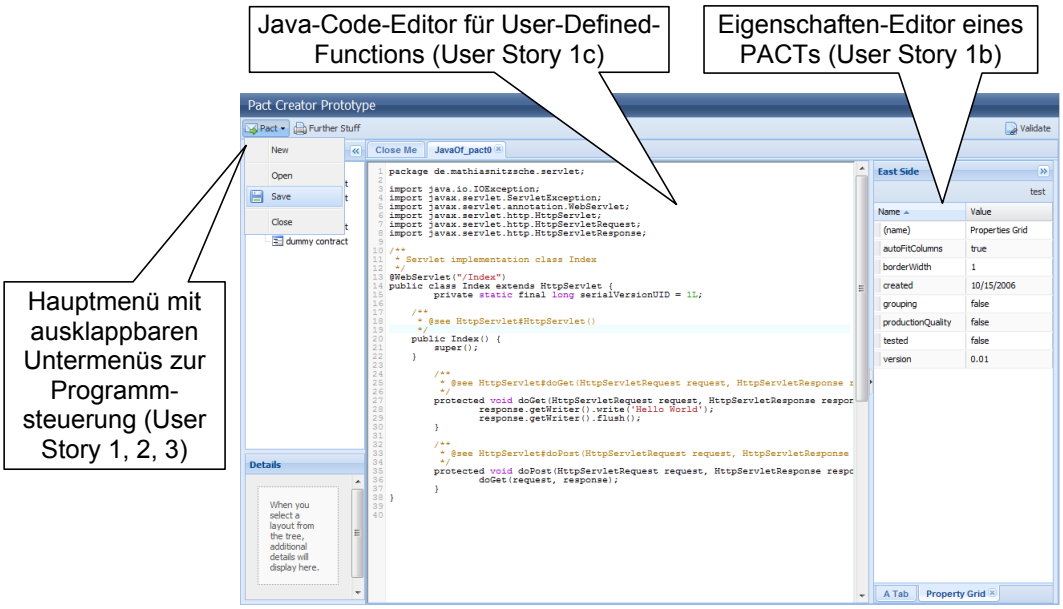


Abbildung 4.5: GUI-Prototyp mit UDF-Editor-Ansicht

Abbildung 4.5 zeigt das aufklappbare Hauptmenü oben sowie einen Java-Editor mittig und einen Eigenschaften-Editor rechts im Arbeitsbereich der GUI. Der Java-Editor dient dazu die UDF eines PACTs (User Story 1c) und die Java-Hilfsfunktionen (User Story 1d) zu editieren. Um die Bearbeitung des Java-Quelltextes für den Endnutzer so einfach und angenehm wie möglich zu gestalten, wurde anstelle eines einfaches Textfelds mit der JavaScript-Bibliothek *CodeMirror* [Hav11] ein vollwertiger Quelltexteditor verwendet. Dieser umfasst bekannte Funktionen wie Syntaxhervorhebung, Zeilennummerierung, Undo/Redo[8] und automatische Einrückung. Der Eigenschaften-Editor erlaubt die Eigenschaften eines PACTs (Name, Parallelisierungsgrad) zu bearbeiten (User Story 1b). Das Umschalten zwischen dem Graph-Editor (Abbildung 4.4) und den Editoren eines einzelnen PACTs

[8]Die Undo-Funktion macht die letzte Eingabe rückgängig. Redo hebt den letzten Undo-Aufruf auf, führt den Befehl also erneut aus.

(Abbildung 4.5) erfolgt mittels eines Reitermenüs, wodurch es möglich ist, beliebig viele PACTs gleichzeitig zu editieren. Andere Realisierungsoptionen wie PopUp-Fenster oder die Aufteilung der Graph-Editor-Fläche hätten dies nicht oder nur sehr unübersichtlich ermöglicht.

4.2.2 User Story 2: Datenpersistenz

Mit der PACT-GUI erstellte PACT-Programme existieren nur im Arbeitsspeicher des Browsers und gehen verloren, sobald dieser geschlossen wird. Daher ist es notwendig entsprechend User Story 2 erstellte Modelle persistent speichern und wieder laden zu können. Eine Datenbanklösung wäre zu aufwändig und erlaubt zudem keine einfache Übertragung zu anderen Personen. Die Datenpersistenz für die PACT-GUI sollte daher durch Objektserialisierung mittels Dateien realisiert werden. Serialisierung ist die Abbildung von internen Objekten und deren Zuständen auf eine externe sequenzielle Darstellungsform [RAS, S. 237]. Sie erlaubt verschachtelte Objektstruktur als Zeichenkette in Dateien zu speichern. Durch den umgekehrten Weg, die Deserialisierung, können diese Objekte zu einem späteren Zeitpunkt wieder hergestellt werden. Unterkapitel 4.4.2 wird als Teil der Implementierung die gewählte Technik zur Serialisierung vorstellen.

4.2.3 User Story 3: Generierung von PACT-Programmen

PACT-Programme können entsprechend dem PACT-Programmiermodell erstellt (4.2.1) und mittels eines PACT-GUI-spezifischen Dateiformates gespeichert und geladen (4.2.2) werden. Wie beschrieben handelt es sich bei dem Speicherformat nur um die PACT-GUI-spezifische Serialisierung des PACT-Programms und nicht um eine Generierung entsprechend dem in Unterkapitel 3.1 beschriebenen Prozess.

Daher soll es nach User Story 3 möglich sein, modellierte PACT-Programme in ein für den PACT-Compiler verständliches Format zu überführen, um die Ausführung des PACT-Programms auf Nephele (2.2.3) zu ermöglichen. Im Folgenden werden implementierungsunabhängig die nötigen Schritte beschrieben.

Modell zum Server übertragen
Die Generierung von PACT-Programmen ist nicht im Browser möglich. Sie muss auf einem Server stattfinden, da Kommandozeilen-Programme ausgeführt werden müssen (3.1 - Schritt 7 und 8). Sollte das PACT-Programm nur im Browser existieren, muss es zuerst zum Server übertragen werden.

Statische Validierung
Um die Anzahl nicht erfolgreicher Generierungsversuche zu minimieren, sollte das Modell des PACT-Programms zuerst statisch validiert werden. Ein PACT-Programm-Graph muss mindestens folgende Eigenschaften erfüllen, um vollständig und valide zu sein:

- Das PACT-Programm enthält mindestens eine Datenquelle und eine Datensenke.
- Der Start jedes Datenkanals ist ein PACT oder eine Datenquelle.

- Das Ende jedes Datenkanals ist ein PACT oder eine Datensenke.
- Jede Quelle hat mindestens einen ausgehenden Datenkanal; jede Senke hat mindestens einen eingehenden Datenkanal.
- Jeder Eingang eines PACTs hat mindestens einen eingehenden Datenkanal; jeder Ausgang mindestens einen ausgehenden Datenkanal.
- Die Java-Klasse der UDF eines PACTs hat den gleichen Namen wie der PACT und implementiert ein Interface entsprechend seinem Input Contract.

Andere Eigenschaften des Modelles wie die Zyklenfreiheit des Graphen oder die Einhaltung von Namenskonventionen werden bereits während der Erstellung durch die PACT-GUI sicher gestellt (4.2.1 - Konzeption der GUI).

Generierung vorbereiten
Wenn das PACT-Programm erfolgreich validiert wurde, muss im nächsten Schritt ein temporäres Arbeitsverzeichnis auf der Festplatte des Servers für die Generierung erstellt werden. Eine Struktur von Unterverzeichnissen muss anschließend entsprechend Abbildung 4.6 vorbereitet werden (3.1 - Schritt 3). Mittels des PACT-GUI hochgeladene externe Bibliotheken (.jar-Dateien) werden in das *lib*-Verzeichnis geschrieben (3.1 - Schritt 4). Eine Manifest-Datei wird im *META-INF*-Verzeichnis angelegt (3.1 - Schritt 5).

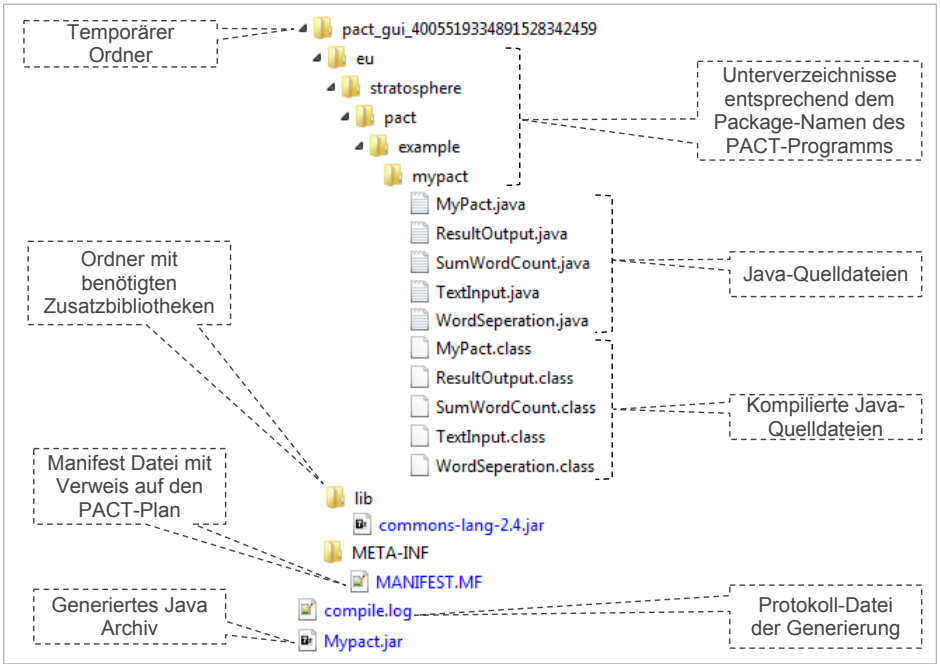


Abbildung 4.6: Temporäres Verzeichnis nach der Generierung eines PACT-Programms

Java-Dateien erstellen
Die als Klassen definierten UDFs der PACTs (User Story 1c) und die Java-Hilfsklassen

(User Story 1d) werden nun als Java-Dateien in das temporäre Verzeichnis gespeichert. Zusätzlich wird auf Basis des modellierten PACT-Graphen (User Story 1a) und definierter Eigenschaften der PACTs (User Story 1b) eine zentrale PACT-Plan-Java-Datei (Programmierbeispiel 3.1) angelegt. An dieser Stelle muss der zuvor visuell modellierte PACT-Graph inklusive aller PACT-Eigenschaften automatisiert in Java Anweisungen überführt werden, die für den PACT-Compiler verständlich sind (3.1 - Schritt 6).

Kompilieren und archivieren
Entsprechend Schritt 7 und 8 (3.1) werden die Java-Quellen kompiliert und anschließend alle benötigten Dateien in einem Java-Archiv zusammengefasst.

Ergebnis / Problembehandlung
Während der Generierung sollte ein umfangreiches Protokoll erzeugt werden, das dem Nutzer nach Abschluss der Generierung angezeigt wird. Es hilft etwaige Fehler während der Validierung oder der Generierung zu erkennen. Zudem sollte es möglich sein alle im temporären Verzeichnis erzeugten Dateien zu betrachten beziehungsweise vom Server herunterzuladen.

4.3 Entwurf

In der *Entwurfsphase* wird, in Abgrenzung zu den beiden vorangegangenen Phasen *Planung* (4.1) und *Analyse* (4.2), welche definieren, was entwickelt werden soll, erstmals das *Wie* der fachlichen Umsetzung betrachtet. Es werden technische Alternativen abgewogen und Grundsatzentscheidungen beispielsweise bezüglich der Programmiersprache, eines Anwendungsframeworks, der Art der Datenhaltung oder der Verteilung von Komponenten getroffen. Auf dieser Basis erfolgt anschließend ein detaillierter Systementwurf unter Nutzung von Hilfsmitteln wie UML-Klassendiagrammen[3] [Bal09, S.133] oder Entity-Relationship-Modellen [Bal09, S.199].

Dieses Vorgehen ist aber wie in Unterkapitel 2.4.1 beschrieben, nur möglich, wenn alle Anforderungen bekannt sind, konstant bleiben und das Entwicklerteam ein hohes Maß an Kenntnis der verwendeten Techniken besitzt. Für die Entwicklung der PACT-GUI trifft keine dieser Anforderungen zu, weshalb sich dieses Kapitel auf die fundierte Begründung getroffener technischer Grundsatzentscheidungen beschränken wird. Wie bei der agilen Softwareentwicklung üblich, wird ein qualitativ hochwertiges Endprodukt nicht durch einen umfangreichen Systementwurf angestrebt, sondern durch sorgfältige Auswahl der richtigen Softwarekomponenten/werkzeuge, regelmäßige Codereviews, stetiges Refactoring sowie einer hohen Testautomatisierung [Bal08, S. 651ff].

Einleitend wird die Wahl der Programmiersprache, des Ausführungsservers (4.3.1) und des Applikationsframeworks (4.3.2) begründet. Anschließend werden das gewählte Framework (4.3.3) und weitere Werkzeuge (4.3.4), die bei der Implementierung zum Einsatz kommen, beschrieben.

4.3.1 Programmiersprache und Ausführungsumgebung

Im Fall der PACT-GUI musste explizit keine Programmiersprache zur Entwicklung der Webanwendung gewählt werden. Das von Sun Microsystems entwickelte Java [Ora12a] ist bereits für das gesamte Stratosphere-Projekt als Standardprogrammiersprache definiert. Java wurde aus den folgenden Gründen gewählt:

- Java ist lizenzkostenfrei und quelloffen,
- Java ist eine voll objektorientierte, typisierte moderne Sprache,
- Viele Frameworks (4.3.2), Bibliotheken (2.4.1) und Entwicklertools (4.3.4) sind vorhanden,
- Java wird bereits während des Grundstudiums gelehrt, weshalb Studenten und Diplomanden einfacher zu finden und schnell einzuarbeiten sind.

Als Ausführungsumgebung für die PACT-GUI-Webanwendung wurde darüber hinaus Jetty [Mor11] vorgegeben, da dieser Webserver (2.3.1) bereits in anderen Teilen des Stratosphere-Projekts (3.1 - *Nephele and PACTs Query Interface*) verwendet wird. Jetty umfasst einen Java-basierten vollständigen HTTP1.1 Webserver (2.3.1) sowie einen Servlet-Container.

Ein Servlet-Container[9] [Com07] ist eine Software-Komponente zur Verwaltung von Servlets [Com07] und ihren Ressourcen über ihren kompletten Lebenszyklus (von der Erstellung bis zur Beendigung) hinweg. Ein Servlet ist eine Java-Klasse, welche die Java Servlet API[3] [Com07] implementiert. Über diese API kann ein Servlet an sich gestellte Anfragen analysieren und dynamisch eine Antwort generieren. Im einfachsten Fall ist der Inhalt dieser Antwort eine HTML Seite – möglich sind aber auch andere Textformate (wie beispielsweise XML[3]- oder JavaScript[3]-Dateien) sowie Binärformate (wie beispielsweise Bilder oder PDFs[3]). Um die Erstellung von textuellen Antworten zu vereinfachen, bietet der Servlet-Container auch die Möglichkeit Java Server Pages (JSP) zu nutzen. Diese Technologie baut auf Servlets auf und bietet eine bessere Trennung von Logik und (HTML-)Markup. [FJ04, S. 32ff, 109ff]

Bei der Entwicklung muss die volle Kompatibilität mit anderen Servlet-Containern wie denen des Apache Tomcats [The11b] oder des JBoss-Applikationsservers [Red11] gewährleisten werden, um eine spätere Migration zu ermöglichen. Jetty wurde diesen vorgezogen, da er weniger Festplatten- und Arbeitsspeicher benötigt, aber dennoch sehr performant ist [Wil08].

4.3.2 Auswahl eines Applikationsframeworks

Motivation

Mit den bis hierhin vorgestellten Technologien und Konzepten wäre die PACT-GUI bereits realisierbar. Eine Java-Webanwendung (2.3.1) würde auf dem Jetty-Webserver laufen

[9] auch Web-Container genannt

und mittels Servlets/JSPs (4.3.1) Webseiten nach dem Client/Server-Modell (2.3.1) bereitstellen. Innerhalb dieser Webseiten würden die Browsertechnologien HTML, CSS und JavaScript (2.3.2) zum Einsatz kommen, um eine GUI (2.3.1) zur Bewältigung aller Aufgaben zu generieren. Das Problem an diesem Ansatz ist, dass er den zeitlichen Rahmen dieser Arbeit aus folgenden Gründen deutlich überschreiten würde:

- **Browser-Inkompatibilität**[10]: Verschiedene Browser und teilweise sogar gleiche Browser in verschiedenen Versionen verhalten sich bei der Interpretation von HTML, JavaScript und CSS (2.3.2) teils sehr unterschiedlich [Koc11]. Der Aufwand, für jeden Browser speziell zu implementieren und separat zu testen, wäre groß.
- **Desktop „Look and Feel"**: Steuerelemente wie ein Reitermenü oder ein Baum sind standardmäßig im Browser nicht vorhanden und müssen komplett selbst entwickelt werden. Die Erstellung einer umfangreichen Rich-Client-Webanwendung (2.3.1), welche einer Desktopanwendung in Verhalten und Aussehen ähnelt, ist daher sehr komplex und zeitaufwändig.
- **„Boilerplate code"**: Mit diesem Begriff werden Programmcode und Konfigurationsanweisungen bezeichnet, die nicht zur Erfüllung des eigentlichen Anwendungszwecks beitragen, aber dennoch für das Funktionieren der Anwendung nötig sind. Nur mit den oben genannten Technologien müsste in großem Umfang und oftmals wiederholt solcher Code geschrieben werden, was unnötig Zeit kostet. Beispiele sind: Servlet-Konfiguration, Ajax-Code, Logging und Mehrsprachigkeit.
- **Wartung und Restrukturierung**[11]: Auf Grund der vielen verwendeten Technologien, aber auch weil nicht alle verwendeten Programmiersprachen typisiert sind und von der Entwicklungsumgebung (4.3.4) validiert werden können, wird die Codebasis mit zunehmender Projektgröße schwerer zu verwalten.

Die Umsetzung des GUI-Prototyps hat gezeigt, dass die ersten beiden Punkte durch Front-End-Bibliotheken wie Ext.js (4.2.1) gelöst werden können. Dabei hat sich jedoch auch bestätigt, dass mit zunehmendem Projektumfang diese Lösung auf Grund des letzten Punkts schnell an ihre Grenzen stößt – insbesondere da im Prototypen nur das Front-End ohne jegliche Back-End-Anbindung umgesetzt wurde. Um die nötige Zeit und Komplexität der Entwicklung zu reduzieren, sollte daher ein integriertes server- und clientseitiges Framework entsprechend dem komponentenbasierten Ansatz (2.4.1) verwendet werden.

Frameworks stellen einen wiederverwendbaren generischen Entwurf eines Softwaresystemes für eine Klasse von Anwendungsfällen bereit. Sie geben die Struktur einer Anwendung vor, indem sie den Aufbau, Abhängigkeiten und allgemein Programmabläufe definieren. Sie bilden den Rahmen für den anwendungsspezifischen Code des Entwicklers

[10] Es handelt sich hier um einen gängigen Fachbegriff, wobei eigentlich nicht die Webbrowser inkompatibel sind, sondern sich nur ihre Interpretation von Webstandards unterscheidet. Für Details zu den Unterschieden sei auf weiterführende Literatur unter den Stichworten JavaScript capability/user-agent detection, Polyfills, gracefully degrading, Quirks Mode, Conditional Comments, DocTypes, Browserweichen und Cross-Browser-Testing verwiesen.

[11] englisch „Refactoring": Prozess der Überarbeitung von Quellcode

und stellen eine Vielzahl weiterer Funktionen bereit. Im Fall moderner Java-Applikationsframeworks kann dies unter anderem die folgenden Punkte umfassen: Anfrage/Antwort-Applikationsfluss (2.3.1), Datenbankzugriff, Protokollierung, Ereignis- und Fehlerbehandlung, Caching, Konfiguration, Kommunikation, Anwendungsüberwachung und -tests, Sicherheitsmechanismen, Präsentation der Benutzungsschnittstelle. [Nas03, S. 12ff, 54ff]

Alternativen
Vergleiche zwischen der Vielzahl existierender Java-Applikationsframeworks (Wikipedia führt beispielsweise 37 verschiedene auf [Wik11]) sind unter anderem in [Ngu10], [Zor10] und [Wä10] zu finden. Abbildung 4.7 zeigt in Anlehnung an [Wä10] eine Übersicht der populärsten Frameworks. Sie sind gegliedert in verschiedene Framework-Klassen und dargestellt in Bezug auf ihre Komplexität sowie den Grad ihrer Unterstützung bei der Erstellung einer Rich-Client-Webanwendung (2.3.1).

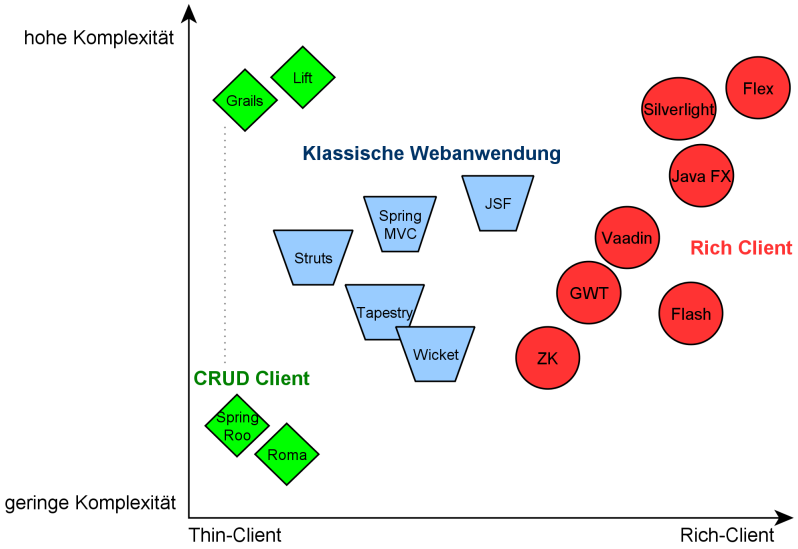


Abbildung 4.7: Vergleich verschiedener Java-Applikationsframeworks

Um die Framework-Auswahl zu erleichtern, werden zuerst alle Frameworks-Klassen ausgeschlossen, die zu den Anforderungen der PACT-GUI im Widerspruch stehen oder in ihrem Funktionsumfang nicht geeignet erscheinen:

- **CRUD[12]-Frameworks**: Eine spezielle Klasse von Frameworks für die Erstellung von Applikationen, bei denen Aktionen der Anwendungsoberfläche direkt Datenbankobjekte erstellen, lesen, verändern oder löschen[12]. Da die PACT-GUI keine Datenbank nutzt (4.2.2), werden CRUD-Frameworks nicht weiter betrachtet.
- **Klassische Frameworks für Webanwendungen**: Trotz verschiedener Stärken

[12]CRUD: Create, Read, Update, Delete

und Ausrichtungen erscheinen alle Frameworks in diesem Bereich hervorragend geeignet für die Erstellung umfangreicher Webanwendungen. Keines der betrachteten Frameworks kann jedoch serverseitig eine komplette Rich-Client-Webanwendung implementieren und clientseitig generieren. Für die PACT-GUI wären sie somit nur geeignet, wenn eine zusätzliche Front-End-Bibliothek wie Ext.js (4.2.1) verwendet wird oder HTML, CSS und JavaScript (2.3.2) selbst programmiert wird, was wiederum zu den in der Motivation erwähnten Problemen führt.

- **Rich-Client-Frameworks**: (2.3.1) Diese teils sehr umfangreichen Frameworks sind besonders geeignet für die Erstellung von interaktionsintensiven, desktopähnlichen Benutzungsschnittstellen mit Webtechnologien. Sie zeichnen sich durch eine sehr gute Unterstützung bei der Implementierung einer cross-browser-kompatiblen GUI (2.3.1) innerhalb des Browsers aus. Die Programmierung von HTML, CSS und JavaScript ist nur in Ausnahmefällen nötig.

Auswahl
Für die umfangreichen Modellierungsaufgaben innerhalb der Anwendungsoberfläche der PACT-GUI sind Rich-Client-Frameworks aus folgenden Gründen am besten geeignet: [Ker10, S. 1f] [CC08, S. 3f]

- **Front-End-Fokus:** Rich-Client-Anwendungen sind darauf ausgelegt, so viel Logik wie möglich im Browser auszuführen. Der Server wird nur noch genutzt, um die Applikation auszuliefern und um Funktionen bereitzustellen, die aus technischen Gründen innerhalb des Browsers nicht möglich sind, wie zum Beispiel die Interaktion mit einer Datenbank. Eine geringere Serverlast durch weniger Anfragen sowie schnellere Reaktionszeiten der GUI sind das Ergebnis.

- **Integration:** Für das Front-End, das Back-End (2.3.1) und die Kommunikation zwischen beiden (2.3.2) wird eine zentrale Codebasis entwickelt. Datenmodelle, Entwurfsmuster, Tests und vieles andere können somit über die ganze Applikation hinweg einheitlich genutzt werden.

- **Cross-Browser-Unterstützung:** Browser-Inkompatibilitäten (siehe Motivation) spielen bei der Entwicklung keine Rolle, da sie durch das jeweilige Framework abstrahiert werden.

Die letzten beiden Eigenschaften werden von den meisten Rich-Client-Frameworks durch Zusatzsoftware wie Microsoft Silverlight [Mic12b], Adobe Flash [Ado12] oder Java Runtime Environment [Ora12a] ermöglicht. Dies widerspricht jedoch Rahmenbedingung 7 (4.1.2), die Installation oder Konfiguration von zusätzlicher Software zu vermeiden. Für die weitere Betrachtung kommen daher nur die ohne Zusatzsoftware nutzbaren Frameworks Google Web Toolkit [Goo11c] und Vaadin [Vaa11] infrage. Beide Kandidaten wurden mittels einer weiterführender Internetrecherche und Prototyping entsprechend gängiger Kriterien für die Evaluation von Frameworks [Nas03, S. 215ff] [For03, S. 311ff] weiter untersucht:

- **Eignung für den Anwendungsfall und Frameworkqualität** *(Funktionsumfang, Komplexität, Erweiterbarkeit, Flexibilität, Architekturkonzept, Werkzeugunterstützung, Innovationen):* Beide Frameworks erschienen als sehr ausgereift und passend für die Erstellung der PACT-GUI. Vaadin hat einen größeren Funktionsumfang (Codebasis ist drei mal größer als bei GWT). GWT ist im Vorteil, weil es keine weiteren Abhängigkeiten hat, während Vaadin sogar auf GWT aufbaut.

- **Dokumentation und Erlernbarkeit** *(Entwicklereinführungen, JavaDocs, offener Quellcode, professionelle Beratung):* Der Einstieg in beide Frameworks wird dem Entwickler durch umfangreiche Dokumentation sowie viele Zusatzmaterialien wie Videos, Podcasts und Beispielprojekte so weit wie möglich erleichtert.

- **Externe Kriterien** *(lebendige Entwicklergemeinschaft, starke Framework-Herstellerfirma, Erfolgsgeschichten, Zukunftsperspektive):* Der Vergleich der Herstellerfirmen Google Inc. und Vaadin Ltd sowie die deutlich größere und aktivere GWT-Nutzerschaft sprechen hier klar für GWT (Details siehe Anhang A.3).

- **Weiche Faktoren** *(Gefühl des Entwicklers, Vorwissen):* Der Autor dieser Arbeit hatte mit keinem der Frameworks Erfahrungen, jedoch bereits mehrfach in seinem Arbeitsumfeld von der erfolgreichen, professionellen Verwendung von GWT erfahren.

Den Ausschlag für die Wahl von GWT hat letztendlich eine mit Fachartikeln [Sto07] und Vortragsvideos [ML08] sehr gut dokumentierte Erfolgsgeschichte der Entwicklung von *„IBM Blueworks Live"*[13] [IBM11a] gegeben. IBM Blueprint (Screenshots der Anwendung in Anhang A.4) ist eine Software zur Modellierung und Ausführung von Geschäftsprozessen, entwickelt von der Firma Lombardi im Auftrag der IBM. Die Software basiert seit 2007 komplett auf GWT, wobei frühere Versionen Flash- [Ado12] und JavaScript-Bibliotheken nutzten. Vor allem im Hinblick auf die Modellierung eines Graphen und die allgemeine Interaktion mit der Benutzungsoberfläche entspricht die Anwendung den Anforderungen an die PACT-GUI (4.1.2). Die erfolgreiche Entwicklung von Blueworks beweist die Realisierbarkeit des Projekts auf Basis von GWT und eliminiert somit das Risiko einer Fehlentscheidung bei der Framework-Auswahl (2.4.1 - Komponentenbasiertes Modell) fast völlig.

4.3.3 Google Web Toolkit

Das *Google Web Toolkit* (kurz GWT) [Goo11c] ist ein Java-Framework zur Erstellung von Rich-Client-Webanwendungen (2.3.1). Es wurde 2006 von Google Inc. als kostenloses, quelloffenes Entwicklerwerkzeug veröffentlicht und seitdem kontinuierlich bis zur aktuellen Version 2.4 weiterentwickelt.

GWT-Compiler: Java-zu-JavaScript-Übersetzung

Von Beginn an erhielt GWT große Aufmerksamkeit, weil es nicht nur ein weiteres Java-

[13] vormals „IBM Blueprint"

oder JavaScript-Framework ist, sondern eine völlig neue Herangehensweise an die Webentwicklung darstellt. GWT ist, wie in Abbildung 4.8 veranschaulicht, ein integrierte Ansatz, der es ermöglicht, nur mit Java aus einer zentralen Entwicklungsumgebung heraus sowohl das Front-End als auch das Back-End zu entwickeln. Die Nutzung der Browser-Technologien HTML, JavaScript und CSS (2.3.2) ist nach wie vor möglich, jedoch zur Entwicklung einer Rich-Client-Webanwendung (2.3.1) nicht mehr erforderlich.

Realisiert wird dies durch einen Java-zu-JavaScript-Compiler, der die zentrale Komponente von GWT darstellt. Er übersetzt den für den Browser entwickelten Java-Code in browserspezifisch optimierten JavaScript-Code. Beim Aufruf einer GWT-Anwendung wird dieser JavaScript-Code vom Webserver heruntergeladen und im Webbrowser ausgeführt. Für die Webanwendung nötiges HTML und CSS wird per JavaScript erstellt.

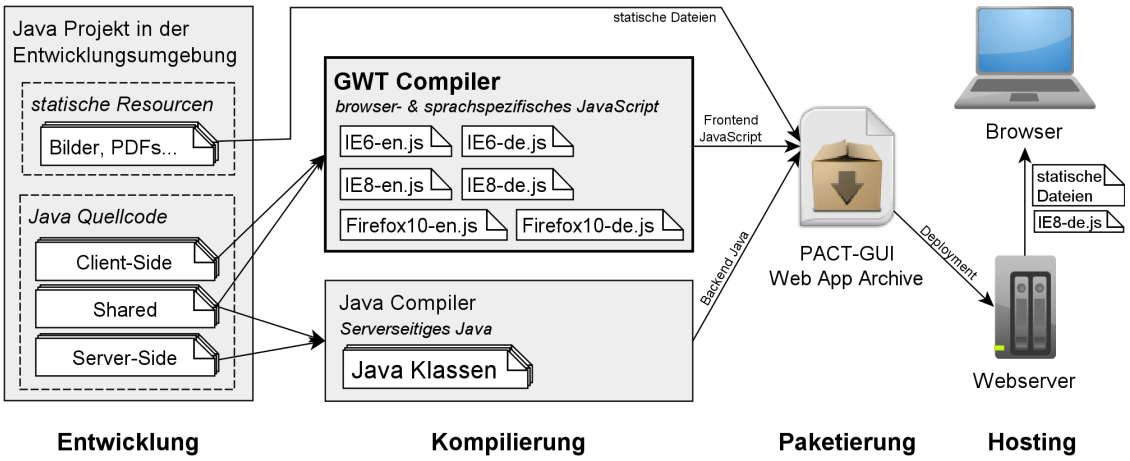


Abbildung 4.8: Erstellungsprozess einer GWT-Anwendung

Zusatzkomponenten

Über die Grundfunktionalität der Java-zu-JavaScript-Kompilierung hinaus bietet GWT viele weitere Komponenten und Konzepte, um die Entwicklung zu vereinfachen. Unter anderem sind dies: ein Remote-Procedure-Call-Mechanismus (4.4.2) für die Ajax-basierte Kommunikation (2.3.2) zwischen Browser und Webserver; Mehrsprachigkeit des Frond-Ends; einen Event-Bus (2.3.3 - Publish/Subscribe); ein Model-View-Presenter-Konzept (2.3.3); eine Sammlung von Steuerungselementen wie Menüs, Tabs und Bäume [Goo11d].

Die Liste vorhandener Steuerelemente[3] (english: Widgets) ist jedoch weder vollständig noch wurde besonderer Wert auf ein modernes einheitliches Design gelegt. Um diesen Umstand zu beheben, können entweder selbst weitere Widgets implementiert werden oder nach dem komponentenbasierten Modell (2.4.1) eine zusätzliche Widget-Bibliothek wie SmartGWT[14] [Iso11], Gwt-Mosaic [Geo11] oder GXT[15] [San11a] eingebunden werden. Letztere ist die komplett Java-basierte GWT-Adaption der bereits für den GUI-Prototypen verwendeten Bibliothek *Ext.js* (4.2.1). Sie bietet eine große Menge qualitativ hochwertiger

[14] vormals Gwt-Ext `http://gwt-ext.com`
[15] vormals Ext-GWT

und visuell ansprechender Widgets und ist, sofern die Anwendung keinen kommerziellen Zweck verfolgt, frei erhältlich. GXT erscheint im Vergleich zu den anderen Alternativen technisch am ausgereiftesten[16] und wird daher für die PACT-GUI eingesetzt. Anhang A.5 zeigt den Vergleich der nativen GWT-Widgets [Goo11d] und der moderneren GXT-Widgets [San11a].

Beschränkungen
An dieser Stelle darf nicht unerwähnt bleiben, dass GWT auch Nachteile birgt und sich nicht für jeden Anwendungsfall eignet [Ker10, S. 10f]:

- **Suchmaschinenoptimierung:** Zum einen erstellt eine GWT-Anwendung sämtliche HTML-Elemente clientseitig durch die Ausführung des ausgelieferten JavaScripts. Zum anderen ändert sich die Browser-URL[3] während der Navigation durch die GWT-Anwendung nicht (2.3.2 - Verwendung von Ajax zur Vermeidung des Paradigmas einzelner Seiten). Somit sollte GWT nicht verwendet werden, wenn die Suchmaschinenfreundlichkeit der Anwendung wichtig ist.

- **Unnötige Abstraktion:** Vor allem sehr gute Front-End- und speziell JavaScript-Entwickler argumentieren gegen die Verwendung von GWT. Durch die Übersetzung von Java zu JavaScript, aber auch die automatische Erzeugung von HTML würde nicht immer optimaler Code erzeugt. Die zusätzliche Abstraktionsschicht verbirgt einige Stärken von JavaScript, erzeugt mehr Code und senkt in bestimmten Fällen die Ausführungsgeschwindigkeit [Mar11, Shi09].

  Diese Argumente sind auf einzelne Beispiele bezogen korrekt, ähneln jedoch denen bei früheren Sprüngen auf die jeweils höhere Programmiersprache. Im Vergleich zu JavaScript verspricht GWT, zumindest über alle Browser und den gesamten Softwarelebenszyklus hinweg betrachtet, eine schnellere Entwicklung von kompatibler, performanter, stabiler und gut wartbarer Software [Ker10, S. 11f].

- **Unnötige Kompilierung:** Ein Nachteil von Java ist die Notwendigkeit ein Projekt nach Codeänderungen neu zu kompilieren. Dies wird bei GWT zumindest für den clientseiten Java-Code durch den GWT-Development-Mode [Ker10, S. 27] behoben. Dieser führt Java-Code mit Hilfe eines Browser-Plugins „On-The-Fly" als Bytecode im Webbrowser aus. Änderungen im GWT-Projekt werden ohne Neustart des Servers direkt im Webbrowser sichtbar.

- **Sicherheit:** GWT-Anwendungen sind nicht weniger sicher als andere JavaScript-Anwendungen. Für besonders sensible Applikationen sollte jedoch speziell geprüft werden, ob GWT die nötigen Anforderungen erfüllt oder ob möglicherweise nur eine stärker Back-End-fokussierte Lösung das benötigte Maß an Sicherheit bieten kann.

[16] komplett Java-basiert; GWT 3 kompatibel; gut dokumentiert; professionelles Entwicklerteam sowie die meisten Nutzer

4.3.4 Weitere Werkzeuge

Bei der Entwicklung einer Java-basierten GWT-Software bietet sich die Verwendung verschiedener Werkzeuge an. Dieses Unterkapitel listet für die Entwicklung der PACT-GUI verwendete Werkzeuge auf und beschreibt kurz ihre Funktion.

Eclipse IDE [17] [The12g] ist eine quelloffene, integrierte Entwicklungsumgebung zur Softwareerstellung. Im Jahr 2001 erschienen, war Eclipse anfangs nur für Java-Entwicklung geeignet, unterstützt aber inzwischen viele weitere Sprachen wie PHP, JavaScript oder C++. Eclipse besitzt bereits in der Grundversion umfangreiche Funktionen, wie die baumartigen Darstellung aller Projektressourcen, einen Quellcode-Editor mit Syntax-Hervorhebung, Autovervollständigung, statische Quellcode-Validierung und Refactoring-Werkzeuge. Darüber hinaus können durch Erweiterungen, sogenannte Eclipse-Plugins, zusätzliche Funktionen in die Entwicklungsumgebung integriert werden. Für alle weiteren in diesem Unterkapitel vorgestellten Werkzeuge existieren kostenlose Plugins, um diese ohne Medienbrüche aus einer zentralen Entwicklungssoftware heraus nutzen zu können.

Git [Cha12] ist eine kostenlose, quelloffene Software zur verteilten Versionsverwaltung von Dateien. Sie wurde 2005 von Linus Torvalds entwickelt zur Quellcodeverwaltung des Linux-Betriebssystemkernels. Im Vergleich zu anderen älteren Versionsverwaltungssystemen zeichnet sich Git besonders durch die verteilte, sehr effiziente, nicht-lineare Verwaltung aus. Jeder Git-Benutzer verwaltet eine lokale Kopie des gesamten Projekt-Repositories und kann jederzeit neue Entwicklungszweige erstellen, diese zusammenführen und zwischen verschiedenen Versionen navigieren.

Jenkins [Jen12] (vormals bekannt als Hudson) ist ein webbasiertes Softwaresystem zur wiederholten Ausführung und Überwachung von Aufgaben. Jenkins ist quelloffen, kostenlos und mit einer Vielzahl von Plugins erweiterbar. In der agilen Softwareentwicklung wird die Jenkins-Plattform als *Continuous Integration System* genutzt. Ein solches System führt in regelmäßigen Abständen auf Basis des derzeitigen Projektstands automatisiert Softwaretests durch. Auftretende Fehler werden überwacht und protokolliert, um das Entwicklerteam zeitnah auf Probleme der letzten Änderungen hinzuweisen.

Maven [The12b] ist ein *Build Management Tool*, das Entwicklern bei der Verwaltung von Java-Projekten hilft. Es unterstützt den gesamten Softwarelebenszyklus von der Erstellung über das Kompilieren, Testen und Paketieren eines Projekts bis zur Verteilung an den Webserver. Die Konfiguration von Maven in einem Projekt geschieht über ein *Project Object Model*, definiert durch eine pom.xml-Datei. In dieser sind alle projektspezifischen Einstellungen zusammengefasst, wobei in Maven nur konfiguriert werden muss, was von den standardisierten Konventionen abweicht[18]. Besonders nützlich bei der Verwendung von Maven ist die automatische Abhängigkeitsverwaltung, welche externe Bibliotheken und ihre Abhängigkeiten selbstständig aus einem zentralen Verzeichnis im Internet lädt.

[17] IDE = Integrated Development Environment (zu deutsch: Integrierte Entwicklungsumgebung)
[18] Dieses Prinzip ist bekannt als *Convention over configuration*

JavaDoc [Ora12b] ist ein Software-Dokumentationswerkzeug, um aus Java-Quelltexten automatisch eine HTML-Dokumentation zu erzeugen. Es wurde von Sun Microsystems entwickelt und ist ein Bestandteil des Java-Development-Kits [RAS]. In der generierten Dokumentation sind alle Klassen, Interfaces, Methoden, Konstruktoren und Datenelemente aufgeführt. Die Dokumentation kann mittels speziell gekennzeichneter Quellcode-Kommentare des Entwicklers ergänzt werden. Anhang A.2 zeigt beispielhaft eine mit JavaDoc generierte Dokumentation einer Java-Klasse der PACT-GUI.

4.4 Implementierung

Im Rahmen dieser Masterarbeit wurde die PACT-GUI prototypisch implementiert, um die Umsetzbarkeit eines Werkzeugs zur visuellen Modellierung von Datenflussgraphen (3.3) und die Eignung der gewählten Technologien (4.3) zu überprüfen. Die Entwicklung erfolgte entsprechend dem evolutionären Vorgehensmodell (4.1.1) auf Basis der beschriebenen Anforderungen (4.1.2) und der konzeptionellen Vorüberlegungen (4.2). Dieses Unterkapitel stellt die Anwendungsarchitektur überblicksartig vor und geht auf ausgewählte technische Aspekte des Front- (4.4.1) und Back-Ends (4.4.2) detailliert ein.

Um große Softwareprojekte langfristig warten, weiterentwickeln und vor allem verstehen zu können, ist eine stabile und anpassungsfähige Software-Architektur von größter Bedeutung [GU10, S. 178ff]. Die Software-Architektur ist gegeben durch funktionstragende Komponenten und die strukturbestimmenden Beziehungen des Systems[3] [Bau08, S. 26]. Sie sollte zu Beginn der Implementierung als erstes realisiert werden, da sie wesentlichen Einfluss auf alle weiteren Entwicklungsschritte hat.

Abbildung 4.9 zeigt eine vereinfachte Übersicht der Software-Architektur der PACT-GUI. Es handelt sich um eine 2-Tier-Rich-Client-Webanwendung (2.3.1), geteilt in Front-End (2.3.1 - Webbrowser) und Back-End (2.3.1 - Webserver), ohne serverseitige Persistenzschicht (Datenbank). Die gesamte Oberfläche der PACT-GUI wird im Front-End generiert. Die Anwendungslogik ist nach dem Konzept der kooperativen Verarbeitung (2.3.1) auf Front- und Back-End verteilt.

Die Abbildung zeigt zusätzlich den PACT-GUI-Nutzer, der nicht Teil des technischen Systems ist, aber auf verschiedenen Wegen mit der PACT-GUI interagiert. Seine erste Anfrage beim Aufruf der Anwendung im Webbrowser wird vom GWT-Framework an die obligatorische GWT-EntryPoint-Klasse weiter geleitet, um den GWT-ApplicationController und damit die gesamte GWT-Anwendung zu initialisieren. Anschließend steuert der Nutzer die Anwendung durch Bedienung der GUI und kann Dateien mittels direkter HTTP-Anfragen (2.3.1) zum Webserver hoch und von ihm herunter laden.

4.4.1 Front-End

Im Webbrowser wird die gesamte Anwendungsoberfläche erzeugt und der größte Teil der Applikationslogik ausgeführt. Die PACT-GUI verwendet das MVP-Muster, wie bei der

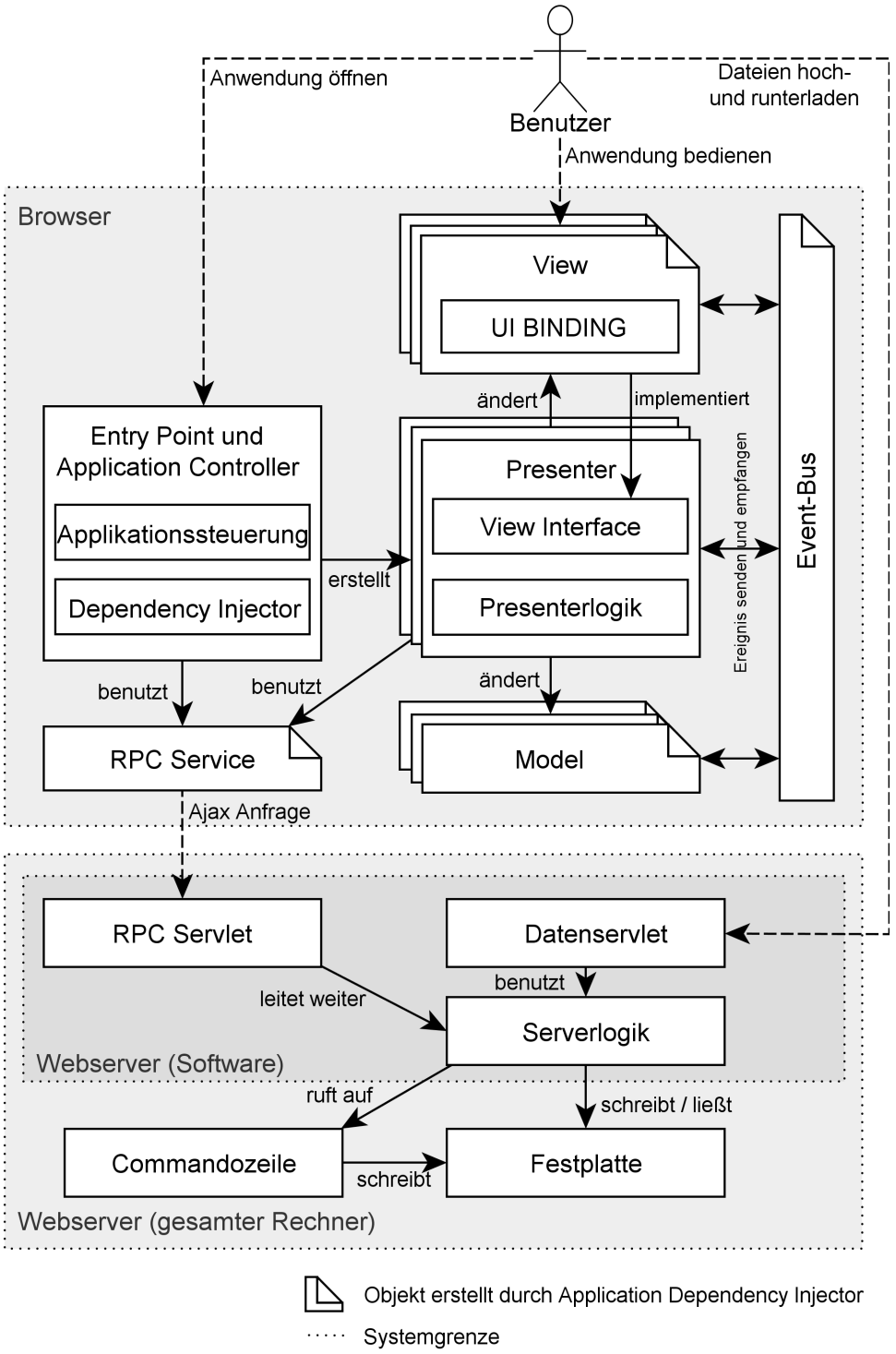


Abbildung 4.9: PACT-GUI-Architektur

GWT-Entwicklung empfohlen [Rya09, Cha09, Ram10], sowohl global für das gesamte Front-End als auch einzeln für jeden Teilbereich der Benutzungsoberfläche. Die Presenter enthalten die Anwendungslogik. Steuerelemente[3] werden durch die Views erstellt. PACT-Programme befinden sich im Model. Zusätzlich definiert jeder Presenter ein View-Interface, um die Anbindung verschiedener Views beispielsweise für Desktop oder mobile Endgeräte zu ermöglichen. Der ApplicationController steuert den Wechsel zwischen Presentern und initialisiert den Dependency Injector, welcher, wie in 4.5 detailliert beschrieben, für die weitere Objekterstellung und Ressourcenverwaltung verantwortlich ist. Die Views werden zum größten Teil als *Passive Views* [Fow06] deklarativ per XML[3] mit nur einem Minimum Java-Code erstellt. Durch dieses *Declarative UI Binding* [Ker10, S. 69f] wird die Erstellung einer View weiter vereinfacht und Softwaretests der View (4.5) unnötig. Das Model (Abbilung 4.12) ist der einzige Anwendungsbestandteil, der sowohl vom Frond-End zur Modellierung als auch vom Back-End zum Laden, Speichern und Generieren von PACT-Programmen verwendet wird. Um die Kommunikation von Model, View und Presenter zu ermöglichen, stellt GWT mit dem Event-Bus [Ram10] eine Nachrichtenverteilungskomponente nach dem Publish/Subscribe-Muster (2.3.3) zur Verfügung. Der RPC-Service wird genutzt zur Kommunikation mit dem Server und wird wie das Model in Unterkapitel 4.4.2 vorgestellt.

Der **Graph-Editor** ist das wichtigste und komplizierteste Steuerelement der PACT-GUI. Er dient zur Modellierung des PACT-Programm-Graphen und stellt während der Nutzung der Anwendung die zentrale Repräsentation des zu entwickelnden PACT-Programms dar. Anhang A.6 zeigt das entsprechend 4.2.1 implementierte Widget. Alle erstellten Objekte können auf der Arbeitsfläche mittels der Zusatzbibliothek GWT-DND[19] [Sau11] frei bewegt werden. Einzelne PACTs sind um diverse Elementen und Funktionen erweiterte GXT-Komponenten. Datenkanäle bestehen aus einem Dateneingangs- und einem Datenausgangselement, zwischen denen automatisch eine Linie gezeichnet wird.

Zum Zeichnen der Linien wird die Browser-Technologie *HTML 5 Canvas*[20] genutzt. Canvas wird nativ durch GWT unterstützt und ermöglicht, dynamisch Grafiken durch Setzten von Punkten, Kreisen oder Linien auf einer Zeichenfläche zu erstellen. Bereits gezeichnete Objekte können später nicht mehr bearbeitet werden, weshalb in der PACT-GUI Linien ständig neu gezeichnet werden müssen, wenn ein Datenkanal oder mit ihm verbundene PACTs bewegt werden. Um den Umfang der Neuzeichnung zu minimieren, befindet sich jede Linie auf einer separaten, rechteckigen und transparenten Zeichnungsfläche. Problematisch ist, dass diese Flächen sich möglicherweise überlagern, wodurch die Selektion einer Linie per Mausklick erst durch geometrische Berechnung der Entfernung des Klickpunkts zu allen Strecken in einem zweidimensionalen Koordinatensystem möglich ist.

Die ständige Neuzeichnung und das Problem der sich überlagernden Zeichnungsflächen könnte durch die Verwendung der alternativen Technologie SVG[21] vermieden werden. Mit

[19] DND = „Drag and Drop", zu deutsch „Ziehen und Fallenlassen"
[20] Canvas, zu deutsch „Leinwand""
[21] SVG = Scalable Vector Graphics

SVG lassen sich skalierbare Vektorgrafiken erstellen, die mittels JavaScript transformiert und bewegt werden können. Würden nicht nur die Linien, sondern alle Objekte des Graph-Editor-Widgets per SVG erstellt, wäre es zudem mit geringem Aufwand möglich, den Graph als PDF[3] zu exportieren oder eine Zoom-Funktion zu implementieren. Canvas wird dennoch vorgezogen, da die Umsetzung einer SVG-Lösung deutlich komplexer ist und von GWT nicht nativ, sondern nur durch Zusatzbibliotheken [Spe11, Vec11] unterstützt wird.

Das **Java-Editor-Widget** zur Erstellung der UDF eines PACTs ist in Anhang A.6 zu sehen. Für die Umsetzung wurde wie im GUI-Prototypen (4.2.1) die JavaScript-Bibliothek *CodeMirror* [Hav11] verwendet, da kein Quelltext-Editor mit ähnlichem Funktionsumfang auf GWT-Basis existiert und eine komplette Eigenentwicklung, wie erste Versuche gezeigt haben, zu zeitaufwändig ist. Die Nutzung von JavaScript in GWT-Anwendungen widerspricht eigentlich dem Anliegen von GWT, JavaScript komplett aus Java zu generieren (4.3.3). Sie ist aber nicht zu vermeiden, wenn eine große JavaScript-Codebasis weiter verwendet oder von GWT nicht bereitgestellte Browserfunktionalitäten genutzt werden sollen. Ermöglicht wird der Zugriff auf JavaScript-Funktionen mittels Java durch die GWT-spezifische Technik *JavaScript Native Interface* [Goo11a] (Programmbeispiel 4.10). Das Java-Editor-Widget wurde entsprechend früheren Versuchen in dieser Richtung [Vai10, Guz10] als ein völlig eigenständiges GWT-Modul entwickelt und kann als Java-Archiv kompiliert in anderen Projekten wiederverwendet werden.

```
public static native void alert(String msg) /*-{
  $wnd.alert(msg);
}-*/;
```

Abbildung 4.10: Zugriff auf JavaScript aus Java mittels GWT-JSNI

4.4.2 Back-End

GWT-Anwendungen werden möglichst komplett im Webbrowser (Front-End) ausgeführt. In einigen Fällen kann aber auch bei GWT-Anwendungen nicht auf die Interaktion mit dem Webserver (Back-End) verzichtet werden. Dieser wird beispielsweise benötigt, wenn Codeteile während der Laufzeit dynamisch nachgeladen werden sollen (*Code Splitting* [Goo12c]) oder GWT-inkompatible Anweisungen [Goo12a] auf dem Server ausgeführt werden müssen. Meist wird ein Server jedoch benötigt, um Ressourcen zu nutzen, auf die aus der JavaScript-Umgebung des Webbrowsers kein Zugriff besteht, wie eine Datenbank oder ein öffentlicher Webservice. Denkbar ist aber auch rechenintensive Operationen aus der Browserumgebung zum Server oder einer Serverfarm zu übertragen.

Die PACT-GUI benötigt den Webserver für Dateioperationen[22] und zum Ausführen externer Programme beim Laden, Speichern (4.2.2 - User Story 2) und Generieren (4.2.2 -

[22]Es sei erwähnt, dass mit Techniken wie HTML5 oder Flash [Ado12] auch aus dem Browser in gewissem Umfang Dateioperationen ohne Einbindung des Servers durchgeführt werden können

User Story 3) von PACT-Programmen. Im Folgenden wird die Ajax-basierte Kommunikation zwischen (2.3.2) Front-End und Back-End, die Serialisierung von PACT-Programmen (4.2.2) und die Ausführung externer Programme (4.2.3) beschrieben.

Die **Ajax-basierte Kommunikation** (2.3.2) zwischen Browser und Server zur Übertragung von PACT-Programmen (4.2.3 - Schritt 1), wird in GWT durch einen eigenen Remote-Procedure-Call-Mechanismus (GWT-RPC [Goo12b]) ermöglicht. Dieser erlaubt direkt aus dem clientseitigen Teil der GWT-Anwendung heraus serverseitige Methoden aufzurufen (Abbildung 4.9). Die Details der Datenübertragung, wie die Serialisierung der als Parameter übergebenen Objekte, sind für den Entwickler dabei völlig transparent. Für ihn besteht der einzige Unterschied zu lokalen Methodenaufrufen darin, dass RPCs asynchron[23] sind. Das heißt, der Rückgabewert der Methode kann nicht direkt für die weitere Verarbeitung genutzt werden, sondern nur über den Umweg einer Callback-Funktion. Diese Funktion wird zu einem späteren Zeitpunkt nach Beendigung des RPCs mit der Serverantwort als Übergabeparameter aufgerufen. Um einen GWT-RPC-Dienst zu erstellen, muss ein Dienst-Interface definiert sowie serverseitig als Servlet (4.3.1) implementiert werden. Clientseitig wird ein Dienst-Objekt vom GWT-Framework automatisch instantiiert und kann zur Übertragung von Objekten wie PACT-Programmen genutzt werden. Abbildung 4.11 verdeutlicht den zeitlichen und technischen Ablauf eines RPCs [Ras10].

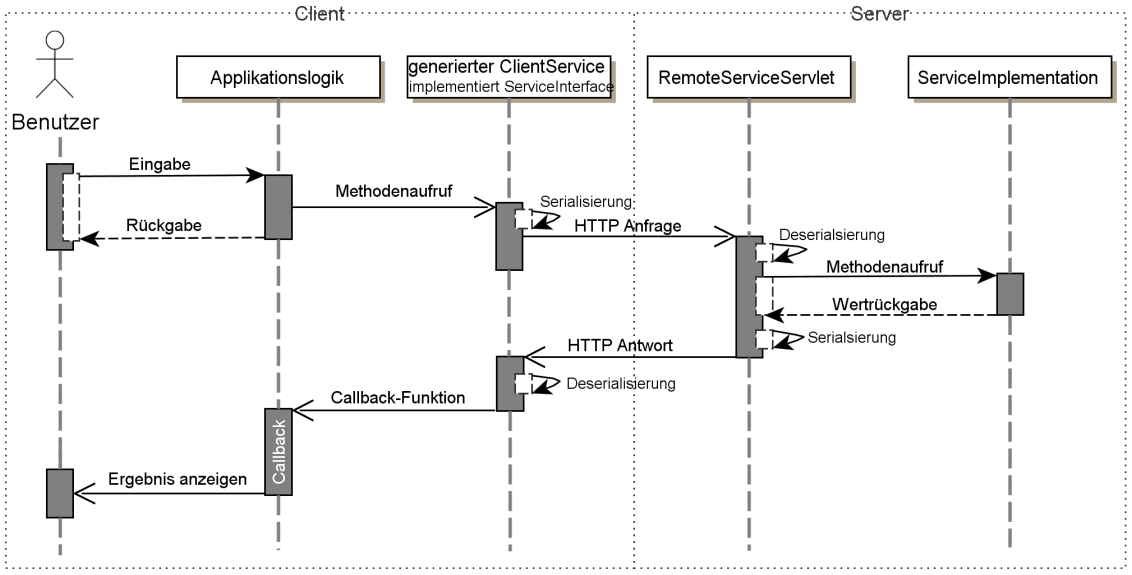


Abbildung 4.11: GWT-RPC-Sequenzdiagram

Die **Serialisierung von PACT-Programmen** (4.2.2 - User Story 2) wird mit der *XML-Encoder/Decoder API*[3] [RAS, S. 262ff] implementiert. Die API ist Teil des Standard Development Kits und seit Java 1.4 der offiziell empfohlene Mechanismus zur Serialisie-

[23] Hier liegt der Hauptunterschied zu Java Remote Method Invocations (RMI), die stets synchron, also blockierend sind [Ker10, S. 77].

rung[24]. Sie ermöglicht Java-Beans-Objekthierarchien automatisch nach XML zu serialisieren. XML ist eine Auszeichnungssprache zur Darstellung von hierarchisch strukturierten Daten in Textform. Die Sprache ist für Menschen lesbar sowie plattform- und programmiersprachenunabhängig. Java-Beans sind Java-Klassen, die speziellen Konventionen entsprechen: Sie müssen einen öffentlichen parameterlosen *Constructor* haben, das Interface *Serializable* implementieren und Zugriffsmethoden (*getter/setter*) für Objektattribute bereitstellen. [RAS, S. 262ff]

Abbilung 4.12 zeigt das UML-Klassendiagramm[3] des Java-Bean-Modells eines PACT-Programms. Anhang A.8 zeigt einen Ausschnitt eines mittels der XMLEncoder/Decoder API serialisierten PACT-Programms.

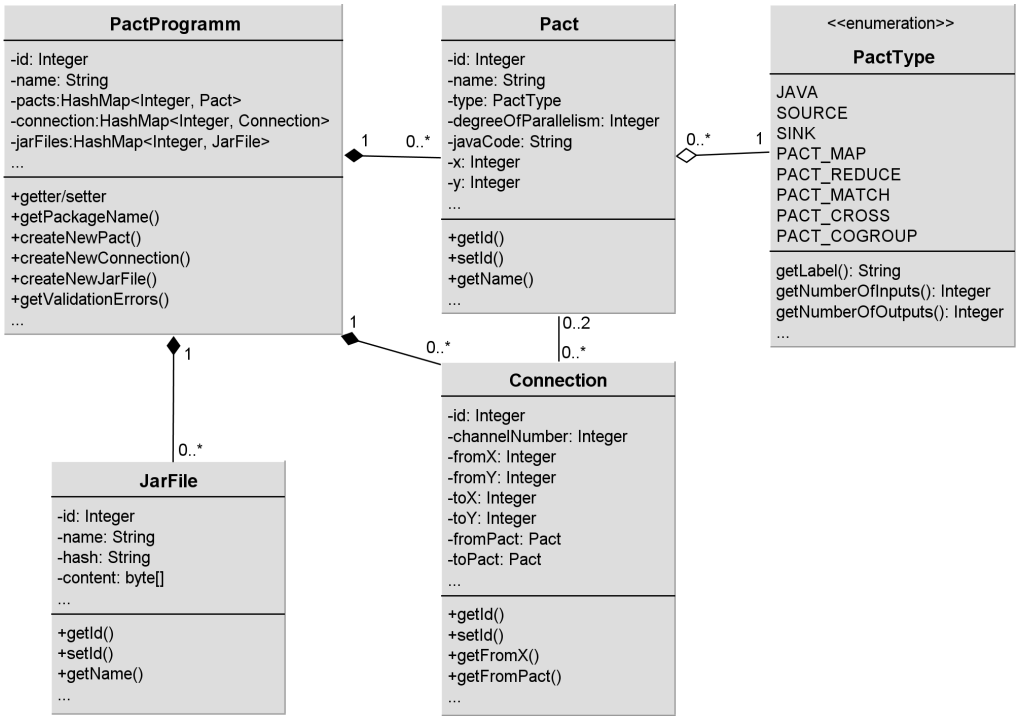


Abbildung 4.12: UML-Klassendiagramm von PACT-Programmen

Die verwendete Serialisierung entspricht nicht komplett dem Standard, sondern wurde mittels *PersistenceDelegates* erweitert. Dies ist nötig, um die Serialisierung von ByteArrays, welche verwendet werden für im PACT-Programm eingebettete externe Programmbibliotheken (4.1.2 - User Story 1d), zu überschreiben. Im Standard wird jedes einzelne Byte in einem eigenen XML-Knoten gespeichert, wodurch ByteArrays während der Serialisierung auf ein Vielfaches ihrer eigentlichen Größe anwachsen. Die Erweiterung erlaubt das gesamte ByteArrays in einem Knoten zu speichern.

[24]technische Alternativen wie die Java Serialization API, JSON, JAXB oder XML Parser wie SAX oder DOM werden daher nicht betrachtet.

Der **Aufruf der externen Kommandozeilen-Programme** *javac* und *jar* zur Generierung von PACT-Programmen (4.2.3) erfolgt auf dem Server mittels der Java-Runtime-Schnittstelle. Diese Schnittstelle ermöglicht aus der Java-Anwendung heraus mit der jeweiligen Systemumgebung zu interagieren und beispielsweise Programme zu starten. Der Ausgabestrom der Programmaufrufe wird in einer Protokolldatei abgespeichert und nach Beendigung des Aufrufes per GWT-RPC (4.4.2) zurück an den Webbrowser übertragen. Der Endnutzer kann das Protokoll und alle erstellten Dateien der Generierung im Erfolgs- und Fehlerfall einsehen (Anhang A.7). Da der Graph des PACT-Programms und die Eigenschaften der PACTs bereits während der Erstellung validiert wurden (4.2.1, 4.2.3), hilft das Protokoll vor allem fehlerhaften Java-Code innerhalb der UDFs zu korrigieren.

4.5 Test

Softwaretests und im speziellen voll automatisierte Softwaretests sind ein elementarer Bestandteil der Java- und GWT-Entwicklung. Sie werden durchgeführt, um im Vergleich zur Spezifikation fehlerhaftes Verhalten der Software zu finden. Sie helfen bei der Entwicklung, sind aber keine Garantie für eine fehlerfreie Software. Tests weisen nur die Anwesenheit von Fehlern nach, nicht deren Abwesenheit. Im Gegensatz zu statischen Prüfmethoden, die allein den Quelltext untersuchen, werden bei der dynamischen Prüfung durch Softwaretests das Programm oder Teile des Programms immer auch ausgeführt. Somit kommt Tests auch eine Dokumentationsfunktion zu, da sie ein funktionierendes Beispiel der Nutzung des Codes sind. [Bal09, S. 71]

In einem optimalen Szenario wird zuerst ein einzelner Test (Testfall) für eine bestimmte noch nicht implementierte Funktionalität (Testobjekt) geschrieben. Dieser schlägt dann so lange fehl, bis das Testobjekt entsprechend der Spezifikation fertig entwickelt ist. Einfacher ist es jedoch meist den Testfall parallel oder erst im Anschluss zum Testobjekt zu entwickeln. Der Testfall wird im weiteren Entwicklungsverlauf wiederholt automatisch ausgeführt (4.3.4 - Continuous Integration System), wodurch stets die korrekte Funktion des Testobjekts gewährleistet ist und Inkompatibilitäten mit später entwickelten Softwarekomponenten festgestellt werden. Es werden drei Arten von automatischen Tests unterschieden [Ker10, S. 229f]:

- *Unit-Tests* prüfen isoliert eine Einheit, beispielsweise eine Komponente, eine Klasse oder eine Methode, unabhängig von externen Einflüssen.
- *Integration-Tests* prüfen das Zusammenwirken verschiedener Systemkomponenten.
- *Acceptance-Tests* prüfen das gesamte System entsprechend definierter von Anwendungsfälle.

4.5.1 Unit-Tests

Unit-Tests prüfen ein Testobjekt auf korrekte Funktionsweise, indem sie das Resultat einer Operation (zum Beispiel den Rückgabewert einer Funktion) mit dem erwarteten Wert

vergleichen. Um aussagefähige Unit-Tests zu erstellen, muss das Testobjekt soweit isoliert werden, dass es unabhängig von externen Einflüssen geprüft werden kann. Dies ist schwierig, da Objekte normalerweise selbst benötigte Ressourcen verwalten und andere Objekte erstellen. Isoliert ist ein Objekt erst, wenn es keinerlei Kenntnis seiner Umgebung beinhaltet und keine anderen Objekte erzeugt. Somit ist es nötig, schon bei der Softwareentwicklung Schritte zu unternehmen, um die spätere Testbarkeit vorzubereiten. Allein das Vorhaben, Testfälle zu erstellen, führt demnach oft bereits zu besserem Softwaredesign und höherer Softwarequalität.

Gelöst wird das Problem der Objektisolation durch das Entwurfsmuster *Dependency Injection*, welches auf dem Prinzip der *Inversion of Control*[25] basiert. Dabei werden alle innerhalb eines Objektes benötigten externen Ressourcen von außen hineingegeben und nicht durch das Objekt aktiv erstellt oder angefordert. Dies geschieht mittels eines Verwaltungsobjektes, dem *Dependency Injector*, welcher dem Objektkonstruktor als Parameter übergeben wird. Beispielsweise muss das Dependency-Injector-Objekt für einen Presenter zum einen die View und das Model erzeugen können, zum anderen Referenzen zu Diensten wie dem Event-Bus und dem GWT-RPC-Mechanismus (4.4.2) bereitstellen. Um die Konfiguration des Dependency Injectors zu vereinfachen und die Objekterstellung und -verwaltung zu automatisieren, wird für die PACT-GUI das GWT-spezifische Framework GIN [Goo11b] (GWT Injection) verwendet.

Bei der Erstellung von Testfällen wird das Testobjekt mit einem textspezifischen Dependency Injector instantiiert, der nur Stellvertreterobjekte (Mocks[26]) bereitstellt. Mock-Objekte implementieren die Schnittstellen der echten Umgebungsobjekte, stellen also die gleichen Methoden und Rückgabewerte zur Verfügung, ohne dabei beispielsweise wirklich eine Anfrage zum Server zu schicken oder eine View zu erstellen. Für die automatische Erstellung von Mocks wird die Zusatzbibliothek Mockito [Fab11] verwendet.

Für die PACT-GUI wurden 10 Unit-Tests mit Hilfe der Java-spezifischen Testbibliothek *JUnit* [JUn12] erstellt. Zwei der Testfälle prüfen das Model, ein Testfall prüft verschiedene Server-Funktionen und sieben Testfälle prüfen die verschiedenen Presenter (Main, Login, CompilerResults, GraphDesigner, JavaEditor, Programlist, TabContainer).

4.5.2 Integration- und Acceptance-Tests

Für Integration und Acceptance-Tests, die das Testen der View-Komponenten beinhalten, ist bei GWT-Anwendungen ein Browser nötig [Ker10, S. 230]. Dieser kann mittels spezieller Erweiterungen von JUnit-Tests, den GWTTestCases [Cha09] virtuell simuliert werden. Laut der Entwickler von GWT sollten GwtTestCases dennoch soweit wie möglich vermieden werden [Cha09], da ihre Entwicklung komplex ist und ihre Ausführung mindestens 30 bis 60 Sekunden in Anspruch nimmt [Ker10, S. 253]. Wenn Tests jedoch schwer zu schreiben sind und langsam laufen, werden sie wahrscheinlich nie geschrieben und nie durchgeführt

[25] zu deutsch: „Steuerungsumkehr"

[26] vom Englischen „to mock" - „etwas vortäuschen"

werden [Ker10, S. 57]. Stattdessen wird empfohlen, konsequent das MVP-Muster (2.3.3) umzusetzen. Dadurch wird die View auf wenige Zugriffsfunktionen (Getter) von Viewinhalten sowie Event-Weiterleitungen reduziert und benötigt keine expliziten Tests (4.4.1). Für die PACT-GUI wurden daher keine Testfälle der View erstellt.

Kapitel 5

Evaluation und Ausblick

Nach Abschluss der Implementierung der PACT-GUI werden in diesem Kapitel der Grad der Anforderungserfüllung (5.1), die gewählte Vorgehensweise (5.2) sowie die getroffenen Entwurfsentscheidungen (5.3) evaluiert. Abschließend wird ein Ausblick über potentielle Weiterentwicklungsmöglichkeiten (5.4) gegeben.

5.1 Erfüllung der Anforderungen

Das Projektziel, ein Browser-basiertes Werkzeug zur visuellen Erstellung und Bearbeitung von PACT-Programmen prototypisch zu entwickeln (3, 4.1), wurde erreicht. Betriebssystem- und Webbrowser-unabhängig (4.1.2 - Rahmenbedingung 7, 8 ,9) ist es nun jeder Person mit Vorwissen bezüglich des PACT-Programmiermodells (4.1.2 - Rahmenbedingung 2) mit geringem Zeitaufwand möglich, PACT-Programme zu erstellen. Die Bedienung der Software ist unkompliziert und leicht zu erlernen (4.1.2 - Ziel 3).

PACT-Programme können gespeichert und geladen werden (4.1.2 - User Story 2), was eine spätere Weiterbearbeitung und die Übertragung an andere Personen erlaubt. Aus der PACT-GUI heraus können PACT-Programme zu einem Java-Archiv generiert werden, um sie anschließend mittels des PACT-Compilers und Nephele auszuführen (4.1.2 - User Story 3).

Alle erstellten PACT-Programme entsprechen dem PACT-Programmiermodell (4.1.2 - User Story 1). Jedoch können in der jetzigen Version der PACT-GUI nicht alle theoretisch möglichen PACT-Programme erstellt werden. Auf Grund von Änderungen des Stratosphere-Systems (siehe nächstes Unterkapitel) wurde beispielsweise die Zuordnung von Output Contracts zu PACTs vorläufig entfernt, da die weitere Nutzung dieses Konzepts unklar ist und es mehrfach zu Änderungen in diesem Bereich kam.

5.2 Bewertung des gewählten Vorgehens

Die Entwicklung der PACT-GUI nach dem evolutionären Vorgehen mit Anleihen bei den agilen Methoden (4.1.1) hat sich über alle Phasen der Entwicklung als zweckmäßig herausgestellt. Rückblickend ist festzustellen, dass die Anforderungen an die Software sowie

die Grenzen und Möglichkeiten der verwendeten Technologien zu Projektanfang nur unzureichend bekannt waren. Somit wäre eine strikte Anforderungsdefinition nach klassischen Vorgehensweisen zwar möglich, aber wenig erfolgversprechend gewesen. Erst durch Nutzung des GUI-Prototypen (4.2.1) und der ersten Versionen der lauffähigen Software hat sich bei den Projektbeteiligten eine genaue Vorstellung des zu entwickelnden Systems gebildet.

Durch das agile Vorgehen konnte zusätzlich zu den Wünschen der Nutzer auch auf Änderungen der Systemumgebung flexibel reagiert werden. So wurde während der Implementierung der PACT-GUI das Stratosphere-System von Version 0.1 auf 0.2 weiterentwickelt. Die neue Version hat weitreichende Änderungen der PACT-Programme im Hinblick auf zu erweiternde Schnittstellen, vorhandene Datentypen und die Verwendung von Output Contracts zur Folge. Dahingegen wurden für die PACT-GUI eigentlich bereits geplante Funktionen, wie die automatische Vereinigung (Union) von mehreren Eingabedatenquellen, vorerst noch nicht in Stratosphere eingeführt.

Als nachteilig an dem agilen Vorgehen hat sich auch im Rahmen dieser Masterarbeit gezeigt, dass häufiges Reagieren auf Änderungen das Risiko erhöht, den vorab geplanten zeitlichen Rahmen zu überschreiten. Zudem ist das Erreichen des Projektzieles, also der Abschluss der Softwareentwicklung, bei nicht exakt definierten Anforderungen unklar.

Technisch war vor allem die Verwendung von Prototypen (2.4.1) von großem Nutzen. Bei der Evaluation von Technologien (4.3.2) haben sie geholfen, Entscheidungen schneller und fundierter zu treffen; bei der Entwicklung der graphischen Oberfläche (4.2.1) haben sie die Kommunikation vereinfacht und schneller zu einem Resultat geführt, als dies mit einer vollwertigen Implementierung möglich gewesen wäre.

Außerdem war es erst durch die Verwendung von Halbfabrikaten, wie sie im komponentenbasierten Ansatz (2.4.1) beschrieben sind, möglich, eine so umfangreiche[1] Software wie die PACT-GUI innerhalb weniger Monate allein zu entwickeln. Ausführlicher geht das folgende Unterkapitel auf Vor- und Nachteile der verwendeten externen Softwarekomponenten ein.

5.3 Evaluierung der verwendeten Technologien

Wie die Umsetzung des Prototyps mit Ext.js (4.2.1) bereits angedeutet hat, ist eine Webanwendung vom Umfang der PACT-GUI[1] allein mit JavaScript nur schwer realisierbar. Dies liegt vor allem an der fehlenden statischen Typisierung, unzureichender Objektorientierung und nur geringer Werkzeugunterstützung bei der JavaScript-Entwicklung.

Durch den Einsatz von GWT konnte die PACT-GUI voll objektorientiert und ausschließlich mit Java entwickelt werden. Es war möglich automatisierte Softwaretests durchzuführen (4.5), auf die umfangreiche Java-Werkzeugunterstützung (4.3.4) zurückzugreifen

[1] Die fertige PACT-GUI-Webanwendung besteht aus gut 8000 Zeilen Programmcode (ohne Leerzeilen und Kommentare) in 120 Klassen und Interfaces mit mehr als 600 Methoden. Kompiliert ist die Anwendung knapp 50 MB groß, auf Grund der zahlreichen browserspezifischen Versionen. Beim Aufruf der Anwendung werden Dateien mit einer Gesamtgröße von mehr als 1MB an den Client übertragen – rund 80% der Daten sind gepackter JavaScript-Code.

und bekannte Konzepte der GUI-Entwicklung wie das MVP- oder das Publish/Subscribe-Muster (4.4.1) zu nutzen. Die Übersetzung von Java nach JavaScript (4.3.3) funktionierte im Rahmen dieses Projekts stets fehlerfrei. Die erzeugte Applikation ist voll cross-browserkompatibel und performanter, als es direkt programmiertes JavaScript üblicherweise ist [McC10, Gal09].

Zusätzliche Bibliotheken wie GWT-DND (4.4.1), GIN (4.5) und GXT (4.3.3) ließen sich unkompliziert und nahtlos integrieren. Die eingebetteten Technologien Event-Bus und GWT-RPC (4.4.1) haben die Entwicklung deutlich vereinfacht. Das MVP-Muster (4.4.1) war sowohl bei der Entwicklung als auch beim Testen von großem Nutzen. Es hat sich jedoch auch gezeigt, dass der Quellcode durch die weiteren nötigen Abstraktionsschichten, Schnittstellen und Interfaces teilweise deutlich komplexer und umfangreicher wird. Aus diesem Grund wurde absichtlich an einigen Stellen die strikte Trennung zwischen Model, View und Presenter aufgeweicht. Darüber hinaus erforderte GWT teilweise die Erstellung von umfangreichem Boilerplate-Code (4.3.2), was eigentlich durch ein Framework vermieden werden sollte. Insgesamt hat sich das GWT-Framework aber als gut geeignet zur Bewältigung des vorliegenden Anwendungsfalles erwiesen, da es die Produktivität der Entwicklung und die Qualität der Software deutlich erhört hat.

Dennoch sollte erwähnt werden, dass die Einarbeitung in GWT trotz der umfangreichen Dokumentation, sehr zeitaufwendig ist. Dies liegt zum einen am Umfang des Frameworks und der Menge eingebetteter Konzepte. Zum anderen ist die Java-Entwicklung zur Ausführung im Browser ungewohnt und erfordert eine gewisse Eingewöhnungszeit.

5.4 Weiterentwicklungsmöglichkeiten

„Software veraltet in dem Maße, wie sie mit der Wirklichkeit nicht Schritt hält" [Sne83]. Aus diesem Grund sollte Software einer ständigen Verbesserung und Weiterentwicklung unterliegen, indem sie auch nach der eigentlichen Fertigstellung an veränderte Umweltbedingungen und neue Anforderungen angepasst wird. Nachfolgend werden einige Anregungen zur zukünftigen Weiterentwicklung der PACT-GUI aufgeführt.

Fehlerkorrektur und organische Verbesserungen
Nicht triviale Software enthält mit sehr hoher Wahrscheinlichkeit auch nach intensivem manuellen und automatischen Testen Fehler [Hor11, S. 695]. Von daher ist es wichtig im produktiven Einsatz erkannte Fehler zeitnah zu beheben. Des weiteren sollten Verbesserungsvorschläge der zukünftigen Nutzer der PACT-GUI beispielsweise per Interview oder Fragebogen gesammelt werden. Die Analyse dieser Vorschläge und gegebenenfalls ihre Umsetzung können helfen, die Nutzung der PACT-GUI weiter zu vereinfachen und an die Bedürfnisse der Endnutzer anzupassen.

Funktionserweiterungen
Durch ihre flexible Systemarchitektur (4.1.2 - Ziel 4) ist es möglich die PACT-GUI um

zusätzliche Funktionen zu erweitern. Nachfolgend werden einige Möglichkeiten der Weiterentwicklung aufgeführt.

In einem ersten Schritt könnten die Funktionen der webbasierten Oberfläche der *Nephele and PACTs Query Interface* (3.1) in die PACT-GUI integriert werden. Innerhalb einer Anwendung wäre es dann möglich, PACT-Programme zu erstellen, sie zu Ausführungsgraphen aufzuspannen und direkt auszuführen (2.2.2, 2.2.3). Auch andere auf dem PACT-Programmiermodell basierende Modellierungssprachen wie Sopremo [HN12] könnten zukünftig in die Anwendung integrieren werden. Dabei sollten die verschiedenen Abstraktionsebenen jedoch immer erkennbar getrennt sein.

Eine weitere sinnvolle Vereinfachung wäre die Möglichkeit für den PACT-Compiler generierte PACT-Programme auch wieder mit der PACT-GUI öffnen zu können. Dafür müsste das PACT-GUI-Modell des Programms in das genierte Java-Archiv eingebunden werden. Auf die separate Speicherung mittels XML kann aber weiterhin nicht verzichtet werden, um auch unvollständige oder nicht valide PACT-Programme speichern zu können.

Ein nicht gespeichertes PACT-Programm geht verloren, sobald die Webanwendung neu geladen wird. Um den ungewollten Verlust des PACT-Programms beispielsweise bei einem Browserabsturz zu verhindern, wäre ein automatisierte Zwischenspeicherung im Front-End (unter anderem möglich mittels HTML 5 WebStorage [Hic11]) oder innerhalb der Back-End-Session empfehlenswert.

Um die Einarbeitung in das PACT-Programmiermodell und die Nutzung der PACT-GUI zu erleichtern, wäre eine umfangreiche Dokumentation innerhalb der Anwendung hilfreich. Außerdem wäre es denkbar, die Möglichkeiten der Programmoberfläche zu nutzen, um einen interaktiven Lehrgang anzubieten. Entsprechend ähnlichen Ansätzen aus anderen Programmen müsste dabei eine Abfolge von aufeinander aufbauenden Teilaufgaben der Modellierung vom Benutzer durchlaufen werden. Jeder Schritt muss so erklärt werden, dass der Nutzer am Ende des Lehrgangs ohne besonderen mentalen Aufwand (3.2) die Lerninhalte verinnerlicht hat. Die technische und methodische Komplexität dieses Vorschlages ist jedoch hoch.

Portierung
Mit der PACT-GUI können PACT-Programme nicht nur erstmals ohne jegliche Zusatzsoftware im Browser erstellt, sondern zudem auch rein graphisch modelliert werden. Vor allem letztere Möglichkeit stellt auch für erfahrene Entwickler von PACT-Programmen einen Mehrwert dar. Jene Entwickler haben daher vorgeschlagen, die Modellierungsfunktionalitäten der PACT-GUI in die Eclipse-IDE (4.3.4) zurück zu übertragen, um den alten, manuellen Erstellungsprozess (3.1) zu unterstützen.

Die Entwicklung eines Eclips-Plugins erfolgt auch in der Programmiersprache Java. Sie ist konzeptuell jedoch eher mit der Entwicklung einer Desktop-Anwendung vergleichbar als mit der GWT-basierten PACT-GUI-Webanwendung, weshalb nur ein geringes Poten-

tial für Softwarewiederverwendung besteht. Es können aber existierende Bibliotheken für die visuelle Graph-Modellierung in einem Eclipse-Plugin [The12e, The12f] genutzt werden. Inklusive des Einarbeitungsaufwandes ist allein für die Modellierungsgrundfunktion mit einer mehrwöchigen Entwicklungszeit zu rechnen. Ob der volle Funktionsumfang der PACT-GUI realisierbar ist, kann an dieser Stelle noch nicht abgeschätzt werden. In jedem Fall sollte kritisch abgewogen werden, ob der Aufwand nicht besser in die Weiterentwicklung der PACT-GUI investiert wäre.

Mit geringerem Aufwand wäre es möglich die PACT-GUI für Tablets zu optimieren. Dabei müssten vor allem die Bedienkonzepte einiger Steuerelemente angepasst werden. Beispielsweise würde das Scrollen mittels Scrollbalken am Fensterrand ersetzt durch direktes Bewegen der Arbeitsfläche. Auch hier ist jedoch abzuwägen, ob der Aufwand im Verhältnis zur zu erwartenden Nutzung steht.

Kapitel 6

Fazit

Das Ziel dieser Arbeit war es, einen Lösungsansatz zu erarbeiten, um den zeitaufwändigen und komplizierten Prozess der Erstellung von PACT-Programmen zu vereinfachen. Dieses Ziel wurde durch die prototypische Entwicklung der PACT-GUI-Webanwendung erreicht. Es ist nun erstmals möglich dem PACT-Programmiermodell entsprechende, valide PACT-Programme nur mittels eines Webbrowsers graphisch zu modellieren.

Es wurden die Probleme des ursprünglichen Erstellungsprozesses dargestellt und ein graphisches Modellierungswerkzeug als Lösungsansatz motiviert. Die Anforderungen an das zu entwickelnde Werkzeug wurden konzeptionell analysiert und anschließend in einem evolutionären Vorgehen anhand von laufender Software Schritt für Schritt verfeinert. Besonders intensiv wurde evaluiert, mit welchen Technologien als Basis der Anwendungsarchitektur bereits bekannte und bisher unbekannte, zukünftige Anforderungen am besten bewältigt werden können. Die Entscheidung zugunsten des Google Web Toolkits und damit für den innovativen Ansatz der Kompilierung von Java zu JavaScript, hat sich bewährt. Erst sie ermöglichte die effiziente Erstellung der PACT-GUI unter Verwendung bewährter Konzepte und Werkzeuge der Softwareentwicklung. Des weiteren bietet sie eine zukunftsfähige Basis zur Umsetzung der beschriebenen Weiterentwicklungsmöglichkeiten. Neben der neuartigen verwendeten Technologie war die Umsetzung der PACT-GUI anspruchsvoll, da sie durch die gleichzeitige Weiterentwicklung des Stratosphere-Systems in einem dynamischen, sich stetig wandelnden Umfeld stattfand.

Die konzeptionellen und technischen Lösungsansätze dieser Arbeit lassen sich auf ähnliche Anwendungsfälle der visuellen Modellierung von Datenflussgraphen übertragen. Die prototypische Umsetzung der PACT-GUI erlaubt weitere Erfahrungen bei der graphischen Modellierung von PACT-Programmen und der Interaktion mit einer Webbrowser-basierten Anwendungsoberfläche zu sammeln. Langfristig ist die Weiterentwicklung der PACT-GUI zur zentralen Benutzungsschnittstelle des Stratosphere-Projekts wünschenswert.

Literaturverzeichnis

[ABE+10] Alexander Alexandrov, Dominic Battré, Stephan Ewen, Max Heimel, Fabian Hueske, Odej Kao, Volker Markl, Erik Nijkamp, and Daniel Warneke. Massively parallel data analysis with pacts on nephele. In *VLDB Conference*, 2010.

[Ado12] Adobe Inc. Adobe Flash Player. `http://get.adobe.com/de/flashplayer`, 2012.

[AEH+11] Alexander Alexandrov, Stephan Ewen, Max Heimel, Fabian Hueske, Odej Kao, Volker Markl, Erik Nijkamp, and Daniel Warneke. Mapreduce and pact - comparing data parallel programming models. In *BTW*, 2011.

[Ama11] Amazon.com Inc. Amazon Elastic Compute Cloud. `http://amazon.com/ec2`, 2011.

[BAC+09] Philip A. Bernstein, Daniel J. Abadi, Michael J. Cafarella, Joseph M. Hellerstein, Donald Kossmann, and Samuel Madden. How best to build web-scale data managers? *Proc. VLDB Endow.*, 2:1647–1647, August 2009.

[Bal99] Helmut Balzert. *Lehrbuch Grundlagen der Informatik: Konzepte und Notationen in UML, Java und C++, Algorithmik und Software-Technik, Anwendungen.* Spektrum Lehrbücher der Informatik. Spektrum Akad. Verl., 1999.

[Bal00] Helmut Balzert. *Lehrbuch der Software-Technik: Software-Entwicklung.* Spektrum-Akademischer Vlg, 2.,Überarb. und erw. a. edition, 12 2000.

[Bal08] Helmut Balzert. *Lehrbuch Der Softwaretechnik: Softwaremanagement.* Spektrum Lehrbücher der Informatik. Spektrum Akademischer Verlag, 2008.

[Bal09] Helmut Balzert. *Lehrbuch Der Softwaretechnik: Basiskonzepte und Requirements Engineering.* Springer, September 2009.

[Bau08] Günther Bauer. *Architekturen für Web-Anwendungen: Eine praxisbezogene Konstruktions-Systematik.* Vieweg + Teubner, 2008.

[BBC+11] Alexander Behm, Vinayak R. Borkar, Michael J. Carey, Raman Grover, Chen Li, Nicola Onose, Rares Vernica, Alin Deutsch, Yannis Papakonstantinou, and Vassilis J. Tsotras. Asterix: towards a scalable, semistructured data platform for evolving-world models. *Distrib. Parallel Databases*, 29:185–216, June 2011.

[BCG+11] Vinayak R. Borkar, Michael J. Carey, Raman Grover, Nicola Onose, and Rares Vernica. Hyracks: A flexible and extensible foundation for data-intensive computing. In Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan, editors, *ICDE*, pages 1151–1162. IEEE Computer Society, 2011.

[Bec01] Kent Beck. Manifesto for Agile Software Development. `http://agilemanifesto.org`, 2001.

[BEH+10] Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, and Daniel Warneke. Nephele/PACTs: A Programming Model and Execution Framework for Web-Scale Analytical Processing. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC), pages 119-130*, 2010.

[Bes02] Klaus Beschorner. *Untersuchungen zur effizienten Kommunikation in komponentenbasierten Client/Server-Systemen.* PhD thesis, Eberhard-Karls-Universität zu Tübingen - Fakultät für Informatik, 2002.

[BHBE10] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. Haloop: Efficient iterative data processing on large clusters. *PVLDB*, 3(1):285–296, 2010.

[BJ87] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the eleventh ACM Symposium on Operating systems principles*, SOSP '87, pages 123–138, New York, NY, USA, 1987. ACM.

[Boe81] Barry W. Boehm. *Software engineering economics.* Prentice-Hall advances in computing science and technology series. Prentice-Hall, 1981.

[Boe88] Barry W. Boehm. A spiral model of software development and enhancement. *Computer*, 21:61–72, May 1988.

[Bra05] R. Brause. *Kompendium der Informationstechnologie: Hardware, Software, Client-Server-Systeme, Netzwerke, Datenbanken.* Xpert. press Series. Springer, 2005.

[BW99] G. Bannert and M. Weitzel. *Objektorientierter Softwareentwurf mit UML.* Addison-Wesley; Auflage: 2. Aufl., May 1999.

[CC08] Robert Cooper and Charlie Collins. *GWT in Practice.* Manning Pubs Co Series. Manning, 2008.

[CDG+08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, June 2008.

[Cha09] Sumit Chandel. Testing Methodologies Using Google Web Toolkit. `http://code.google.com/intl/de-DE/webtoolkit/articles/testing_methodologies_using_gwt.html`, March 2009.

[Cha12] Scott Chacon. Git: the fast version control system. `http://git-scm.com`, 2012.

[CJL+08] Ronnie Chaiken, Bob Jenkins, Pere-Ake Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. Scope: easy and efficient parallel processing of massive data sets. *Proc. VLDB Endow.*, 1:1265–1276, August 2008.

[Com07] Java Community. Java Specification Requests (JSR) 154 - Java Servlet Specification. `http://jcp.org/en/jsr/detail?id=154`, September 2007.

[Con11] Concurrent Inc. Cascading. `http://www.cascading.org`, 2011.

[DEF+08] Jürgen Dunkel, Andreas Eberhart, Stefan Fischer, Carsten Kleiner, and Arne Koschel. *Systemarchitekturen für Verteilte Anwendungen: Client-Server, Multi-Tier, SOA, Event Driven Architectures, P2P, Grid, Web 2.0*. Hanser Fachbuchverlag, 2008.

[DG04] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large cluster. *Commun. ACM*, 51:107–113, January 2008.

[DG10] Jeffrey Dean and Sanjay Ghemawat. System and method for efficient large-scale data processing. United States Patent 7,650,331 angemeldet am 18. Juni 2004, erteilt am 19. Januar 2010 in United States Pattern and Trademark Office to Google Inc. (Mountain View, CA), 2010.

[Die00] R. Diestel. *Graphentheorie*. Springer Lehrbuch. Springer, 2000.

[Dis11] Disco Project. Disco: massive data - minimal code. `http://discoproject.org/`, 2011.

[Dut94] S. Dutke. *Mentale Modelle: Konstrukte des Wissens und Verstehens: kognitionspsychologische Grundlagen für die Software-Ergonomie*. Arbeit und Technik. Verlag für Angewandte Psychologie, 1994.

[ELZ+10] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 810–818, New York, NY, USA, 2010. ACM.

[EP02] K. Erk and L. Priese. *Theoretische Informatik: Eine Umfassende Einführung*. Springer-Lehrbuch. Springer, 2002.

[Fab11] Szczepan Faber. Mockito: simpler and better mocking. `http://code.google.com/p/mockito/`, 2011.

[FJ04] Jayson Falkner and Kevin W. Jones. *Servlets and JavaServer Pages: the J2EE technology Web tier*. The DevelopMentor series. Addison-Wesley, 2004.

[For03] Neal Ford. *Art of Java web development: Struts, Tapestry, Commons, Velocity, JUnit, Axis, Cocoon, InternetBeans, WebWorks*. Manning Pubs Co Series. Manning, 2003.

[Fow06] Martin Fowler. Passive View. `http://martinfowler.com/eaaDev/PassiveScreen.html`, June 2006.

[FSV05] Andreas Fink, Gabriele Schneidereit, and Stefan Voß. *Grundlagen der Wirtschaftsinformatik*. Physica-Verlag, 2. auflage edition, 2005.

[Gal09] Michael Galpin. High-performance Web development with Google Web Toolkit and Eclipse Galileo. `http://www.ibm.com/developerworks/web/library/os-eclipse-googlegalileo/index.html`, October 2009.

[Gat10] Alan Gates. Pig and Hive at Yahoo! `http://developer.yahoo.com/blogs/hadoop/posts/2010/08/pig_and_hive_at_yahoo/`, August 2010.

[GB09] T. Grechenig and M. Bernhart. *Softwaretechnik: Mit Fallbeispielen aus realen Entwicklungsprojekten*. Pearson Studium. Pearson Studium, 2009.

[Geo09] Lars George. Hive vs. Pig. `http://www.larsgeorge.com/2009/10/hive-vs-pig.html`, October 2009.

[Geo11] Georgios Georgopoulos. Gwt Mosaic. `http://code.google.com/p/gwt-mosaic/`, 2011.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, October 2003.

[GNC+09] Alan F. Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M. Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava. Building a high-level dataflow system on top of map-reduce: the pig experience. *Proc. VLDB Endow.*, 2:1414–1425, August 2009.

[Goo11a] Google Inc. Coding Basics - JavaScript Native Interface (JSNI). `http://code.google.com/intl/de-DE/webtoolkit/doc/latest/DevGuideCodingBasicsJSNI.html`, 2011.

[Goo11b] Google Inc. GIN (GWT INjection) is Guice for Google Web Toolkit. `http://code.google.com/p/google-gin/`, 2011.

[Goo11c] Google Inc. Google Web Toolkit. `http://code.google.com/webtoolkit`, 2011.

[Goo11d] Google Inc. Google Web Toolkit: Widget List. `http://code.google.com/intl/de-DE/webtoolkit/doc/latest/RefWidgetGallery.html`, 2011.

[Goo12a] Google Inc. Coding Basics - Compatibility with the Java Language and Libraries. `http://code.google.com/intl/de-DE/webtoolkit/doc/latest/DevGuideCodingBasicsCompatibility.html`, 2012.

[Goo12b] Google Inc. Communicate with a Server. `http://code.google.com/intl/de-DE/webtoolkit/doc/latest/DevGuideServerCommunication.html`, 2012.

[Goo12c] Google Inc. Developer's Guide - Code Splitting. `http://code.google.com/intl/de-DE/webtoolkit/doc/latest/DevGuideCodeSplitting.html`, 2012.

[Gra01] M. Grand. *Java Enterprise Design Patterns*. Patterns in Java. Wiley, 2001.

[GU10] Daniel Guermeur and Amy Unruh. *Google App Engine Java and GWT Application Development*. Packt Publishing, November 2010.

[Guz10] Dominik Guzei. Source Code Editor in GWT. `http://code.google.com/p/gwt-codemirror-widget/`, 2010.

[Hav11] Marijn Haverbeke. Codemirror gwt-widget. `http://codemirror.net`, 2011.

[Hic11] Ian Hickson. Web Storage (W3C Candidate Recommendation). `http://www.w3.org/TR/webstorage/`, December 2011.

[HL97] A. Hügli and P. Lübcke. *Philosophielexikon: Personen und Begriffe der abendländischen Philosophie von der Antike bis zur Gegenwart*. Rowohlts Enzyklopädie. Rowohlt-Taschenbuch-Verlag, 1997.

[HN12] Arvid Heise and Felix Naumann. Integrating open government data with stratosphere for more transparency. 2012.

[Hof08] Dirk W. Hoffmann. *Software-Qualität*. Springer, 2008.

[Hor11] I. Horton. *Ivor Horton's Beginning Visual C++ 2008*. John Wiley & Sons, 2011.

[IBM11a] IBM Inc. IBM Blueworks Live. `https://www.blueworkslive.com`, 2011.

[IBM11b] IBM Inc. IBM Research: Jaql. `http://www.almaden.ibm.com/cs/projects/jaql/`, 2011.

[IBY+07] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, New York, NY, USA, 2007. ACM.

[IEE90] IEEE. *IEEE 610: Standard Glossary of Software Engineering Terminology*. Software Engineering Standard Committe of the IEEE Computer Society, New York, USA, September 1990.

[IEE98] IEEE. *IEEE 830: Recomended Practice for Software Requirements Specification*. Software Engineering Standard Committe of the IEEE Computer Society, New York, 1998.

[Int12] International Business Machines Corp. IBM Data Studio. `http://www.ibm.com/software/data/optim/data-studio/`, 2012.

[Iso11] Isomorphic Software Inc. SmartGwt. `http://www.smartclient.com/product/smartgwt.jsp`, 2011.

[Jen12] Jenkins CI Community. Jenkins Continuous Integration Server. `http://jenkins-ci.org`, 2012.

[JUn12] JUnit.org Community. JUnit.org - Resources for Test Driven Development. `http://www.junit.org`, 2012.

[Ker10] Federico Kereki. *Essential GWT: Building for the Web with Google Web Toolkit 2*. Developer's Library. ADDISON WESLEY (PEAR), 2010.

[Koc11] Peter-Paul Koch. Cross-Browser - Compatibility Master Table. `http://www.quirksmode.org/compatibility.html`, 2011.

[Lin07] Vinny Lingham. Top 20 Reasons why Web Apps are Superior to Desktop Apps. `http://www.vinnylingham.com/top-20-reasons-why-web-apps-are-superior-to-desktop-apps.html`, August 2007.

[LLC+12] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: a survey. *SIGMOD Rec.*, 40(4):11–20, January 2012.

[Mar11] Ganeshji Marwaha. GWT – Pros and Cons. `http://www.gmarwaha.com/blog/2011/05/09/gwt-pros-and-cons/`, May 2011.

[MBB07] D. Moore, R. Budd, and E. Benson. *Professional Rich Internet Applications: AJAX and Beyond*. Wrox professional guides. John Wiley & Sons, 2007.

[McC10] Ciarán McCann. Benchmark: Google Web Toolkit JavaScript vs Hand Crafted JavaScript. `http://goo.gl/R87KY`, July 2010.

[Met02] Steven J. Metsker. *Design patterns Java workbook*. Software Patterns Series. Addison-Wesley, 2002.

[Met10] Cade Metz. Google blesses Hadoop with MapReduce patent license. `http://www.theregister.co.uk/2010/04/27/google_licenses_mapreduce_patent_to_hadoop/`, April 2010.

[Mic12a] Microsoft Corporation. Microsoft .Net. `http://www.microsoft.com/net`, 2012.

[Mic12b] Microsoft Corporation. Microsoft Silverlight. `http://www.microsoft.com/silverlight`, 2012.

[ML08] Alex Moffat and Damon Lundin. 2008 Google I/O Session: Using GWT to Build a High Performance Collaborative Diagramming Tool. `http://googlewebtoolkit.blogspot.com/2007/10/lombardi-blueprint-built-with-gwt.html`, May 2008.

[Moc05] L. Moczar. *Tomcat 5.* Open source library. Addison-Wesley, 2005.

[Mor11] Mort Bay Consulting. Jetty. `http://jetty.codehaus.org`, 2011.

[MyS09] MySpace Inc. MySpace Qizmt. `http://qizmt.myspace.com/`, 2009.

[Nas03] Michael Nash. *Java frameworks and components: accelerate your Web application development.* Cambridge University Press, 2003.

[Ngu10] Tien Nguyen. 10 Best Java Web Development Framework. `http://lilylnx.wordpress.com/2010/05/19/10-best-java-web-development-framework/`, May 2010.

[Nie94] Jakob Nielsen. Ten Usability Heuristics. `http://www.useit.com/papers/heuristic/heuristic_list.html`, 1994.

[Nie03] Jakob Nielsen. Introduction to Usability. `http://www.useit.com/alertbox/20030825.html`, August 2003.

[Ora12a] Oracle Inc. Java. `http://www.java.com`, 2012.

[Ora12b] Oracle Inc. JavaDoc. `http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html`, 2012.

[Ora12c] Oracle Inc. MySQL - The world's most popular open source database. `http://www.mysql.com`, 2012.

[ORS+08] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1099–1110, New York, NY, USA, 2008. ACM.

[PDGQ05] Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with sawzall. *Sci. Program.*, 13:277–298, October 2005.

[PP04] G. Pomberger and W. Pree. *Software Engineering.* Hanser, 2004.

[PPR+09] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 35th SIGMOD international*

conference on Management of data, SIGMOD '09, pages 165–178, New York, NY, USA, 2009. ACM.

[Ram10] Chris Ramsdale. Large scale application development and MVP. `http://code.google.com/intl/de-DE/webtoolkit/articles/mvp-architecture.html`, March 2010.

[RAS] W.C. Richardson, D. Avondolio, and S. Schrager. *Professional Java JDK 6 Edition.*

[Ras10] Franck Rasolo. GWT RPC. `http://code.google.com/p/welikegwt-presentation/wiki/GwtRpc`, February 2010.

[Rec06] Peter Rechenberg. *Informatik-Handbuch.* Hanser, 2006.

[Red11] Red Hat. JBoss. `http://www.jboss.org`, 2011.

[Roy70] Winston W. Royce. Managing the development of large software systems: concepts and techniques. *Proc. IEEE WESTCON, Los Angeles*, pages 1–9, August 1970. Reprinted in *Proceedings* of the Ninth International Conference on Software Engineering, March 1987, pp. 328–338.

[Rub12] Ruby-Community. Ruby - A programmers best friend. `http://www.ruby-lang.org`, 2012.

[Rup07] Chris Rupp. *Requirements-Engineering und Management.* Hanser, München, 2007.

[Rya09] Ray Ryan. Google IO 2009: Best Practices For Architecting Your GWT App. `http://www.google.com/intl/de-DE/events/io/2009/sessions/GoogleWebToolkitBestPractices.html`, 2009.

[San11a] Sancha Inc. Ext GWT Internet Application Framework for Google Web Toolkit. `http://www.sencha.com/products/extgwt`, 2011.

[San11b] Jose Maria Arranz Santamaria. The Single Page Interface Manifesto. `http://itsnat.sourceforge.net/php/spim/spi_manifesto_en.php`, July 2011.

[San12a] Sancha Inc. Ext JS 4.1: JavaScript Framework for Rich Apps in Every Browser. `http://www.sencha.com/products/extjs/`, 2012.

[San12b] Sancha Inc. Sancha GXT Explorer Demo. `http://www.sencha.com/examples`, 2012.

[Sau11] Fred Sauer. GWT-DND: Drag-and-Drop Library for GWT. `http://code.google.com/p/gwt-dnd/`, 2011.

[Sch05] Steven M. Schafer. *Web standards programmer's reference: HTML, CSS, JavaScript, Perl, Python, and PHP.* Wrox programmer's references. Wiley Pub., 2005.

[Shi09] William Shields. Lost in Translation or Why GWT Isn't the Future of Web Development. `http://www.cforcoding.com/2009/10/lost-in-translation-or-why-gwt-isnt.html`, October 2009.

[Sky08] SkyNet Project. SkyNet - A Ruby MapReduce Framework. `http://skynet.rubyforge.org/`, March 2008.

[Sne83] Harry M. Sneed. Software-wartungsorganisation. In *Tagungsband des Struktur-Kongresses 83, Frankfurt*, 1983.

[Spe11] Geoff Speicher. Raphael-GWT: Raphaël JavaScript Library as a GWT Widget. `http://code.google.com/p/raphaelgwt/`, 2011.

[Sta73] H. Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, 1973.

[Sto07] Tom Stocky. Google Web Toolkit Blog: Lombardi Blueprint, built with GWT. `http://googlewebtoolkit.blogspot.com/2007/10/lombardi-blueprint-built-with-gwt.html`, October 2007.

[Str10] Stratosphere Initiative. About Page. `http://www.stratosphere.eu/about`, November 2010.

[The11a] The Apache Software Foundation. Apache Hadoop. `http://hadoop.apache.org`, 2011.

[The11b] The Apache Software Foundation. Apache Tomcat. `http://tomcat.apache.org`, 2011.

[The12a] The Apache Software Foundation. Apache HTTP Server Project. `http://httpd.apache.org`, 2012.

[The12b] The Apache Software Foundation. Apache Maven Project. `http://maven.apache.org`, 2012.

[The12c] The Apache Software Foundation. HBase - The Hadoop database. `http://hbase.apache.org`, 2012.

[The12d] The Apache Software Foundation. Powered By Hadoop. `http://wiki.apache.org/hadoop/PoweredBy`, February 2012.

[The12e] The Eclipse Foundation. Eclipse Modeling Project. `http://www.eclipse.org/modeling/`, 2012.

[The12f] The Eclipse Foundation. Graphical Editing Framework. `http://www.eclipse.org/gef/`, 2012.

[The12g] The Eclipse Foundation. Homepage. `http://www.eclipse.org`, 2012.

[TSJ+09] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive- a warehousing solution over a map-reduce framework. In *IN VLDB '09: PROCEEDINGS OF THE VLDB ENDOWMENT*, pages 1626–1629, 2009.

[Vaa11] Vaadin Ltd. Vaadin Java Framework. `http://vaadin.com`, 2011.

[Vai10] Gaurav Vaish. GCodeMirror - CodeMirror for GWT. `http://code.google.com/p/gcodemirror/`, 2010.

[Vec11] Vectomatic. SVG GWT Libriary. `http://www.vectomatic.org/libs/lib-gwt-svg`, 2011.

[vL10] Frank von Lachmann. *Interaktionen in HTML-basierten Applikationsoberflächen: Realisierung von interaktiven Menus unter Berücksichtigung verschiedener Anforderungsschichten*. GRIN Verlag GmbH, 2010.

[VMw12] VMware Inc. WmWare. `http://www.vmware.com`, 2012.

[Wä10] Kai Wähner. Categorization and Comparison of Popular Web Frameworks in the Java / JVM Environment. `http://java.dzone.com/articles/categorization-web-frameworks`, December 2010.

[Wat08] Kelly Waters. All about agile: Writing Good User Stories. `http://www.allaboutagile.com/writing-good-user-stories`, April 2008.

[Wik11] Wikipedia Foundation. Comparison of Web application frameworks. `http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks`, 2011.

[Wil08] Greg Wilkins. Jetty vs Tomcat: A Comparative Analysis. `http://www.webtide.com/choose/jetty.jsp`, May 2008.

[WK09] Daniel Warneke and Odej Kao. Nephele: efficient parallel data processing in the cloud. In *SC-MTAGS*, 2009.

[YDHP07] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D. Stott Parker. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 1029–1040, New York, NY, USA, 2007. ACM.

[YIF+08] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. Dryadlinq: a system for general-purpose distributed data-parallel computing using a high-level language. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 1–14, Berkeley, CA, USA, 2008. USENIX Association.

[Zor10] Kelby Zorgdrager. Choosing the Right Java Web Development Framework. `http://olex.openlogic.com/wazi/2010/choosing-the-right-java-web-development-framework/`, July 2010.

Glossar

Ajax (Asynchronous *JavaScript* and *XML*) bezeichnet eine Technik der asynchronen Datenübertragung zwischen Webserver und *Webbrowser*, wodurch eine *HTTP*-Anfrage gestellt werden kann, ohne dass die komplette Webseite neu geladen werden muss. Siehe Kapitel 2.3.2.

API (Application Programming Interface; deutsch: Schnittstelle zur Anwendungsprogrammierung) ist ein Programmteil, der von einem Softwaresystemen bereitgestellt wird, um die Nutzung des *Systems* durch andere Systeme oder Entwickler zu ermöglichen.

Architektur /Software-Architektur ist gegeben durch funktionstragende Komponenten und die strukturbestimmenden Beziehungen des *Systems*. Siehe Kapitel 4.4.

CSS (Cascading Style Sheets) ist eine standardisierte, deklarative Sprache zur Formatierung von *HTML*-Dokumenten. Siehe Kapitel 2.3.2.

DAG (Directed Acyclic Graph; zu deutsch: gerichteter, azyklischer Graph) ist ein *Graph*, der gerichtet und frei von Zyklen ist. Siehe Kapitel 2.2.1.

Datenbanksystem ist ein System zur dauerhaften, effizienten, widerspruchsfreien Speicherung und Verwaltung großer Datenmengen. Außerdem realisiert es die bedarfsgerechte Bereitstellung von Teilmengen dieser Daten für Benutzer und Anwendungsprogramme. Siehe Kapitel 2.1.1.

Datenflussgraph ist ein *Graph*, der Abhängigkeiten von Einheiten aufgrund des Transfers von Daten verdeutlicht. Knoten bilden beliebige datenverarbeitende oder -speichernde Einheiten, Kanten verdeutlichen die Übertragung von Daten. Siehe Kapitel 2.2.1.

Datenverarbeitung bezeichnet den organisierten Umgang mit Datenmengen, um sie zu analysieren oder zu verändern. Siehe Kapitel 2.1.1.

Framework (Softwareframework) ist ein Programmiergerüst, welches einen wiederverwendbaren generischen Entwurf eines Softwaresystemes für eine Klasse von Anwendungsfällen entsprechend dem komponentenbasierten Entwicklungsmodell (2.4.1) bereitstellt. Siehe Kapitel 4.3.2. Im allgemeinen Sinne stellt ein Framework einen Ord-

nungsrahmen dar. Ein *Map/Reduce*-Framework bezeichnet beispielsweise eine Laufzeitumgebung zur Ausführung von *Map/Reduce*-Programmen. Siehe Kapitel 2.1.3.

Graph ist in der Graphentheorie eine abstrakte Struktur, die eine Menge von Objekten und Verbindungen dieser Objekte repräsentiert. Die Objekte werden Knoten genannt, die paarweisen Verbindungen zwischen Knoten heißen Kanten. Siehe Kapitel 2.2.1.

GUI (Graphical User Interface; zu deutsch graphische Benutzungsoberfläche oder auch Benutzeroberfläche) beschreibt den Teil einer Software, die dem Benutzer die Interaktion mit einem Computersystem mittels graphischer Elemente wie Symbole, Menüs, Eingabefelder, Knöpfe, Listen und Fenster erlaubt. Siehe Kapitel 2.3.1.

GWT (Google Web Toolkit) [Goo11c] ist ein kostenloses, quelloffenes *Java-Framework* zur Erstellung von Rich=Client=Webanwendungen der Firma Google Inc. Siehe Kapitel 4.3.3.

GXT [San11a] ist die komplett Java-basierte *GWT* 3-Adaption des *JavaScript Rich Client Frameworks* Ext JS [San12a]. Sie bietet eine große Menge qualitativ hochwertiger und visuell ansprechender *GUI-Steuerelemente* (Widgets). Siehe Kapitel 4.3.3.

HTML (Hypertext Markup Language) ist eine textbasierte, an *XML* angelehnte Auszeichnungssprache zur semantischen Strukturierung der Inhalte einer Webseite. Siehe Kapitel 2.3.2.

JAR (Java ARchiv) ist ein komprimiertes Dateiformat zur Verteilung von Programmen und Klassenbibliotheken, welches zusätzlich Metadaten in einer eingebetteten Manifest-Datei enthalten kann. Siehe Kapitel 3.1.

Java ist eine objektorientierte, typisierte Programmiersprache. Siehe Kapitel 4.3.1.

JavaScript ist eine im Browser ausgeführte Programmiersprache zur Dynamisierung von Webseiten. Siehe Kapitel 2.3.2.

JDBC (Java Database Connectivity) ist eine universelle Schnittstelle zwischen *Datenbanken* und *Java*-Anwendungen. Sie orientiert sich stark an *ODBC*, wurde jedoch an das objektorientierte Design von *Java* angepasst.

JSON (JavaScript Object Notation) ist eine Datendarstellungsform für halbstrukturierte Daten in Form von gültigem *JavaScript*.

JVM (Java Virtual Machine) ist eine Software, die zur Ausführung von *Java*-Programmen benötigt wird. Java-Programme können auf jedem System (z.B. Windows, Linux, Palm OS uvm.) ausgeführt werden, für das eine JVM verfügbar ist. Die JVM ist Bestandteil der Java Runtime Environment.

Map/Reduce ein Programmiermodell zur verteilten, parallelen Verarbeitung großer semistrukturierter Datenmengen. Siehe Kapitel 2.1.

Modell ist nach der allgemeinen Modelltheorie [Sta73] eine Abbildung der Wirklichkeit, wobei entsprechend dem Verwendungszweck (charakterisiert durch das Frage-Quadrupel: Wovon? Für wen? Wann? Wozu?) bestimmte Attribute der Wirklichkeit reduziert oder betont werden [Dut94].

MVC (Model-View-Controller) ist ein *Architekturmuster* der Softwareentwicklung zur Strukturierung einer Softwarekomponente in die Bereich Model (Datenmodell), View (Presentation) und Controller (Steuerung). Siehe Kapitel 2.3.3.

MVP (Model-View-Presenter) ist ein *Architekturmuster* der Softwareentwicklung zur Strukturierung einer Softwarekomponente in die Bereich Model (Datenmodell), View (Presentation) und Presenter (Steuerung). Siehe Kapitel 2.3.3.

Nephele ist eine parallele Ausführungsumgebung innerhalb des *Statosphere*-Projekts für gerichtete *Datenflussgraphen*, formuliert als Nephele-*DAG*s. Siehe Kapitel 2.2.3.

ODBC (Open Database Connectivity) ist eine von Microsoft entwickelte, standardisierte Sammlung von Funktionen, die den Zugriff auf die Daten einer *Datenbank* mit *SQL* ermöglichen.

Open-Source beschreibt die freie Verbreitung und Zugänglichkeit zu den Entwurf- und Implementierungsdetails einer Software, wodurch neue Wege der gemeinschaftlichen Weiterentwicklung und Qualitätssicherung ermöglicht werden.

PACT (Parallelization Contract) ist eine Funktion zweiter Ordnung zur Aufteilung von Eingabedaten in *PUs*. PACTs sind Teil des *PACT-Programmiermodells*. Siehe Kapitel 2.2.1.

PACT-Programm ist ein gerichteter, azyklischer *Datenflussgraph*. Er besteht aus *PACTs* welche die Datenverarbeitung realisieren, und gerichteten Kanten, welche den Datenfluss abbilden. Siehe Kapitel 2.2.1.

PACT-Programmiermodell ist ein Modell zur verteilten Verarbeitung großer semistrukturierter Datenmengen innerhalb des *Stratosphere*-Projekts. Siehe Kapitel 2.2.1.

PDF (Portable Document Format) ist ein plattformunabhängiges, weltweit gebräuchliches Austauschformat für elektronische Dokumente.

PU (Parallelization Unit; zu deutsch: Parallelisierungseinheit) heißt ein Einzelteil einer Datenmenge, das unabhängig vom Kontext der Gesamtdaten verarbeitet werden kann. Siehe Kapitel 2.1.2.

SQL (inoffizielle Abkürzung für Structured Query Language) ist eine deklarative Sprache zur Definition von Datenstrukturen in relationalen *Datenbanken* sowie zum Einfügen, Verändern, Löschen und Abfragen von darauf basierenden Datenbeständen.

Steuerelement (englisch: Widget) ist ein Interaktionselement in einer *GUI*, beispielsweise eine Schaltfläche, ein Reitermenü oder eine Tabelle.

Stratosphere ist ein gemeinschaftliches Forschungsprojekt verschiedener deutscher Universitäten zur Entwicklung eines von *Map/Reduce* und *Datenbanksystemen* beeinflussten *Systems* zur Verarbeitung großer Datenmengen. Siehe Kapitel 1 und 2.2.

System ist ein Komplex von Elementen, die miteinander verbunden und voneinander abhängig sind und insofern eine strukturierte Ganzheit bilden [...]; ein geordnetes Ganzes, dessen Teile nach bestimmten Regeln, Gesetzen oder Prinzipien ineinandergreifen [HL97].

UDF (User-Defined Function; deutsch: benutzerdefinierte Funktion) enthält die in *Java* programmierte Verarbeitungslogik eines *PACTs*. Siehe Kapitel 2.2.1.

UML (Unified Modelling Language) ist eines Sprache und Notation zur Spezifikation, Konstruktion Visualisierung und Dokumentation von Modellen für die Softwareentwicklung [BW99].

URL (Uniform Ressource Locator) beschreibt Zugriffsmethode und Ort einer Ressource innerhalb von Computernetzwerken. Im allgemeinen Sprachgebrauch gleichbedeutend mit Internetadressen, wie beispielsweise http://www.example.com/index.html. Siehe Kapitel 2.3.2.

User Story (deutsch: „Benutzergeschichten") ist eine kurze, natürlichsprachlich formulierte Software-Anforderung aus dem Bereich der agilen Softwareentwicklung. Siehe Kapitel 4.1.2.

Webbrowser (verkürzt Browser) bezeichnet eine Software zur Darstellung von Webseiten und Medieninhalten. Siehe Kapitel 2.3.2.

Webserver bezeichnet einen Computer der statische und dynamische Dateien an Clients wie zum Beispiel einen *Webbrowser* ausliefert, um eine Webseite bereitzustellen. Siehe Kapitel 2.3.1.

XML (Extensible Markup Language) ist eine Auszeichnungssprache zur Darstellung von hierarchisch strukturierten Daten in Textform. Die Sprache ist für Menschen lesbar sowie plattform- und programmiersprachenunabhängig. Siehe Kapitel 4.4.2.

Anhang A

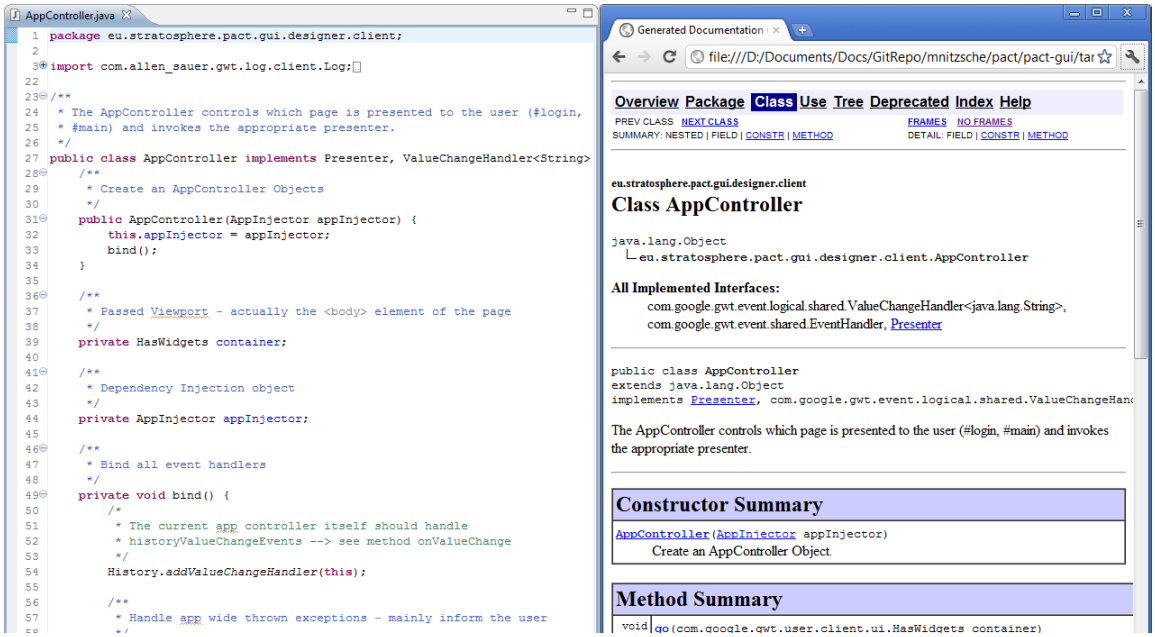
Anhang

A.1 Quellcode

Der Quellcode der PACT-GUI befindet sich auf dem beigelegten Datenträger im Verzeichnis */stratosphere/pact/pact-gui*. Das Hauptverzeichnis */stratosphere* ist ein Eclipse-IDE-Projekt (4.3.4) und kann in die Entwicklungsumgebung importiert werden. Die PACT-GUI-Anwendung kann mit Hilfe des Build-Werkzeugs Maven (4.3.4) und dem Befehl *"mvn clean compile gwt:compile package"* erstellt werden. Um die Anwendung auszuführen, muss die Datei */stratosphere/pact/pact-gui/target/pact-gui-0.1.1.war* in eine Ausführungsumgebung für Java-Webanwendungen wie dem Apache Tomcat [The11b] deployed werden.

A.2 Quellcodedokumentation mit JavaDoc

Java-Quellcode links und die daraus generierte JavaDoc-Dokumentation rechts:



AppController.java

```
1  package eu.stratosphere.pact.gui.designer.client;
2
3  import com.allen_sauer.gwt.log.client.Log;
22
23 /**
24  * The AppController controls which page is presented to the user (#login,
25  * #main) and invokes the appropriate presenter.
26  */
27 public class AppController implements Presenter, ValueChangeHandler<String>
28     /**
29      * Create an AppController Objects
30      */
31     public AppController(AppInjector appInjector) {
32         this.appInjector = appInjector;
33         bind();
34     }
35
36     /**
37      * Passed Viewport - actually the <body> element of the page
38      */
39     private HasWidgets container;
40
41     /**
42      * Dependency Injection object
43      */
44     private AppInjector appInjector;
45
46     /**
47      * Bind all event handlers
48      */
49     private void bind() {
50         /*
51          * The current app controller itself should handle
52          * historyValueChangeEvents --> see method onValueChange
53          */
54         History.addValueChangeHandler(this);
55
56         /**
57          * Handle app wide thrown exceptions - mainly inform the user
```

Generated Documentation

file:///D:/Documents/Docs/GitRepo/mnitzsche/pact/pact-gui/tar

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

eu.stratosphere.pact.gui.designer.client

Class AppController

java.lang.Object
└ eu.stratosphere.pact.gui.designer.client.AppController

All Implemented Interfaces:

com.google.gwt.event.logical.shared.ValueChangeHandler<java.lang.String>, com.google.gwt.event.shared.EventHandler, Presenter

```
public class AppController
extends java.lang.Object
implements Presenter, com.google.gwt.event.logical.shared.ValueChangeHan
```

The AppController controls which page is presented to the user (#login, #main) and invokes the appropriate presenter.

Constructor Summary

AppController(AppInjector appInjector)
Create an AppController Object.

Method Summary

void go(com.google.gwt.user.client.ui.HasWidgets container)

A.3 Vergleich externer Faktoren zwischen GWT und Vaadin

Stand: 1. Oktober 2011

Herstellerfirmen

GWT	Vaadin
Google Inc. (Sitz in Californien USA; knapp 29.000 Angestellte und 30 Mrd. Jahresumsatz)	Vaadin Ltd. (Sitz in Turku, Finnland; ca. 50 Angestellte)

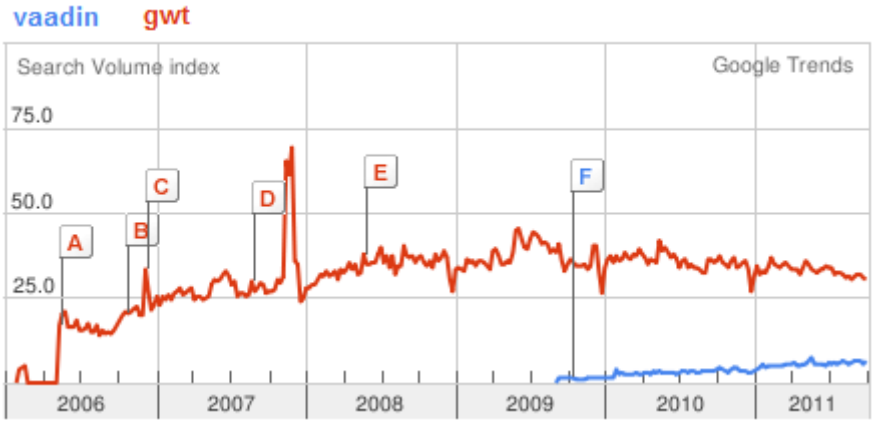
Entwicklergemeinschaft

	GWT-Forum	Vaadin-Forum
Mitglieder	27.420	2.708
Einträge	133.985	28.918

Quelle: `http://groups.google.com/group/google-web-toolkit/about`, `https://vaadin.com/forum/-/message_boards?_19_topLink=statistics`

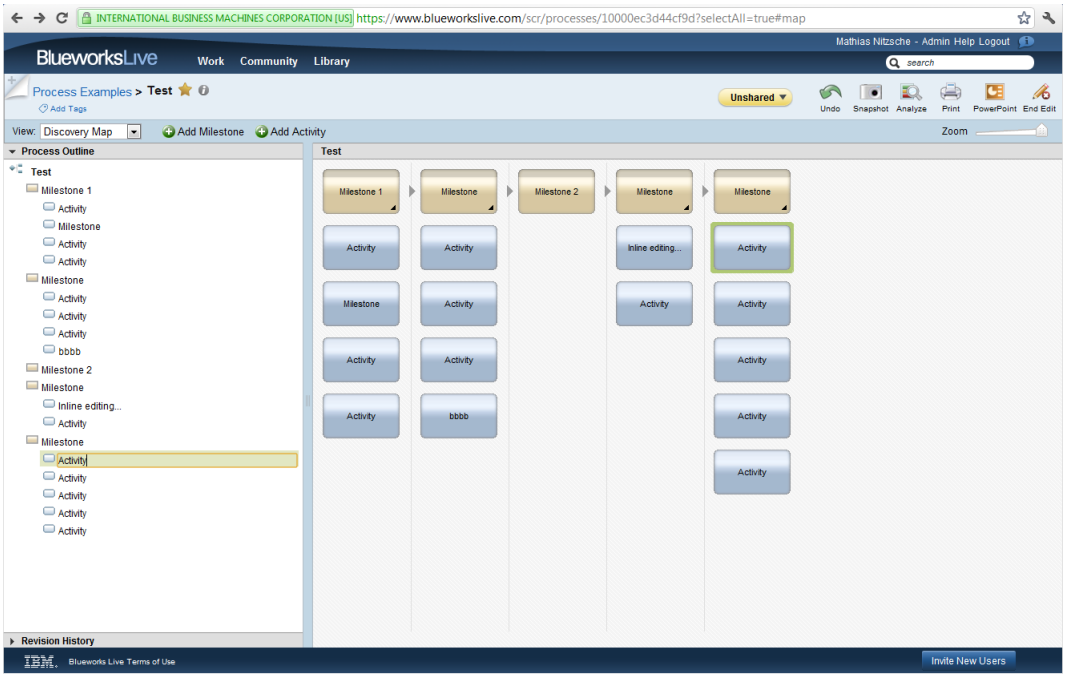
Suchaufkommen

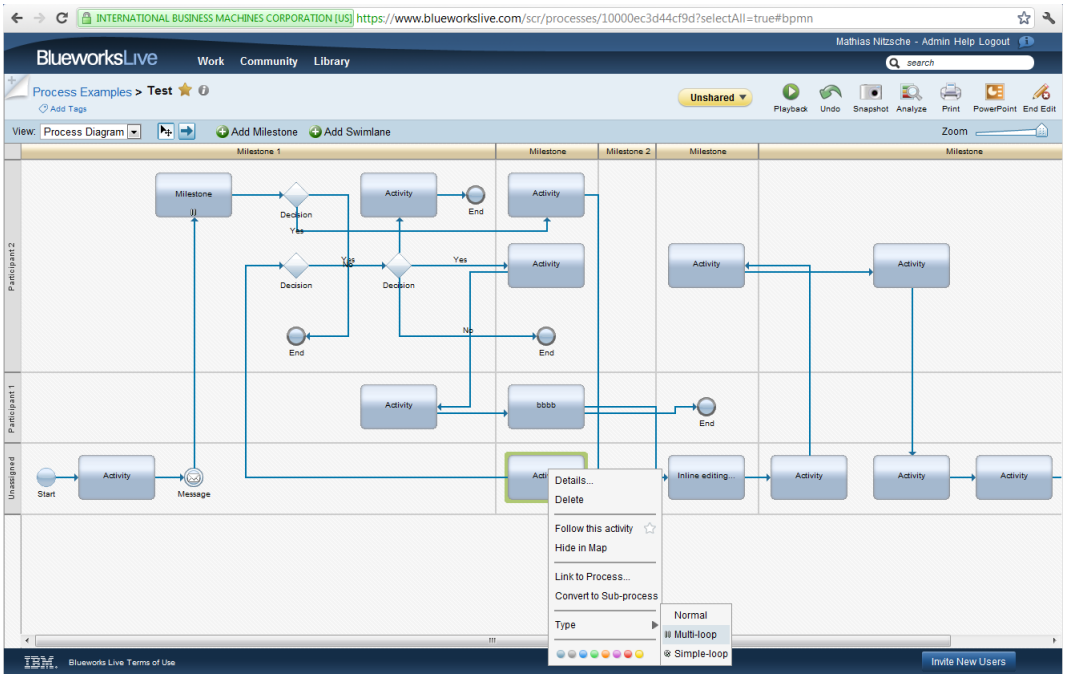
Nach Google Trends wird nach *GWT* ca. 5 mal häufiger als nach *Vaadin* gesucht.
Quelle: `http://www.google.de/trends?q=vaadin%2C+gwt`



A.4 IBM Blueworks Live

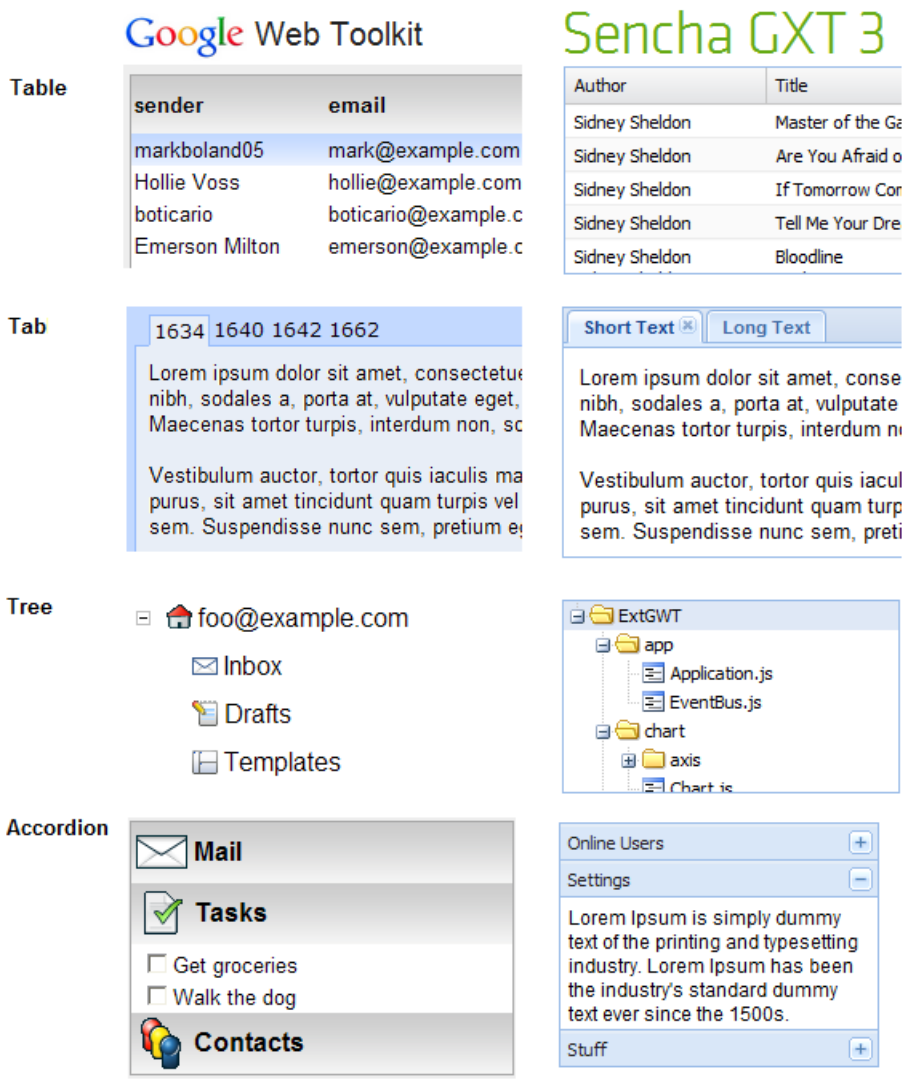
Screenshots der GWT-basierten Software *IBM Blueworks Live*; Quelle: [IBM11a]





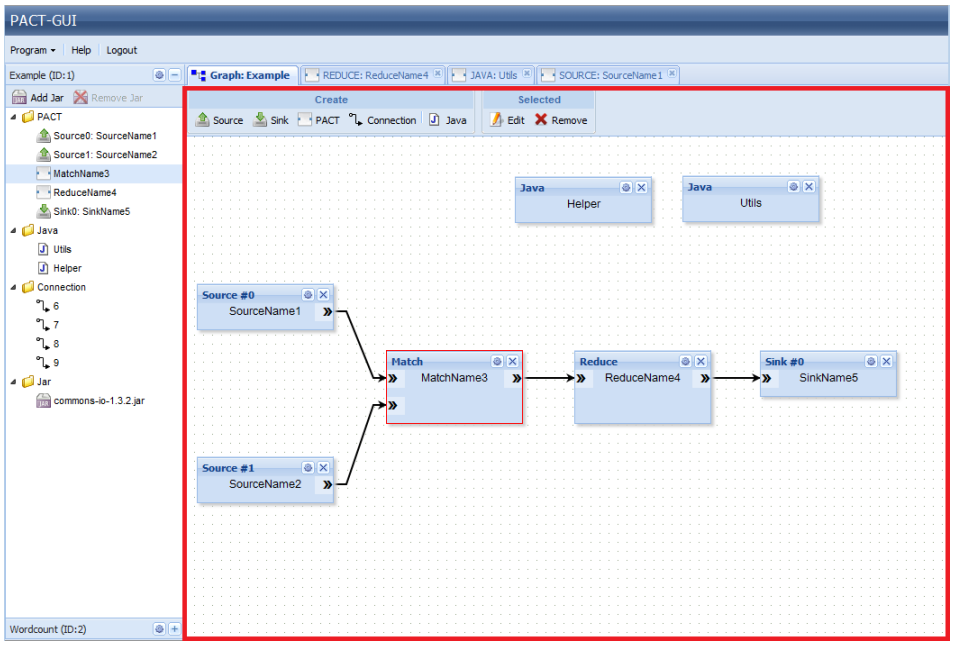
A.5 Widgetvergleich zwischen GWT und GXT

Quellen: GWT [Goo11d]; GXT [San12b]

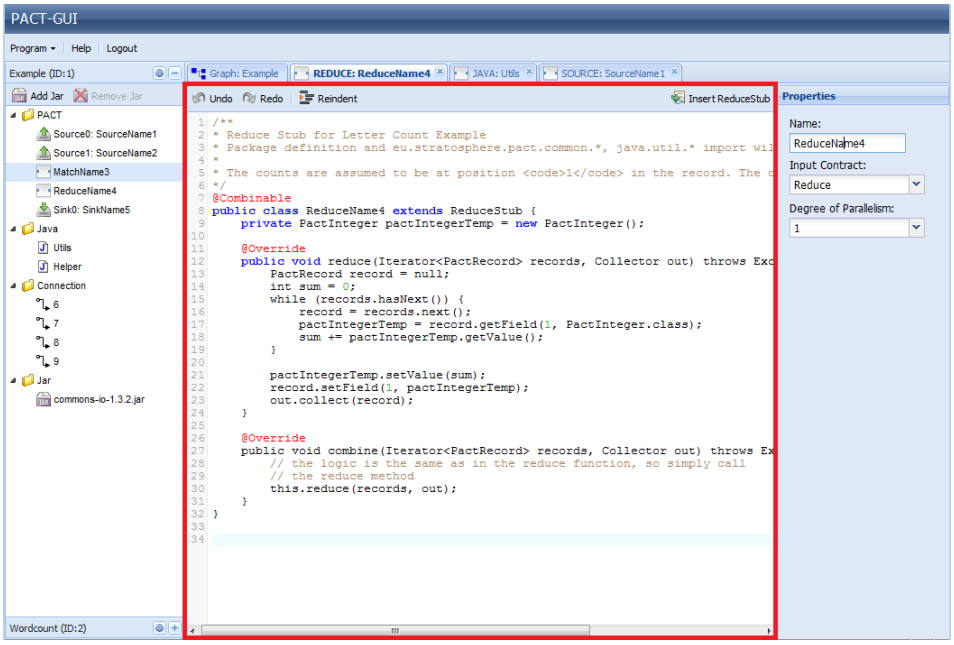


A.6 Screenshots der PACT-GUI

Graph-Editor-Widget (rot umrahmt):

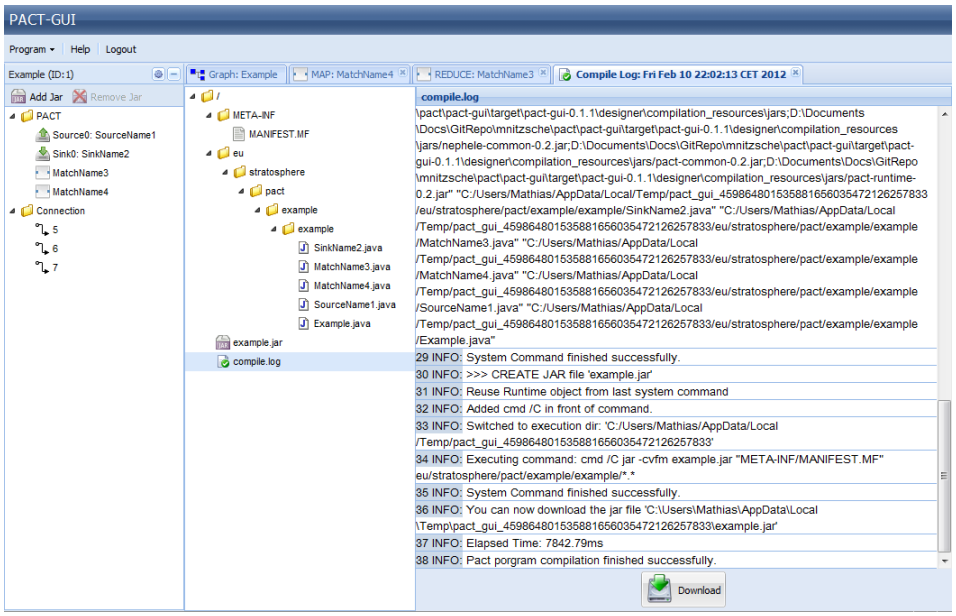


Java-Editor-Widget (rot umrahmt):

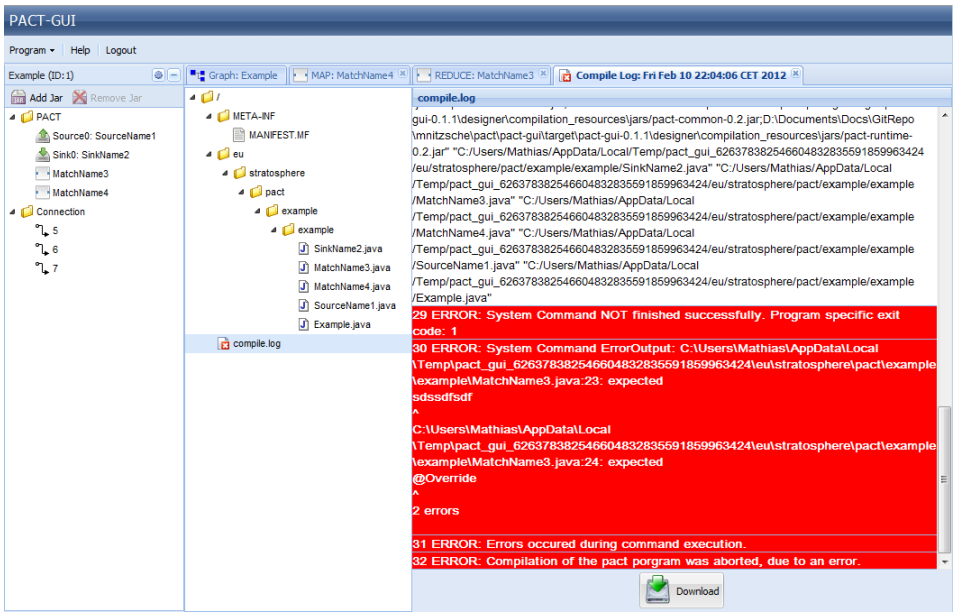


A.7 Screenshots der Generierung eines PACT-Programms

Erfolgreiche Generierung; Jar-Datei wurde erzeugt



Fehlgeschlagene Generierung; Jar-Datei wurde nicht erzeugt; Anzeige Fehlerprotokoll



A.8 Serialisiertes PACT-Programm

XML-Dateiausschnitt:

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0_19" class="java.beans.XMLDecoder">
 <object id="PactProgram0" class="eu.stratosphere.pact.gui.designer.shared.model.PactProgram">
  <void property="connections">
   <void method="put">
    <int>5</int>
    <object class="eu.stratosphere.pact.gui.designer.shared.model.Connection">
     <void property="fromPact">
      <object id="Pact0" class="eu.stratosphere.pact.gui.designer.shared.model.Pact">
       <void property="id">
        <int>1</int>
       </void>
       <void property="javaCode">
        <string> /* Converts a input line, assuming to contain a string, into a record that
 * has a single field, which is a {@link PactString}, containing that line.
 */
 public class TextInput extends DelimitedInputFormat {
private final PactString lineStr = new PactString();

 @Override
 public boolean readRecord(PactRecord record, byte[] line, int numBytes) {
lineStr.setValueAscii(line, 0, numBytes);
 record.setField(0, lineStr);
 return true;
 }
 }</string>
       </void>
       <void property="name">
        <string>TextInput</string>
       </void>
       <void property="pactProgram">
        <object idref="PactProgram0"/>
       </void>
       <void property="type">
        <object class="eu.stratosphere.pact.gui.designer.shared.model.PactType" method="valueOf">
         <string>SOURCE</string>
        </object>
       </void>
      </object>
     </void>
     <void property="fromX">
      <int>150</int>
     </void>
     <void property="fromY">
      <int>30</int>
     </void>
     <void property="id">
      <int>5</int>
     </void>
     <void property="toPact">
      <object id="Pact1" class="eu.stratosphere.pact.gui.designer.shared.model.Pact">
       <void property="id">
        <int>2</int>
       </void>
       <void property="javaCode">
```

A.9 TestCase: Main Presenter

```
public class MainPresenterTest {
    AppInjector appInjectorMock = TestAppInjectorFactory.getNewAppInjectorForTest();

    /**
     * JUnit Test for the Main Presenter
     * Programlist and TabContainer are added in View to MainMenu
     * Presenter Logic mainly consists of main menu
     */
    @Test
    public void test() {
        // create MainPresenter - pass mock object
        MainPresenter mainPresenter = new MainPresenter(appInjectorMock, appInjectorMock.getMainView());

        // assure that no programs exist in the beginning
        assertEquals(appInjectorMock.getPactProgramManager().getPactPrograms().size(), 0);

        // create new test program (check if program size is 1 afterwards)
        appInjectorMock.getPactProgramManager().addTestPactProgram();
        assertEquals(appInjectorMock.getPactProgramManager().getPactPrograms().size(), 1);

        // create no program with invalid name (no program added)
        // simuating "Enter name for new pact Program"-PopUp
        try {
            mainPresenter.action_AddPactProgram("INCORRECT PACT NAME )?=");
        } catch (PactValidationException e) {

        }
        assertEquals(appInjectorMock.getPactProgramManager().getPactPrograms().size(), 1);

        // add new program by name (click on new menu item, with valid entry)
        try {
            mainPresenter.action_AddPactProgram("MyProgram");
        } catch (PactValidationException e) {

        }
        assertEquals(appInjectorMock.getPactProgramManager().getPactPrograms().size(), 2);

        // create pact program via pactProgramManager (open/load menu item)
        PactProgram existingPactProgram = null;
        try {
            existingPactProgram = appInjectorMock.getPactProgramManager().createNewPactProgram("MyProg2");
        } catch (PactValidationException e) {

        }
        assertEquals(appInjectorMock.getPactProgramManager().getPactPrograms().size(), 3);

        // add again existing program (Should not work)
        mainPresenter.action_AddExistingPactProgram(existingPactProgram);
        assertEquals(appInjectorMock.getPactProgramManager().getPactPrograms().size(), 3);

        // add null pact program by object (Should not work)
        mainPresenter.action_AddExistingPactProgram(null);
        assertEquals(appInjectorMock.getPactProgramManager().getPactPrograms().size(), 3);

        // add pact program by object (load menu item)
        PactProgram newPactProgram = new PactProgram();
        mainPresenter.action_AddExistingPactProgram(newPactProgram);
        assertEquals(appInjectorMock.getPactProgramManager().getPactPrograms().size(), 4);

        // remove pact program (close menu item)
        mainPresenter.action_ClosePactProgram(existingPactProgram);
        assertEquals(appInjectorMock.getPactProgramManager().getPactPrograms().size(), 3);
        mainPresenter.action_ClosePactProgram(existingPactProgram);
        assertEquals(appInjectorMock.getPactProgramManager().getPactPrograms().size(), 3);

        // remove pact program (close menu item)
        mainPresenter.action_ClosePactProgram(newPactProgram);
        assertEquals(appInjectorMock.getPactProgramManager().getPactPrograms().size(), 2);
    }
}
```

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Masterarbeit in diesem Studiengebiet erstmalig einzureichen.

Berlin, den 31. Mai 2012

Mathias Nitzsche

Statement of authorship

I declare that I completed this thesis on my own and that information which has been directly or indirectly taken from other sources has been noted as such. Neither this nor a similar work has been presented to an examination committee.

Berlin, den 31. Mai 2012

Mathias Nitzsche