Frontend eines Agilent-Parsers erstellen

Belegarbeit

eingereicht am Fachbereich

Informatik

der Hochschule Zittau/Görlitz (HAW)

als Prüfungsleistung im Fach

Data Mining 2

vorgelegt von:

Christof Ochmann (35989)

Ingo Körner (40586)

Görlitz, 07. Februar 2013

Betreuer: Prof. ten Hagen

Abstract

In diesem Projekt...

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

JVM	Java Virtual Machine

Literaturverzeichnis

[1] Martin, Robert C. (2008): Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall International

[2] Freeman, Eric (2007): Entwurfsmuster von Kopf bis Fuß. O'REILLY

[3] `http://www.cs.waikato.ac.nz/ml/weka/arff.html (08.06.2012)`

1 Allgemeines

1.1 Einleitung

Ziel dieses Projektes ist es, das Frontend für einen Agilent-Compiler zu schreiben. Dabei soll das Frontend Agilent-Logdateien einlesen und sie in ein Intermediate-Format umwandeln.

1.2 Aufgabenstellung

In diesem Projekt sollen gegebene Agilent-Logdateien geparst werden. Aus den geparsten Zeilen soll ein Baum in einem Intermediate-Format erstellt werden. Der erzeugte Baum wird im Hauptspeicher des Rechners gehalten. Die Weiterverarbeitung des Baumes im Intermediate-Format erfolgt in einem anderen Projekt und ist nicht Gegenstand dieser Arbeit. In diesem Projekt muss das Intermediate-Format nicht entworfen werden. Statt dessen wird es von Felix Deutschmann und Daniel Horbach übernommen. Ein Baum im Intermediate-Format wird immer nur aus einer Agilent-Logdatei erzeugt, d.h. es soll nicht ein Baum aus zwei oder mehreren Agilent-Logdateien erzeugt werden. $Agilent - Logdatei - > Agilent - Parser - > Baum im Intermediate - Format$.

Das Frontend soll nur Knotennamen berücksichtigen, die in den zur Verfügung stehenden Agilent-Logdateien auch vorkommen. Andere Knotennamen brauchen im Frontend nicht implementiert werden. Treten bei der Verarbeitung einer Agilent-Logdatei einmal unerwartete Knotennamen auf, werden diese in der Datei Unsup-

portedNodeNames.txt weggeschrieben.

Die Reihenfolge der Kindknoten spielt bei der Erstellung des Baumes keine Rolle.

1.3 Relevanz des Forschungsgegenstandes

Der Forschungsgegenstand dieser Arbeit ist, ein Compiler-Frontend für Agilent-Logdateien zu erstellen. Der Forschungsgegenstand ist relevant, da bisher kein Compiler-Frontend für das Umwandeln von Agilent-Logdateien in das Intermediate-Format vorliegt. Darüberhinaus müssen für das Erstellen des Frontends technische Probleme gelöst werden. Ziel der Forschung ist es, einen Frontend zu entwerfen, dass das Agilent-Logformat in ein Intermediate-Format überführt.

1.4 Der aktuelle Wissensstand

Noch nicht vorhandene Kenntnisse über das Parsen von Agilent-Logdateien werden aus der Format6.pdf gewonnen. In der Format6.pdf wird das Agilent-Logformat beschrieben.

1.5 Hintergrund

Ein Compiler hat im Allgemeinen zwei Phasen: das Frontemd und das Backend. In diesem Projekt soll ein Frontend für einen Compiler erstellt werden. In diesem Frontend wird ein sogenannter Logdatei-Parser eingesetzt. Er liest eine Eingabe im Agilent-Logformat und erstellt daraus ein Syntax-Baum in einem gegebenen Zwischenformat. In einem Folgeprojekt wird ein Compiler-Backend erstellt, dass den vom Frontend erzeugten Zwischencode nimmt und daraus die gewünschte Ausgabe in einem Zielformat schreibt.

Da das Agilent-Logformat nur halbformal in einer pdf-Datei beschrieben ist und

es keine formale Grammatik für das Format gibt, können keine Parsergeneratoren wie JavaCC oder yacc eingesetzt werden, sondern es wird hier für das Agilent-Logformat ein Parser von Hand geschrieben.

2 Umsetzung

2.1 Analyse

In Abbildung 2.1 auf Seite 4 ist das Analyseklassendiagramm vom Agilent-Parser zu sehen.

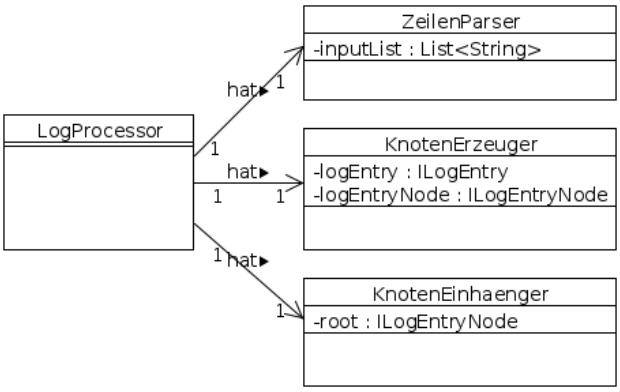

Abbildung 2.1: Analyseklassendiagramm Agilent-Parser

2.2 FunktionaleAnforderungen

Der Parser soll eine Agilent-Logdatei einlesen und sie in ein gegebenes Intermediate-Format umwandeln. Dabei sollen nur die Agilent-Knotennamen berücksichtigt werden, die auch in den zur verfügung gestellten Agilent-Logdateien vorkommen. Die zur Verfügung gestellten Logdateien sind auf der Projekt-CD zu finden: 2012.zip,

20120831.zip, 20120827.zip und 040107105943solmb3t11.dat.

In diesen Logdateien kommen 21 Knoten vor:

@BATCH, @BTEST, @BLOCK, @A-JUM, @LIM2, @PF, @A-MEA, @A-RES, @A-CAP, @A-DIO, @A-FUS, @A-IND, @A-ZEN, @LIM3, @M-CL, @TJET, @TS, @D-T, @BS-CON, @BS-S, @BS-O

Diese Knoten sollen beim Parsen von Agilent-Logdateien berücksichtigt werden. Werden bei Logdateien bisher unbekannte Knoten entdeckt, werden die unbekannten Knotennamen in die Logdatei UnsupportedNodes geschrieben.

2.3 Nichtfunktionale Anforderungen

Die Anwendung soll wartungsfreundlich und damit leicht änderbar und anpassbar sein, wenn z.B. neue Knotennamen hinzugefügt werden.

2.4 Entwurf

In Abbildung 2.2 auf Seite 6 ist das Entwurfsklassendiagramm vom Agilent-Parser zu sehen.

2.5 Text File Encoding

Unter Ubuntu 10.04 ist in Eclipse Juno das Text File Encoding standartmäßig auf UTF-8 gesetzt. Die kann beim Kopieren von Stings aus dem Agilent-Logformat zu Fehlern führen, da bestimmte UTF-8 Zeichen im Java-Editor unsichtbar sind. Um unsichtbare Zeichen im Java-Editor sichtbar zu machen, sollte unter Eclipse → Preference → General → Workspace das “Text File Encoding” von UTF-8 auf ISO-8859-1 umgestellt werden.

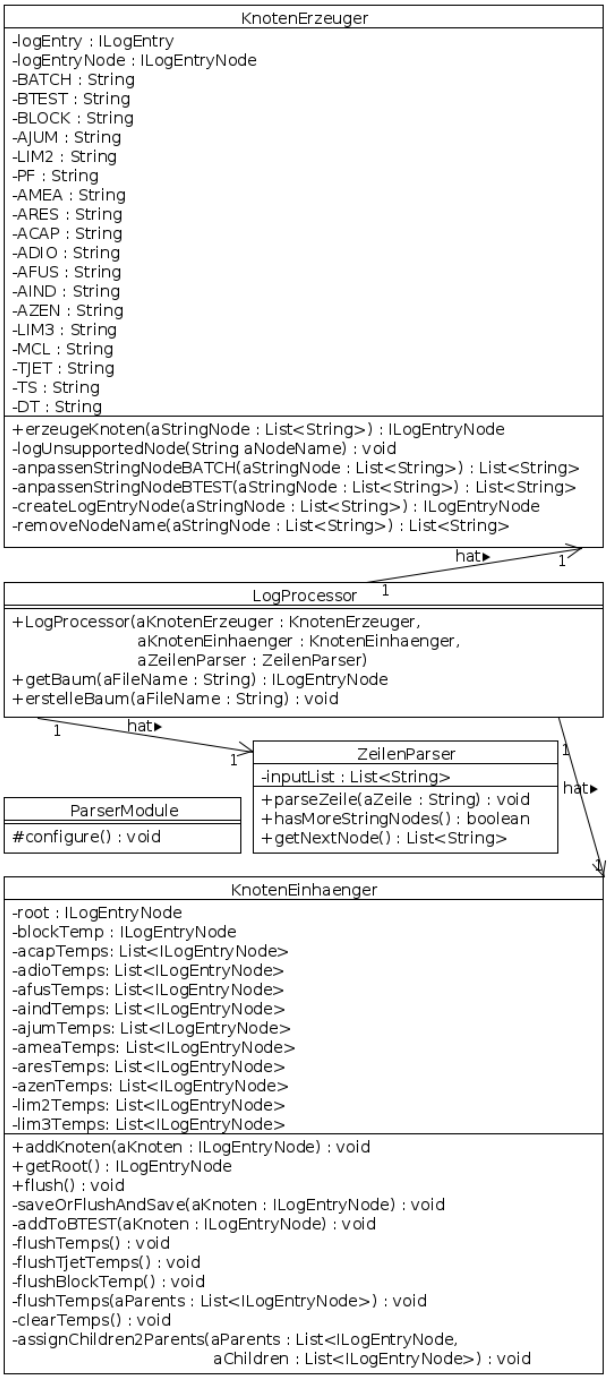

Abbildung 2.2: Entwurfsklassendiagramm Agilent-Parser

2.6 Entwicklungsumgebung

JavaSE-1.6, Maven 3.0.4, Eclipse Java EE IDE for Web Developers (Juno) Service Release 1, Ubuntu 12.04, JUnit 4.8.1, google guice 3.0, AgilentLogCompiler, git

2.7 Dependency Injection mit Google Guice

Um die Abhängigkeiten zwischen den Objekten zu minimieren, wird das Entwurfsmuster Dependency Injection verwendet. Die Abhängigkeiten werden in diesem Projekt von dem Dependency Injection Framework Google Guice bereitgestellt. Da mit Hilfe von Dependency Injection die Abhängigkeiten eines Objektes reduziert werden, ist es leichter unit-testbar.

Ohne Dependency Injection hätte der LogProcessorTest drei zusätzliche Abhängigkeiten: zu Knotenerzeuger, KnotenEinhaenger und ZeilenParser.

LogProcessor lp = new LogProcessor(new KnotenErzeuger(), new KnotenEinhaenger(), new ZeilenParser());

Mit Dependency Injection entfallen diese zusätzlichen Abhängigkeiten.

LogProcessor lp = injector.getInstance(LogProcessor.class);

2.8 Projekt importieren

Das Projekt liegt im Repository unter folgender URL:

https://github.com/rinkdotrink/agilentParser.github

Der Projektquelltext kann über git mit folgenden Befehlen heruntergeladen werden:

- git init

- git remote add origin https://github.com/rinkdotrink/agilentParser.github
- git pull origin master

Um den heruntergeladenen Quelltext in Eclipse zu importieren, kann in Eclipse File → Import → Maven → Existing Maven Projects gewählt werden.

2.9 Projekt ausführen

Das Eclipse-Projekt ist nicht als eigenständiges Programm auszufühen, sondern es ist vielmehr als Bibliothek zu verstehen. Wie es verwendet wird, geht am Besten aus den funktionalen Tests hervor.

2.10 Funktionale Tests

Die funktionalen Tests zeigen u.a. wie der Parser verwendet wird. Ein Compiler-Backend erzeugt sich ein LogProcessor-Objekt und ruft darauf die Methode getBaum() auf. Der Methode getBaum() wird der Pfad zu der zu verarbeitenden Agilent-Logdatei übergeben. Als Ergebnis liefert getBaum() die Wurzel des Baumes im gewünschten Intermediate-Format.

ILogEntryNode root = logProcessor.getBaum(“src/test/resources/Snapshot1”);

Für die funktionalen Tests wurden exemplarisch fünf Agilent-Logdateien angelegt. In den Tests werden diese fünf Agilent-Logdateien verarbeitet. Dabei wird sichergestellt, dass aus ihnen der Baum im Intermediate-Format korrekt erstellt wird.

2.11 Unstimmigkeiten im Agilent-Format

In der Format06.pdf auf Seite 35 stimmt bei BATCH die Anzahl der Attribute nicht mit der Legende überein. Auch in den tatsächlichen Agilent-Logdateien stimmt die

Anzahl der Attribute nicht immer mit der Anzahl der Attribute überein, wie sie in der Format06.pdf beschrieben werden.

In der Format06.pdf auf Seite 40 stimmt bei BTEST die Anzahl der Attribute nicht immer mit der tatsächlich vorkommenden Anzahl in der Agilent-Logdatei überein.

A Arbeitsaufteilung

Arbeit	**C. Ochmann**	**I. Körner**
Abstract		0
Einleitung		1.1
Aufgabenstellung		1.2
Forschungsgegenstand		1.3
akt. Wissensstand		1.4
Zusammenfassung		**??**
Ausblick		**??**

Tabelle A.1: Aufteilung

B Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe. Mir ist bekannt, dass jede Form des Plagiats mit der Note 5 (Betrugsversuch) bewertet wird.

Ochmann, Christof Unterschrift:

Körner, Ingo Unterschrift: