Implementierungsbericht

Wavelength–λ-IDE

Muhammet Guemues, Markus Himmel, Marc Huisinga,
Philip Klemens, Julia Schmid, Jean-Pierre von der Heydt

5. Februar 2018

Inhaltsverzeichnis

1 Einleitung

Nach einer groben Skizze der Anwendung im Pflichtenheft und einer genaueren Spezifikation in der Entwurfsphase folgt nun eine Vorstellung der konkreten Implementierung. Die Erfüllung der geforderten Spezifikationen erfordert den Einsatz verschiedener Bibliotheken, die kurz vorgestellt werden sollen:

1.1 Google Web Toolkit (GWT)

Ziel: Programmierung der Anwendung in Java

Beschreibung: Das Google Web Toolkit kompiliert Java-Quellcode zu JavaScript, das dann auf einem Web-Server ausgeführt werden kann.

Erläuterung: Ziel dieses Projektes ist die Entwicklung einer Online-Entwicklungsumgebung für den untypisierten λ-Kalkül. Diese unterliegt der Einschränkung, in einer statisch typisierten Programmiersprache verfasst zu werden(siehe Pflichtenheft unter N4). Um dieser Anforderung nachzukommen, wird die Anwendung in Java programmiert und mit GWT auf einem Web-Server ausgeführt.

1.2 GWT-Bootstrap

Ziel: optisch ansprechende Benutzeroberfläche

Beschreibung: GWT-Bootstrap stellt Java-Klassen für die Verwendung von Bootstrap-Komponenten mit GWT bereit.

Erläuterung: Für die Entwickler waren die von GWT bereitgestellten GUI-Elemente optisch inkohärent und von niederer Anmut. Ein stimmigeres Bild fanden Sie in den von Bootstrap bereitgestellten Komponenten. Die Verwendung dieser Komponenten mit GWT erwies sich jedoch als umständlich. Um diese Komponenten trotzdem verwenden zu können, wurde GWT-Bootstrap eingebunden, das die Verwendung erheblich erleichtert. Darüber hinaus stellt die Bibliothek auch Icons zur Verfügung, die für die GUI verwendet werden(siehe dazu bspw Pflichtenheft Anhang A). Der Vorteil dieser Icons ist, dass sie einen Satz bilden, also ein konsistentes Bild bieten.

1.3 Monaco

Ziel: Bereitstellung eines Editors

Beschreibung: Monaco ist ein Code-Editor, der in der Software Visual Studio Code verwendet wird.

Erläuterung: Die Entwicklung eines Editors steht bei diesem Projekt nicht im Fokus. Daher bietet es sich an, hier auf eine Bibliothek zurückzugreifen. Darüber hinaus wird der Monaco-Editor nicht nur den gestellten Anforderungen gerecht, sondern bietet darüber hinaus viele weitere Features.

1.4 vis.js

Ziel: Erstellung interaktiver Graphen

Beschreibung: Bei vis.js handelt es sich um eine Bibliothek für Visualisierungen aller Art mittels JavaScript.

Erläuterung: Die Ausgabe von λ-Termen in Form eines Syntaxbaums gestaltet sich als nicht-triviale Aufgabe. Dieser muss nicht nur konstruiert werden, sondern auch noch stellenweise klickbar sein(siehe dazu Pflichtenheft unter F5). Die Modellierung der Daten erfolgt als Java-Quellcode; lediglich die Visualisierung baut auf der JavaScript-Bibliothek auf.

1.5 SQLite

Ziel: Bereitstellung einer Datenbank

Beschreibung: SQLite stellt eine relationale Datenbank zur Verfügung.

Erläuterung: Nutzer sollen in der Lage sein, den aktuellen Zustand der Anwendung mit anderen zu teilen. Dazu wird der Zustand in eine Zeichenkette übersetzt. Diese kann aber eine Länge erreichen, mit der weder moderne Browser noch URL-Kürzer arbeiten können. Daher werden die generierten Zeichenketten in einer Datenbank gespeichert. Als Schlüssel wird dabei eine zufällig erzeugte UUID (Universally unique identifier) verwendet, die anschließend geteilt werden kann.

1.6 JUnit

Ziel: Testen einzelner Komponenten

Beschreibung: JUnit ist ein Test-Framework für Java.

2 Änderungen am Entwurf

2.1 View

Die Pakete *edu.kit.wavelength.client.view.webui.component* und *edu.kit.wavelength.client.view.api* wurden entfernt:
Die Verwendung der Bootstrap-Komponenten anstelle der ursprünglich geplanten GWT-Komponenten hätte eine Anpassung der Wrapper-Klassen im *webui.component-Paket* zur Folge. Diese wurden aber teilweise durch die Komponenten obsolet.
So war bspw die ImageButton Klasse für die Knöpfe mit Icons gedacht. Bootstrap stellt aber über Font Awesome selbst Icons bereit, die den Knöpfen direkt zugewiesen werden können. Damit hat die ImageButton Klasse keine Funktionalität mehr, die sie von TextButton unterscheidet und kann gestrichen werden.
So ähnlich verhält es sich mit vielen anderen Wrappern.
Darüber hinaus stehen die im *api-Paket* definierten Interfaces bereits (unter anderem Namen) zur Verfügung. So entsprechen die Interfaces Readable und Writable gerade GWT's hasText-Interface.
Außerdem kann eine Menge an Quellcode zur Erstellung der Wrapper in der App-Klasse eingespart werden.

Die Serialisierung ignoriert den Übungsmodus:
In Ermangelung von Zeit wurde der Einfachheit halber der Übungsmodus bei der Serialisierung der Applikation ignoriert, d.h. weder die aktuell ausgewählte Übungsaufgabe noch die Fenster mit der Aufgabenstellung und der Lösung werden beim Deserialisieren wiederhergestellt.

Neue Pakete: edu.kit.wavelength.client.database und edu.kit.wavelength.server.database
Die *database-Pakete* stellen Schnittstellen für den Zugriff auf die SQLite-Datenbank bzw für deren Manipulation bereit ebenso wie eine konkrete Implementierung. Auf der Client-Seite stehen die Interfaces *DatabaseService* und *DatabaseServiceAsync*.

DatabaseService definiert die Schnittstelle für die Datenbank:
Erstens eine Methode, die zu einer gegebenen Identifikationsnummer(als String) einen

Serialisierungsstring zurückgibt, falls ein solcher Eintrag in der Datenbank existiert. Zweitens eine Methode die für einen gegebenen Serialisierungsstring einen neuen Eintrag in der Datenbank erstellt und den entsprechenden Identifikationsstring zurückgibt.

DatabaseServiceAsync repräsentiert die Schnittstelle für asynchrone Zugriffe auf die Datenbank. Die Funktionsweise kann unter http://www.gwtproject.org/doc/latest/tutorial/RPC.html nachvollzogen werden.

Auf der Server-Seite steht *DatabaseServiceImpl*.
DatabaseServiceImpl realisiert die Schnittstelle zur persistenten SQLite-Datenbank auf dem Server.

3 Implementierte Funktionalitäten

Hier gibt es einen kurzen Überblick über die im Pflichtenheft definierten Muss- und Kann-Kriterien und deren Umsetzung:

Nummer	Beschreibung	implementiert?
M1	Eingabe von λ-Termen	✓
M2	Auswertung von λ-Termen	✓
M3	Fehlermeldung bei invalider Eingabe	✓
M4	Abbruch der Reduktion	✓
K1	Weitere Auswertungsstrategien	✓
K2	Exportformate	✓
K3	Ausgabeformate	✓
K4	Erweiterte Fehlerdiagnostik	(✓)
K5	Intelligenter Editor	✓
K6	Standardbibliothek	
K7	Übungsaufgaben	
K8	Ausgabe von Teilschritten	✓
K9	Schritt-für-Schritt Modus	✓
K10	Permalinks	✓
K11	komfortabler λ-Code	✓
Gesamt	**15**	**12.5**

4 Implementierungsplan

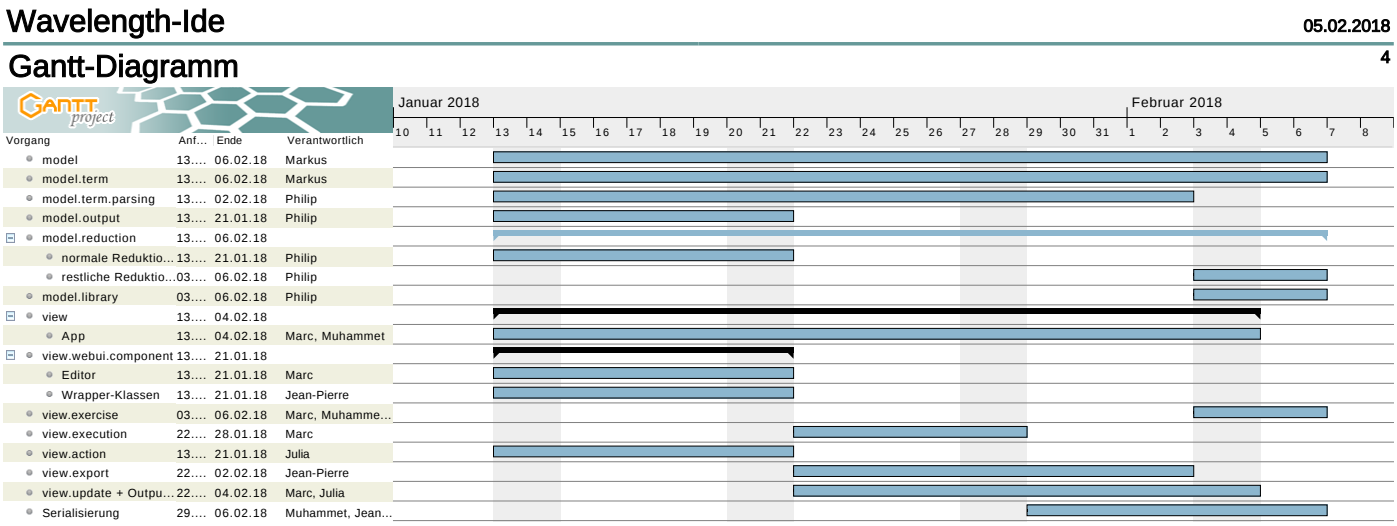

Abbildung 1: Der Implementierungsplan wie er zu Beginn der Implementierungsphase vereinbart wurde

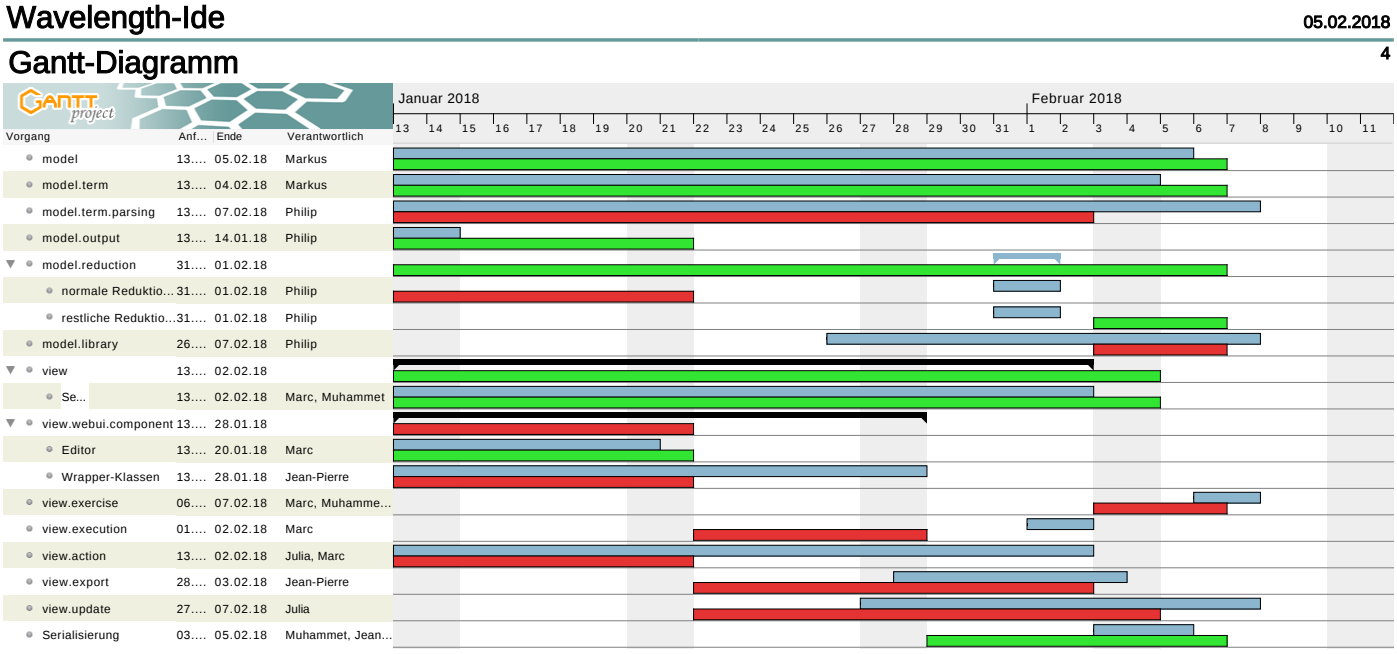

Abbildung 2: Der reale Verlauf der Implementierungsphase: Die oberen Balken entsprechen der Realität, die unteren der Planung.

5 Komponententests

5.1 Paket edu.kit.wavelength.client.model.serialization

Dieses Paket testet die Klasse *SerializationUtilities*. Es wird getestet, ob die Methode *List<String> extract(String input)* eine gegebene Serialisierung korrekt entpackt. Weiter wird getestet, ob die Methode *StringBuilder enclose(StringBuilder... content)* aus gegebenen StringBuildern, die eine Serialisierung erhalten, einen neuen StringBuilder erzeugt, der eine gültige Gesamt-Serialisierung enthält.

5.2 Paket edu.kit.wavelength.client.model.term

In diesem Paket wird der BetaReducer ausführlich getestet: Index-Anpassung, Aliases, Fixpunkte, Wachstum, Entfernung von Namen, Regexe und PartialApplications. Weiterhin wird der ResolvedNamesVisitor auf Gleichheit von Abstraktionen, NamedTerms und Library Terms getestet. Ebenso wird die Serialisierung von LambdaTerm-Objekten getestet

5.3 Paket edu.kit.wavelength.client.model.term.parsing

In diesem Paket wird der Parser getestet. Getestet wird das Verhalten bei leerer Eingabe, invalider Eingabe und valider Eingabe. Weiter wird die Namensbindung getestet.

5.4 Paket edu.kit.wavelength.client.view.export

In diesem Paket werden die ExportFormate und deren Visitors getestet. Für die ExportFormate wird jeweils der Umgang mit null-Termen, null-Libraries, leeren Eingaben, einelementigen Eingaben und ganzen Eingabelisten getestet. Für die Visitor wird jeweils der Umgang mit simplen und komplexen sowie null-Eingaben getestet ebenso wie PartialApplications, Abstraktionen, Applikationen, Named Terms.