Masterarbeit

Automatisierte Transformation von Flow-basiertem JavaScript-Code nach TypeScript

Zur Erlangung des akademisches Grades eines
Master of Science

angefertigt von Jonathan Gruber (68341)

Fakultät Informatik, Mathematik und Naturwissenschaften
Studiengang: Master Informatik (16INM/VZ)

Erstprüfer	Prof. Dr. rer. nat. habil. Frank
Zweitprüfer	*TBA*

ausgegeben am TBA 2019
abgegeben am TBA 2019

Inhaltsverzeichnis

1 Motivation

2 Grundlagen

TODO

3 Analyse

TODO

4 Prototyp

TODO

5 Ergebnisse

TODO

Literatur

[1] Sven Casteleyn, Irene Garrig'os und Jose-Norberto Maz'on. „Ten years of rich internet applications: A systematic mapping study, and beyond". In: *ACM Transactions on the Web (TWEB)* 8.3 (2014), S. 18 (siehe S. VII).

[4] Ecma International. *ECMAScript 2015 Language Specification*. 6th. Geneva, Juni 2015. URL: http://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf (siehe S. VIII).

[13] Charles Severance. „JavaScript: Designing a Language in 10 Days". In: *Computer* 45 (Feb. 2012), S. 7–8. ISSN: 0018-9162. DOI: 10.1109/MC.2012.57. URL: doi.ieeecomputersociety.org/10.1109/MC.2012.57 (siehe S. VII).

[14] Antero Taivalsaari und Tommi Mikkonen. „The web as a software platform: ten years later". In: *Proceedings of the 13th International Conference of Web Information Systems and Technologies. INSTICC, SciTePress*. 2017 (siehe S. VII).

Online-Quellen

[2] Steve Champeon. *JavaScript: How Did We Get Here?* Juni 2001. URL: https://web.archive.org/web/20160719020828/http://archive.oreilly.com/pub/a/javascript/2001/04/06/js_history.html (besucht am 25. 11. 2018) (siehe S. VIII).

[3] Microsoft Corporation. *TypeScript - JavaScript that scales*. 2018. URL: https://www.typescriptlang.org/ (besucht am 22. 11. 2018) (siehe S. IX).

[5] Jesse James Garrett. *Ajax: A New Approach to Web Applications*. Feb. 2005. URL: http://adaptivepath.org/ideas/ajax-new-approach-web-applications/ (besucht am 21. 11. 2018) (siehe S. VIII).

[6] Inc. GitHub. *Electron - Build cross platform desktop apps with JavaScript, HTML, and CSS*. 2018. URL: https://electronjs.org/ (besucht am 22. 11. 2018) (siehe S. VII).

[7] Stack Exchange Inc. *Stack Overflow Annual Developer Survey*. 2018. URL: https://insights.stackoverflow.com/survey/2018 (besucht am 21. 11. 2018) (siehe S. VII, XI).

[8] Facebook Inc. *Flow: A Static Type Checker For JavaScript*. 2018. URL: https://flow.org/ (besucht am 22. 11. 2018) (siehe S. IX).

[9] Facebook Inc. *React - A JavaScript library for building user interfaces*. 2018. URL: https://reactjs.org/ (besucht am 22. 11. 2018) (siehe S. VII).

[10] Google LLC. *Angular*. 2018. URL: https://angular.io/ (besucht am 22. 11. 2018) (siehe S. VII, XI).

[11] Sebastian McKenzie und other contributors. *Babel - The compiler for next generation JavaScript*. 2018. URL: https://babeljs.io/ (besucht am 22. 11. 2018) (siehe S. IX).

[12] Tim O'Reilly. *What Is Web 2.0*. Sep. 2005. URL: https://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html (besucht am 21. 11. 2018) (siehe S. VIII).

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcode-Listings

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst habe. Ich versichere, dass ich keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe, und dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahren gewesen ist.

Leipzig, den 21. Januar 2019

_____

Jonathan Gruber

A TBA

A.1 Expose

A.1.1 Die historische Entwicklung JavaScripts

JavaScript hat als Lingua Franca des World Wide Webs und primäre clientseitige Programmiersprache für Webanwendungen aller Art innerhalb der letzten Jahre enorm an Bedeutung gewonnen. Dies belegt beispielsweise die alljährliche Umfrage „*Stack Overflow Developer Survey*“ der Programmierer-Plattform *Stack Overflow*, welche die Ergebnisse der Befragung von über 100.000 Software-Entwicklern weltweit auswertet [7]. Bereits das sechste Jahr in Folge führt JavaScript die Rangliste der populärsten Programmiersprachen an. Dies ist nicht verwunderlich hinsichtlich des ungebrochenen Trends, dass immer mehr Software als Webanwendung konzipiert wird [14][1]. Viele beliebte Applikationen wie *Spotify*, *Slack* oder *Visual Studio Code* basieren z. B. auf dem Framework *Electron* [6]. Dabei wird die gesamte grafische Oberfläche der Anwendung durch HTML und CSS innerhalb eines Chromium-Webbrowsers realisiert. Derzeitige JavaScript-Frameworks und -Bibliotheken wie Angular [10] oder React [9] verdeutlichen den hohen Bedarf an modernen Entwicklungsansätzen und anspruchsvollen Programmierparadigmen, die es ermöglichen, umfangreiche, skalierbare Anwendungen in JavaScript umzusetzen.

Als JavaScript jedoch 1995 als Bestandteils des Browsers *Netscape Communicator* erfunden worden ist[1] [13], war nicht abzusehen, welche große Bedeutung die Sprache über 20 Jahre später inne haben wird. Ursprünglich war die Skriptsprache lediglich als ergänzendes

[1] Zunächst „LiveScript“ genannt.

Werkzeug gedacht, um den Zugriff auf das *Document Object Model* (DOM) von Websites zu ermöglichen und diese dynamischer zu gestalten. Die zu Anfang sehr inkonsistente Implementierung der Sprache in den verschiedenen Webbrowsern und die schwache Typisierung JavaScripts war (und ist) Grund für große Frustration und anfängliche Ablehnung der Sprache seitens professioneller Software-Entwickler [2]. Mit Aufkommen des Web 2.0 [12] und des damit einhergehenden sukzessiven Bedeutungszuwachses von JavaScript[2] begann die Sprache jedoch allmählich zu reifen. Die sechste Version der als *ECMA Script* bekannten Sprachspezifikation „ECMA Script 2015“[3] [4] versuchte viele der historisch bedingten Schwachstellen auszumerzen und führte eine Vielzahl von syntaktischen Verbesserungen sowie neuen Funktionen ein.

Hierdurch und durch die stetige Weiterentwicklung von ECMA Script wurde der Grundstein für zukunftssichere JavaScript-Anwendungen gelegt. Dennoch stellt insbesondere das dynamische, schwache Typsystems der Programmiersprache ein Hindernis für die Entwicklung umfangreicher Software dar. Unbeabsichtigte, implizite Typumwandlungen zur Laufzeit und falsche Annahmen über vorliegende Datenstrukturen sind häufige Fehlerursachen. Eine starke, statische Typisierung, wie sie beispielsweise in C++ oder Haskell vorliegt ist erstrebenswert, denn sie bietet viele Vorteile für den Software-Entwicklungsprozess: Logik- und Flüchtigkeitsfehler im Quelltext können oftmals bereits vor Ausführung des Programms erkannt und behoben werden. Die Entwickler gewinnen Sicherheit und Zuversicht, dass Änderungen in umfangreichen Projekten keine unerwünschte Nebenwirkung verursachen, wodurch sich die Wartbarkeit der Software erhöht. Eine explizite Typisierung führt darüber hinaus dazu, dass der Programmierer gezwungen ist seine *Intension* klar zu formulieren. Hierdurch verbessert sich die Ausdruckskraft des Codes. Weiterhin wird der Quelltext an Ort und Stelle durch eine vernünftige Typisierung bereits grundlegend dokumentiert („*inline documentation*“).

[2] Beispielsweise als Bestandteil von *AJAX* (Asynchronous JavaScript and XML) [5].
[3] Oft auch als „ES6“ bezeichnet.

A.1.2 Statische Typsysteme in JavaScript

Derzeit existieren zwei Ansätze, um statische Typsicherheit für JavaScript-Code zu erzielen:

1. **Flow** [8] ist ein von Facebook entwickeltes Werkzeug, um den Quelltext mittels Typannotationen hinsichtlich der Korrektheit der so umgesetzten Typisierung zu überprüfen. Die Annotationen und eigenen Typdefinitionen werden in den Quelltext eingefügt und müssen vor dessen Auslieferung durch einen Transpilierungsprozess entfernt werden, sodass wieder standardkonformer JavaScript-Code entsteht. Dieser Schritt kann beispielsweise durch *Babel* [11] realisiert werden.
2. **TypeScript** [3] ist eine von Microsoft entwickelte, vollständige Programmiersprache, welche eine Obermenge von JavaScript darstellt. Ein stark ausgeprägtes Typsystem ist elementarer Bestandteil der Sprache. Im Gegensatz zu Flow wird TypeScript-Code durch einen Compiler in JavaScript übersetzt, welcher auch auch die statische Typprüfung vollzieht. Diese kann natürlich auch simultan zur Arbeit des Programmierers innerhalb der integrierten Entwicklungsumgebung ablaufen.

```
// Beispiel in Flow:
// @flow
type Vector = {
  +x: number,                          // + bedeutet "schreibgeschützt"
  +y: number,
}

function vectorLength<T: Vector>(vec: Vector): number {
  return Math.sqrt(Math.pow(vec.x, 2) * Math.pow(vec.y, 2));
}

const vec: Vector = { x: 2, y: 5 };
vec.x = 3;                             // Fehler: Attribut x ist schreibgeschützt

vectorLength({ x: 2, y: 5 });          // In Ordnung
vectorLength({ x: 2, y: 'string' });   // Fehler: Attribut y ist nicht vom Typ `number`
vectorLength({ y: 5 });                // Fehler: Attribut x fehlt
vectorLength({ x: 2, z: 5 });          // Fehler: Attribut z ist nicht Teil des Typs
```

```
// ---------------------------------------------------------------------------------------

// Analoges Beispiel in TypeScript:
type Vector = {
  readonly x: number,
  readonly y: number,
}

function vectorLength<T extends Vector>(vec: Vector): number {
  return Math.sqrt(Math.pow(vec.x, 2) * Math.pow(vec.y, 2));
}

const vec: Vector = { x: 2, y: 5 };
vec.x = 3;                              // Fehler: Attribut x ist schreibgeschützt

vectorLength({ x: 2, y: 5 });           // Korrekt
// Rest analog...
```

Quellcode 1: Vergleich der zwei Ansätze für statische Typisierung von JavaScript mit Flow (oben) und TypeScript (unten).

Wie Quelltext 1 verdeutlicht, haben Flow und TypeScript eine ähnliche, wenn auch in manchen Fällen nicht völlig identische Syntax:

- Vgl. „`+x: number`“ vs. „`readonly x: number`“
- Vgl. „`<T: Vector>`“ vs. „`<T extends Vector>`“

Innerhalb des eCommerce-Unternehmen **TeamShirts**[4] gibt es strategische Überlegungen, die bestehenden Frontend-Projekte, die im Moment allesamt auf React und Flow basieren, nach TypeScript zu migrieren. Grund hierfür ist die Annahme, dass TypeScript langfristig zukunftsträchtiger und besser geeignet als Flow sein könnte. Es gibt einige Argumente, die für TypeScript sprechen:

- **Performance** — Während derzeit noch keine verlässlichen Zahlen vorliegen, um diese These stichhaltig zu untermauern, wird TypeScript von vielen Entwicklern im Vergleich zu Flow als schneller hinsichtlich der Typprüfung beschrieben.

[4] Ein Tochterunternehmen der Spreadshirt AG. Kunden haben die Möglichkeit Textilien und andere geeignete Gegenstände wie Taschen, Mützen, Tassen etc. mit eigenen und vorgefertigten Bildmotiven zu bedrucken. Hierfür können die Produkte in einer Webanwendung selbstständig gestaltet und im Anschluss bestellt werden.

- **Vorgefertigte Typisierungen** — Jede Software basiert im Normalfall auf externen Bibliotheken. Sowohl für TypeScript als auch für Flow gibt es von der Gemeinschaft verwaltete Projekte[5], welche die Typisierung dieser Abhängigkeiten bereit stellen. Hierdurch wird beispielsweise die Autovervollständigung in integrierten Entwicklungsumgebungen erheblich verbessert. Jedoch bietet TypeScript dabei eine deutlich vollständigere Unterstützung als Flow[6].
- **Kontinuität / Stabilität** – Es besteht Unsicherheit darüber, ob Facebook und Microsoft auf lange Sicht daran interessiert bleiben werden Flow bzw. TypeScript weiterzuentwickeln und zu warten. Da TypeScript als eigenständige Programmiersprache in vielen Bereichen eingesetzt wird und stetig an Popularität gewinnt [7], erscheint es jedoch unwahrscheinlich, dass Microsoft dieses Produkt in naher Zukunft einstellen wird. Angular [10] ist beispielsweise ein großes, erfolgreiches Framework, welches in TypeScript geschrieben ist und die Verwendung der Sprache fördert.
- **Community / Dokumentation** – Auch die Größe und Dynamik der „*Community*“ ist ein relevanter Aspekt, da eine große Nutzerzahl bei Open-Source-Projekten einen hohen Grad an Aktivität bedeutet. Offene Fragen und Probleme können durch Online-Recherche meistens schnell beantwortet werden, da das Problem oft bereits behandelt wurde. Ein einfacher, aber wichtiger Aspekt ist darüber hinaus die Aktualität und Qualität der Dokumentation.

Ziel der angestrebten Masterarbeit ist damit zunächst eine Analyse hinsichtlich der Vor- und Nachteile sowie der Machbarkeit einer Migration von Flow nach TypeScript. Der eigentliche Kern der Arbeit wäre anschließend die prototypische Entwicklung eines Werkzeugs, welches die bestehende Codebasis[7] von TeamShirts automatisch in äquivalenten TypeScript-Programmtext transformiert. Dabei muss die Typisierung aufgrund der unterschiedlichen Syntax von Flow und TypeScript z. T. angepasst werden. Konzeptionell würde dies durch Einlesen (*Parsing*) des Quelltexts, anschließender Manipulation des zugehörigen abstrakten Syntaxbaums und schließlich der Ausgabe als TypeScript-Code realisiert werden. Ein funktionsfähiger Prototyp könnte darüber hinaus auch für weitere Unternehmen, die vor derselben Problematik stehen, von Vorteil sein.

[5] Flow Typed für Flow und Definetely Typed für Typescript.
[6] Vgl. GitHub-Projekte in Fußnote 5.
[7] Circa 150.000 Zeilen JavaScript-Code.