



Hochschule Bremen

Exposé

Media-science Bsc.

Roman Quistler

April 29, 2019

Contents

1	Introduction	1
2	Problem and solution	1
2.1	Problem	1
2.2	Solution	3
3	Specific Tasks	4
4	Early Structure	5
5	Timetable	7

1 Introduction

Das Entwickeln von Applikationen ist nicht einfach.

2 Problem and solution

2.1 Problem

Today's applications seem to be increasing in complexity over the last few years. Especially in the field of frontend development, the range of functions has increased [1]. The web development has transitioned from server rendered, page-reloading websites, to modern so called single page applications or SPA's. But the same applies to the mobile world: Social networks, navigation, sharing and editing files together is a common desire by the user. Ein großer Teil der Applikationen steht somit auch im Austausch mit APIs, greift auf lokale und externe Datenbanken zu oder interagiert mit dem zugrundeliegenden System selbst (bswp. die periodische Erfassung des Standortes via GPS oder WiFi). Zusätzlich zu der Interaktion mit dem Nutzer und dessen Einfluss, muss die Anwendungen auf oben genannte Einflüsse reagieren können und vorrausgegangene Ereignisse und deren Auswirkung speichern. Es muss bekannt sein, welche Zustände bei welchen Ereignissen eintreten können und welche nicht. Dies ist nötig, um eine zuverlässige Funktionsweise einer Applikationen zu garantieren. Folgendes Beispiel soll den Sachverhalt verdeutlichen: Der Nutzer befindet sich auf einem "Anmelden" Fenster. In diesem kann der Nutzer eine E-Mail-Adresse und ein Passwort eingeben. Darunter befindet sich ein herkömmlicher "Anmelden" Knopf. Des weiteren existiert die Möglichkeit auf "Passwort vergessen?" zu klicken. Die folgenden Zustände können hierbei eintreten:

- Die E-Mail-Adresse ist nicht zulässig und das Passwort Feld leer
 - Der "Anmelden" Knopf ist deaktiviert
- Die E-Mail-Adresse ist zulässig und das Passwort Feld leer
 - Der "Anmelden" Knopf ist deaktiviert
- Die E-Mail-Adresse ist zulässig und das Passwort Feld nicht leer
 - Der "Anmelden" Knopf ist aktiviert
- Der Nutzer betätigt den "Anmelden" Knopf
 - Der "Anmelden" Knopf ist aktiviert
- Die Email ist inkorrekt oder existiert nicht
- Die Email ist korrekt und das Passwort falsch
- Die Email ist korrekt und das Passwort richtig
 - Der Nutzer gelangt zum nächsten Fenster

- Eine dreimalige falsche Eingabe des Passworts führt zu einer temporären Sperrung
- Der Nutzer klickt auf "Passwort vergessen"
- Es gibt einen Netzwerkfehler
 - Es wird eine Fehlermeldung angezeigt
- Es besteht keine Verbindung zum Internet
 - Es wird eine Meldung angezeigt
- Der Nutzer dreht das Smartphone
 - Der Zustand muss gespeichert werden
- Die Applikationen wird vom System beendet
 - Der Zustand muss gespeichert werden

Anhand dessen wird sichtbar, wie viele Ereignisse und Zustände bereits aus einer simplen Funktion wie "Anmelden" hervorgehen. Die Schwierigkeit die hierbei unter anderem entsteht, ist, die Fülle an möglichen Zuständen zu konkretisieren und formal als auch inhaltlich "ansprechend" darzustellen/zu beschreiben. Wie sollte die Struktur eines Programmes aufgebaut/organisiert werden, wenn man Zustandsorientiert arbeitet? Wirft der Entwickler einen Blick auf die Implementierung einer Funktion (wie das obige Beispiel des Vorgangs des Anmeldens), ist oft nicht gleich ersichtlich, welche Zustände insgesamt existieren. Dies kann mitunter daran liegen, dass Attribute die einen Zustand bestimmen, sich an unterschiedlichen Orten innerhalb einer Implementation befinden. (In Android bspw. im Fragment und im Presenter zugleich) Es ist sogar möglich, dass ein Zustand von anderen Eigenschaften (is der Knopf aktiviert?) bei jeder Anfrage abgeleitet wird, anstatt diesen zu hinterlegen. Überhaupt ist es oft kompliziert, den Ablauf eines Programmes ohne großen Aufwand nachzuvollziehen. Dazu gesellt sich zwangsläufig die Frage, auf welche Art und Weise ein Zustand überhaupt definiert wird. Wird er dynamisch, anhand eines hinterlegten Regelwerks gebildet, oder ist er von Beginn an festgelegt? Nicht weniger unherheblich ist, wie der Übergang von einem Zustand in einen anderen Zustand erfolgt und was genau der Initiator eines Übergangs ist. Kommt dafür ausschließlich der Nutzer in Frage, oder auch das zugrundeliegende System? Und wird dabei semantisch unterschieden? Eine weitere, nicht zu unterschätzende Problematik ist die asynchrone Programmierung bzw. "Multi Threading", welches in heutigen Applikationen immer öfters Anwendung findet. Soll in einer Android Applikationen bspw. auf das Internet zugegriffen werden, ist der Einsatz eines zusätzlichen "Thread" unabdingbar. In der Kombination mit veränderlichen Attributen und "Global State" kann dies zu sogenannten Seiten Effekten führen und trägt dazu bei, dass der Quellcode unleserlich, schwer vorherzusehen und schwer wartbar wird.

2.2 Solution

Um den genannten Problem entgegenzuwirken gilt es, ein Muster zu entwerfen, in welchem der "Zustand" das Hauptaugenmerk bildet. Eine gute Methode um mit Zuständen zu arbeiten, bietet das Konzept der endlichen Automaten. Hierbei wird ein Startzustand festgelegt, sowie eine endliche Anzahl von Zuständen, die im System mit Zustandsübergängen erreicht werden können. Zu beachten ist, dass sich das System immer nur in genau einem Zustand befinden kann und dieser den SSOT (single source of truth) darstellt. Die wohl wichtigste Eigenschaft eines endlichen Automaten ist jedoch, dass bei gleicher Eingabe immer die selbe Ausgabe erfolgt. Kurz um: Der Automat befähigt einen, alle Zustände in dem sich ein System befinden kann konsistent zu beschreiben. Unter der Vielzahl an endlichen Automaten fällt die Wahl auf den sogenannten Mealy-Automat. Bei diesem wird der nächste Zustand auf Basis des derzeitigen Zustands und der erfolgten Eingabe ermittelt. Die Eingabe spiegelt dabei ein Ereignis wieder, welches durch den Nutzer selbst (und seinen Interaktionen) oder durch Komponenten der Peripherie ausgelöst wurde. Zu nennen wäre bspw. der Klick auf einen Knopf oder die Mitteilung des Betriebssystems, dass der Standort sich verändert hat. Die Beschreibung eines solchen endlichen Automaten lässt sich - unter anderem - mit mathematischen Funktionen umsetzen. Für dies und der o.g. Prämisse, bei der die Ausgabe (hier: der Zustand) bei wiederholter, gleicher Eingabe nicht abweichen darf, eignet sich ein Konzept welches auf Namen "Pure Functions" hört. Dieses entspringt dem Programmierparadigma der funktionalen Programmierung und aus eben jener sollen mehrere Konzepte übernommen werden: Unveränderliche Daten Strukturen.

Keine Seiteneffekte.

Kein globalen Variablen.

Pattern Matching + Expressions.

Unidirektional.

Um der hohen Interaktivität einer Applikation Herr zu werden, wird ein weiteres Programmierparadigma und Abstraktion herangezogen: Reaktive Programmierung. Sie baut auf der funktionalen Programmierung auf und berücksichtigt die oben genannten Eigenschaften. Des Weiteren erleichtert sie das Arbeiten und Koordinieren von "Threads". Um das Arbeiten mit einem endlichen Automaten zu vereinfachen und den Quellcode in seiner Lesbarkeit zu stärken, wird eine DSL (Domain Specific Language) in Erwägung gezogen.

3 Specific Tasks

Für die Umsetzung der Bachelor Thesis sind folgende Aufgaben vonnöten:

- Recherche: Erhebung und Vergleich von weiterer Literatur, für einen tieferen Einstieg in das Thema.
- Funktionale Anforderungen:
 - Das System soll die Arbeit mit Zuständen auf Basis eines endlichen Automaten vereinfachen.
 - Berechenbar.
 - Reaktiv.
 - Unveränderlich.
 - Unidirektional.
- Nicht-funktionale Anforderungen:
 - Die Applikation wird in Kotlin entwickelt.
 - Es werden RxJava, RxKotlin und RxAndroid verwendet.
- prototypische Implementierung: Entwicklung einer nativen Android Applikation, in der die ermittelten Anforderungen
- Evaluation: Bewertung und Diskussion der Ergebnisse

4 Early Structure

Zusammenfassung/Abstract

Eigenständigkeitserklärung/Eidesstattliche Erklärung

1. Einleitung
 - 1.1. Problemfeld/Motivation
 - 1.2. Ziel(e) der Arbeit
 - 1.3. Aufgabenstellung
 - 1.4. Lösungsansatz
2. Anforderungsanalyse
 - 2.1. Funktionale Anforderungen
 - 2.2. Nicht-funktionale Anforderungen
 - 2.3. Übersicht der Anforderungen/Zusammenfassung
3. Grundlagen und verwandte Arbeiten
 - 3.1. Endliche Automaten
 - 3.2. Bidirektionalität
 - 3.3. Funktionale Programmierung
 - 3.3.1. Pure Functions
 - 3.3.2. Expressions (vs Statements)
 - 3.3.3. Function Composition
 - 3.3.4. Monads
 - 3.3.5. Immutability
 - 3.4. Reaktive Programmierung
 - 3.4.1. Observer Pattern
 - 3.4.2. Iterator Pattern
 - 3.4.3. Streams
 - 3.5. Vergleich mit bestehenden Design Patterns
 - 3.5.1. MVVM (Model-View-View Model)
 - 3.5.2. MVI (Model-View-Intent)
 - 3.5.3. Zusammenfassung
 - 3.6. Untersuchung von Bibliotheken
 - 3.6.1. Workflow
 - 3.6.2. Mosby-MVI

- 3.6.3. Zusammenfassung
- 3.7. Verwandte Arbeiten
- 4. Konzeption
 - 4.1. Entwurf eines Design Patterns/ eines Workflow
 - 4.2. Entwurf der Komponenten
 - 4.2.1. Endlicher Automat
 - 4.2.2. Events/Intents
 - 4.2.3. State/Store
 - 4.3. Entwurf einer DSL
 - 4.4. Zusammenfassung
- 5. Prototypische Realisierung/Implementierung
 - 5.1. Wahl der Realisierungsplattform
 - 5.2. Ausgewählte Realisierungsaspekte
 - 5.3. Qualitätssicherung
 - 5.4. Zusammenfassung
- 6. Validierung und Verifikation
 - 6.1. Unit-Tets
 - 6.2. Integrationstest
- 7. Evaluation
 - 7.1. Überprüfung funktionaler Anforderungen
 - 7.2. Überprüfung nicht-funktionaler Anforderungen
 - 7.3. Zusammenfassung
- 8. Zusammenfassung und Ausblick
 - 8.1. Zusammenfassung
 - 8.2. Ausblick

5 Timetable

Geplanter Starttermin: 1. Juni 2019

Bearbeitungsdauer: 9 Wochen

Tabelle 1 stellt die geplanten Arbeitspakete und Meilensteine dar:

M1	Offizieller Beginn der Arbeit	01.06.2019
	<ul style="list-style-type: none">• Einrichtung der Textverarbeitung• Anforderungsanalyse• Verfassen der Thesis: Kapitel 2 (Anforderungsanalyse)	1.5 Wochen
M2	Abschluss der Analysephase	11.06.2019
	<ul style="list-style-type: none">• Recherche• Verfassen der Thesis: Kapitel 3 (Grundlagen	1.5 Wochen
M3	Abschluss der Analysephase	11.06.2019
	<ul style="list-style-type: none">• Recherche• Verfassen der Thesis: Kapitel 3 (Grundlagen	1.5 Wochen
My	Erste Fassung vollständige Thesis	11.06.2019
	<ul style="list-style-type: none">• Korrekturlesen• Drucken / binden lassen	1 Woche
My	Abgabe der Thesis	11.06.2019

List of Figures

Online References

- [1] Kevin Ball. *The increasing nature of frontend complexity*. Jan. 30, 2018. url: <https://blog.logrocket.com/the-increasing-nature-of-frontend-complexity-b73c784c09ae> (visited on 02/04/2019).
- [2] Multiple. *Dont Use Exceptions For Flow Control*. Feb. 18, 2019. url: <http://wiki.c2.com/?DontUseExceptionsForFlowControl> (visited on 02/18/2019).