Hochschule
Kaiserslautern
University of
Applied Sciences

PENTASYS
Unser Maßstab ist der Mensch

Hochschule Kaiserslautern
- Fachbereich Informatik und Microsystemtechnik -

Der Titel der Arbeit

Abschlussarbeit zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

vorgelegt von

Vorname Nachname
12345

Betreuer Hochschule:	Prof. Dr.
Betreuer PENTASYS:	B. Sc.

Abstract

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

> Dies ist ein Zitat.

verstand, scheinen nun doch vorueber zu Dies ist der Text sein.
siehe: http://janeden.net/die-praeambel

Inhaltsverzeichnis

Kapitel 1

Einleitung

Durch den starken Wachstum der IT entstehen immer wieder neue Möglichkeiten Anwendungen zu realisieren. Durch den technischen Fortschritt werden schon bekannte Muster und Architekturen weiter entwickelt oder Ideen für die neu entstandenen Anforderungen umgesetzt. Eine dieser Ideen ist *REACTIVE PROGRAMMING*. Vor noch nicht allzu langer Zeit waren Monolithen die auf eigens gehosteten Servern ausgerollt wurden der Stand der Dinge. Durch die mittlerweile entstandene Vielfalt an Endgeräten wie Smartphones, das Deployment in der Cloud oder die Menge an spezialisierten Programmiersprachen, Frameworks und Entwicklungswerkzeugen haben sich Anforderungen herausgestellt die sich mit bekannten Lösungen wie zum Beispiel einer objektorientierten Herangehensweise nicht zur vollen Zufriedenheit erfüllen lassen. Für einen Benutzer ist es üblich, dass Änderungen sofort sichtbar sind und angefragte Daten in Bruchteilen einer Sekunde bereit stehen, und das zu jedem Zeitpunkt. Kann eine Applikation dies nicht leisten, kann die User Expierience [1] in Mitleidenschaft gezogen werden, was bei der Menge an Diensten gleicher Art dazu führen kann, dass Benutzer auf die Dienste von Mitbewerber zurück greifen. Um diesem vorzubeugen wurde aus vorhandenen Konzepten ein grundlegendes Manifest für *Reaktive Systeme*[2] erstellt. Die im Manifest angeführten Eigenschaften werden im Verlauf dieser Arbeit noch genauer aufgegriffen. Es sei nur jetzt schon gesagt, dass zur Umsetzung der Kriterien für ein Reaktives System, die Verwendung von Reactive Programming nicht notwendig ist, jedoch meist sinnvoll scheint.
Ziel dieser Arbeit ist es, eine Einführung in die Welt von Reactive Programming zu geben. Dazu wird im nächsten Kapitel eine Einordnung in das Gesamtbild der Softwareentwicklung durchgeführt. Ebenso werden die Eigenschaften und Eigenheiten von Reactive Programming beschrieben. Weiterhin gibt es einen Überblick über einige Frameworks mit deren Hilfe ein reaktives Programmieren realisiert werden kann. Darauf folgend wird eine Implementierung einer Beispielanwendung besprochen, um dem Vorangegangen eine praktische Anwendung hinzuzufügen.

[1] TODO: User Experience Def
[2] [Bon14]: Reactive Manifesto 2.0. www.reactivemanifesto.org

Kapitel 2

Was ist Reactive Programming

Ist mein neu in der Domäne von reaktiver Entwicklung stellt man schnell fest, dass allerhand mögliche Definitionen und Beschreibungen findet was Reactive Programming denn zu sein scheint. Bevor jedoch hier eine Definition erläutert wird, muss zwischen unterschiedlichen Begrifflichkeiten differenziert werden: *Reactive Systems, Functional Reactive Programming* und natürlich *Reactive Programming*.

2.1 Was bedeutet *"reactive"* im Kontext der Softwareentwicklung

Wie die Wortherkunft verlauten lässt, wird etwas als reaktiv bezeichnet, wenn eine Reaktion durch eine vorangegangene Aktion ausgelöst wird. Bei diesen Aktionen handelt es sich meist um Veränderungen an verwendeten Daten und stattfindende Ereignisse(*Events*). Eine gutes Beispiel zur Veranschaulichung ist die Benutzeroberfläche(*GUI*). Ein Benutzer bestätigt eine vorgenommene Eingabe durch das klicken eines Button innerhalb der GUI. Dieses Event sorgt dafür, dass die Applikation einen vorgegebenen Vorgang ausführt. Bei diesem Vorgehen gibt es nun einige Punkte zu beachten. Grundsätzlich gestaltet sich ein programmatischen Vorgehen als komplex, wenn es sich um ein imperative, sequentielle Realisierung handelt. Durch die unterschiedlichen Events die innerhalb der GUI auftreten können(Mausklick, Tastendruck, usw.) ist ein klassischer imperativer sowie sequentieller Ablauf den Programmcodes nicht realistisch, da kein Entwickler weiß, in welcher Reihenfolge und zu welchem Zeitpunkt ein beziehungsweise welches Event ausgelöst wird. Somit spielt hier die *Inversion of Control*[1], also das Umkehren der Kontrolle, eine große Rolle. Diese besagt, dass nicht der geschrieben Code den Ablauf beschreibt, sondern die Kontrolle bei dem Framework[2], welches für die Interaktion zuständig ist, liegt, und dieses entscheidet wiederum wie und wann auf ein Event reagiert wird. Im Code spiegelt sich das dadurch wider, das mögliche Reaktionen, meist als Methoden oder Funktionen realisiert, an die mögliche eintreffenden Ereignisse gebunden werden. Dies geschieht über so

[1] Vgl. [Fow05], https://martinfowler.com/bliki/InversionOfControl.html.

[2] In diesem Fall ein Framework für das Realisieren einer GUI. Vgl. [wik].

genannte Event-Handler beziehungsweise Event-Listener oder über die Verwendung von Callbacks[3]. Ein bewährtes und klassisches Vorgehen bei Anwendungsanforderungen dieser Art ist das schon 1994 von Erich Gamma und seinen Mitstreitern beschriebenen Entwurfsmuster[4], dem *Observer-Pattern*[5]. Man betrachtet hier grundlegend zwei Varianten: *push* und *pull*. [6].

2.1.1 Differenzierung zwischen Reactive Proramming und Reactive Systems

Das Manifest bezieht sich eigentlich auf die Systeme. Wo kommt nun RP in Spiel?

2.2 Reactive Programming vs. Functional Reactive Programming

FRP findet nun mal in Funktionalen Programmiersprachen statt. Java is jedoch OO und mit Java 8 und dem Rx Frameworks werden funktionale Eigenschaften in der OO Sprache eingebracht. Unterschiede müssen noch genau belegt[7] werden. Der grundlegende Gedanke reaktiver Systeme wurde schon im Jahre 1985 in einem Paper von D. Harel und A. Pnueli beschrieben[8].

2.3 Reactive Programming - Ein neues Programmierparadigma?

Wie gliedert man einen Programmierstil ein? [9]

2.3.1 Was versteht man unter Programmierparadigmen

Erklärung was sind Paradigmen und wieso werden sie definiert.

2.3.2 Vergleich: Reactive Programming und Objekt orientierte Programmierung

Eventuell bessere mit Funktionaler Programmierung zu vergleichen. Muss noch genauer betrachtet werden. Soll die Unterschiede zu den gängigen, bekannten Methoden aufzeigen.

[3]TODO Beispiel Event-Handler -> ActionEvent, Click on Button; Quellen angeben und Beschreibung zu Beiden Codebeispielen

[4][GHJV11]: Dieses Buch beschreibt viele noch heute verwendete Entwurfsmuster. Die Autoren sind in die Geschichte als die Gang Of Four (*GoF*) eingegangen.

[5]Hier Beispiel vom OP sowie Übersetzung Beobachtermuster; Observer Pattern und Erklärung

[6]Hier gute Beispiele für push und pull Variante

[7][Loh16]

[8][HP85]

[9][Bai13]

2.4 Überblick über bekannte Frameworks und ihre Eigenschaften

Überblick quer über die gängigen Programmiersprachen.

2.4.1 Reactivex.io

Kurze Erläuterung zu der Entstehung von Reactive Extensions

2.4.2 Allgemeine Übersicht

Rx Frameworks zu den jeweiligen Sprachen. Frameworks wie z.B. Akka[10].

2.4.3 Übersicht spezielle für die Entwicklung mit Java

RxJava. Reactive Streams Konvention. Java 9 Api Änderung bzgl. Reactive Streams.

Framework für JavaFX - RxJavaFX

Einführung und Eigenschaften erläutern

2.5 Testen von reaktivem Code mit dem JUnit Framework

Noch nichts genaues. Muss noch geschaut werden wie die Funktionalität von JUnit RP abdeckt.

[10] [Kar16]

Kapitel 3

Einführung in Reactive Programming mit RxJava

Beschreibung wieso RxJava. Beschreibung was beschrieben wird.

3.1 Wie funktioniert Reactive Programming

Einleitung zum Aufbau: Klassenübersicht des Frameworks mit Erklärung.

3.1.1 Syncronität

Sync vs. Async - was bringt RP in dieser Hinsicht

3.1.2 Parallelisierung

Concurrency vs. Parallelism - was tritt wie wann auf bzw. kann wie wann angewandt werden

3.1.3 Push vs. Pull

Wichtigster Unteschied. Observable als Gegenpart zu Interable - somit Push vs. Pull Vergleich.

3.2 Rx.Observable

Interface Übersicht. Nutzen und Anwendung anhand von Beispiel. Hot vs. Cold

3.3 Rx.Observer

Was kann Observer -> Interface Übersicht

3.3.1 Rx.Subscriber

Was ist speziell am Subscriber -> Interface Übersicht

3.4 Operationen und Transformationen

Erläuterung von den Stadien der Operation von Beginn über Mitte bis Ende.

3.4.1 Exkursion: Streams API Java 8

Beschreibung was Streams darstellen, wie sich Observables im Vergleich verhalten

3.4.2 Operation filter()

Beispiel und Perlenbild. Einsatz beschreiben

3.4.3 Transformation map()

Beispiel und Perlenbild. Einsatz beschreiben

3.4.4 Transformation flatMap()

Beispiel und Perlenbild. Einsatz beschreiben

3.4.5 Operation merge()

Beispiel und Perlenbild. Einsatz beschreiben

3.4.6 Operation zip()

Beispiel und Perlenbild. Einsatz beschreiben Eventuell noch mehr Operationen

Kapitel 4

Beispiel: Implementierung eines Systemmonitors

Beschreibung der Funktionen der Anwendung. Überblick über Projekt/Klassenstruktur. Verwendete Tools und Versionen.

4.1 API für Systemwerte

Framework für die Systemwerte kurz erläutern. Grobes Vorgehen beschreiben wie man es in etwa Umsetzen kann.

4.1.1 Beschreibung Klasse 1

4.1.2 Beschreibung Klasse 2

4.1.3 Beschreibung Klasse 3

4.1.4 Beschreibung Klasse 4

4.2 Client für API: GUI zur Repräsentation der Systemwerte

4.2.1 Beschreibung Klasse 1

4.2.2 Beschreibung Klasse 2

Literaturverzeichnis

[Bai13] Bainomugisha, Carreton Andoni Lombide van Cutsem Tom Mostinckx Stijn Meuter Wolfgang d. Engineer: A survey on reactive programming. In: *ACM Computing Surveys* 45 (2013), Nr. 4, S. 1–34. http://dx.doi.org/10.1145/2501654.2501666. – DOI 10.1145/2501654.2501666. – ISSN 03600300

[Bon14] Bonér, Farley Dave Kuhn Roland Thompson M. Jonas: the-reactive-manifesto-2.0. (2014). http://www.reactivemanifesto.org/

[Fow05] Fowler, Martin: *InversionOfControl*. https://martinfowler.com/bliki/InversionOfControl.html. Version: 2005

[GHJV11] Gamma, Erich ; Helm Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. 39. printing. Boston : Addison-Wesley, 2011 (Addison-Wesley professional computing series). – ISBN 0201633612

[HP85] In: Harel, D. ; Pnueli, A.: *On the Development of Reactive Systems*. Berlin, Heidelberg : Springer Berlin Heidelberg, 1985. – ISBN 978–3–642–82453–1, 477–498

[Kar16] Karnok, Dávid: *Operator-fusion (Part 1)*. https://akarnokd.blogspot.de/2016/03/operator-fusion-part-1.html. Version: 2016 (Advanced Reactive Java)

[Loh16] Lohmüller, Jan C.: *Reactive Programming – Mehr als nur Streams und Lambdas*. https://www.informatik-aktuell.de/entwicklung/programmiersprachen/reactive-programming-mehr-als-nur-streams-und-lambdas.html. Version: 2016

[wik] *List von GUI-Bibliotheken*. https://de.wikipedia.org/wiki/Liste_von_GUI-Bibliotheken

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis