



Hochschule
Kaiserslautern
University of
Applied Sciences



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.

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

Inhaltsverzeichnis

1	Einleitung	1
2	Was ist Reactive Programming	2
2.1	Was bedeutet " <i>reactive</i> " im Kontext der Softwareentwicklung	2
2.1.1	Differenzierung zwischen Reactive Programming und Reactive Systems . .	3
2.2	Reactive Programming vs. Functional Reactive Programming	4
2.3	Reactive Programming - Ein neues Programmierparadigma?	5
2.3.1	Was versteht man unter Programmierparadigmen	5
2.3.2	Vergleich: Reactive Programming und Objekt orientierte Programmierung	5
2.4	Überblick über bekannte Frameworks und ihre Eigenschaften	5
2.4.1	Reactivex.io	5
2.4.2	Allgemeine Übersicht	5
2.4.3	Übersicht spezielle für die Entwicklung mit Java	5
2.5	Testen von reaktivem Code mit dem JUnit Framework	6
3	Einführung in Reactive Programming mit RxJava	7
3.1	Wie funktioniert Reactive Programming	7
3.1.1	Synchronität	7
3.1.2	Parallelisierung	7
3.1.3	Push vs. Pull	7
3.2	Rx.Observable	7
3.3	Rx.Observer	7
3.3.1	Rx.Subscriber	8
3.4	Operationen und Transformationen	8
3.4.1	Exkursion: Streams API Java 8	8
3.4.2	Operation filter()	8
3.4.3	Transformation map()	8
3.4.4	Transformation flatMap()	8
3.4.5	Operation merge()	8
3.4.6	Operation zip()	8

4	Beispiel: Implementierung eines Systemmonitors	9
4.1	API für Systemwerte	9
4.1.1	Beschreibung Klasse 1	9
4.1.2	Beschreibung Klasse 2	9
4.1.3	Beschreibung Klasse 3	9
4.1.4	Beschreibung Klasse 4	9
4.2	Client für API: GUI zur Repräsentation der Systemwerte	9
4.2.1	Beschreibung Klasse 1	9
4.2.2	Beschreibung Klasse 2	9
	Literaturverzeichnis	I
	Abbildungsverzeichnis	II
	Tabellenverzeichnis	III

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 Experience¹ 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 Vorangegangenen eine praktische Anwendung hinzuzufügen.

¹TODO: User Experience Def

²[Bon14]: Reactive Manifesto 2.0. www.reactivemanifesto.org

Kapitel 2

Was ist Reactive Programming

Ist man neu in der Domäne von reaktiver Entwicklung stellt man schnell fest, dass allenthalben 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*. Vorab wird jedoch ergründet worum es sich grundlegend handelt und wieso es zur heutigen Zeit von Relevanz ist über die möglichen Verwendung von *reactive* zu sprechen.

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*). Ein 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. Dieses Vorgehen sorgt in der klassischen Programmierweise mit sequentiellm Ablauf sowie dem imperativen Ansatz grundsätzlich für ein stetig wachsendes Maß an Komplexität. 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 geschriebene Code den Ablauf beschreibt, sondern die Kontrolle bei dem Framework², 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

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

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

wider, das mögliche Reaktionen, meist als Methoden oder Funktionen realisiert, an die mögliche eintreffenden Ereignisse gebunden werden. Dies geschieht über so genannte Event-Handler beziehungsweise Event-Listener oder über die Verwendung von Callbacks³. Ein bewährtes und klassisches Vorgehen bei Anwendungsanforderungen dieser Art ist das schon 1994 von Erich Gamma und seinen Mitstreitern beschriebenes Entwurfsmuster⁴, dem *Observer-Pattern*⁵. Man betrachtet hier grundlegend zwei Ansätze: *push* und *pull*.⁶ Auch hier lässt sich wieder die GUI als gutes Beispiel heran ziehen. Man betrachtet eine Oberfläche welche die Temperatur im Raum anzeigt. Die Temperatur wird als Zahl in Grad Celsius sowie graphisch in einem Balken dargestellt. Obwohl beide Elemente auf die selben Daten zugreifen, stehen programmatisch die Objekte in keinem Zusammenhang. Um nun das Beobachtermuster zu implementieren, muss der sich ändernde Wert der zum Beispiel von einem Temperatursensor gemessen wird regelmäßig aktualisiert werden. Das Objekt, welches den Temperaturwert inne hält wird somit als beobachtbar deklariert, die beiden Objekte, die Ausgabe per Zahlenwert und der graphische Balken, werden bei diesem Objekt als Beobachter registriert. Wenn man den push-Ansatz verfolgt, wird jedem Beobachter benachrichtigt und das Objekt mit den geänderten Daten steht dem Beobachter zur Verfügung. Verfolgt man den pull-Ansatz, werden die Beobachter nur kurz darüber informiert das sich Werte innerhalb des zu beobachtenden Objekts geändert haben, müssen jedoch aktiv anfragen um die geänderten Werte zu erhalten. Je nach Vorgehen ergeben sich Vor- und Nachteile. Der push-Ansatz steht eher für lose Kopplung, da der Beobachter keine Details zu dem zu beobachteten Objekt braucht. Jedoch sinkt dadurch die Flexibilität, da Beobachterschnittstellen exakter beschrieben werden müssen damit der Beobachtete weiß welche Information weiter gereicht werden sollen. Die Kopplung im Vergleich zum pull-Ansatz ist nicht wirklich lose, da jeder Beobachter wissen muss, welche Daten das observierte Objekt repräsentiert und wie auf diese Daten zugegriffen werden kann. Jedoch findet sich hier die Flexibilität wieder, da jeder Beobachter wenn er Information braucht, exakt diese Daten abrufen kann und sich nicht auf die korrekte Datenverteilung des Beobachteten verlassen muss. Nach diesem Abschnitt sollte soweit klar sein, wie man ein reaktives Verhalten mit den schon bekannten Bordinstrumenten realisiert.

2.1.1 Differenzierung zwischen Reactive Programming und Reactive Systems

Das zuvor angesprochene Verhalten bezieht sich nur auf ein Beispiel. Wie jedoch wirkt es sich aus, wenn eine komplette Anwendung unter Verwendung dieses Stils entwickelt wird? Wie verhält es sich weiter wenn mehrere Teile reaktiv funktionieren und kommunizieren sollen? Eine

³TODO 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.

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

⁶Hier gute Beispiele für push und pull Variante

gute Richtlinie um Anwendungen reaktiv zu gestalten bietet das Reactive Manifesto⁷. Laut Jonas Bonér und den vielen Unterstützern sind vier Bestandteile essentiell, damit eine Anwendung die Anforderung erfüllt um sich reaktiv zu verhalten. Die Ansicht des Manifest stützt sich auf einen Architektur- beziehungsweise Designstil und soll als Grundlage zur Entwicklung Reaktiver Systeme dienen. Die folgenden Erklärungen sind dem Manifest entnommen und sollen einen Verständnis zu den vier Eigenschaften bieten. Wie in diesem beschrieben sind Reaktive Systeme:

- **Antwortbereit (engl. responsive):**

Ein System muss immer zeitgerecht antworten. Die Antwortbereitschaft ist die Grundlage für die Benutzbarkeit besagten Systems. Ebenso wird um eine Fehlerbehandlung durchführen zu können eine geregelte Antwortbereitschaft vorausgesetzt.

- **Widerstandsfähig (engl. resilient):**

Ein System muss auch bei Ausfällen die Antwortbereitschaft aufrecht erhalten. Die wird durch Replikation der Funktionalität, der Isolation von Komponenten sowie dem Delegieren von Verantwortung erzielt.

- **Elastisch (engl. elastic):**

Das System muss bei sich ändernden Lasten die Funktionalität und Antwortbereitschaft aufrecht erhalten. Ressourcen müssen den auftretenden Lasten, ob steigend oder sinken, angepasst werden können. Ebenso müssen Engpässe innerhalb des Systems unterbunden werden um die Elastizität zu bewahren.

- **Nachrichtenorientiert (engl. message driven):**

Ein loses System soll zur Kommunikation zwischen den Komponenten auf asynchrone, ortsunabhängige Nachrichtenübermittlung zurück greifen. Somit ist nicht relevant auf welchen Rechner die einzelnen Komponenten ausgeführt werden, wodurch wiederum eine gute Skalierbarkeit entsteht.

Das Manifest erwähnt jedoch in keinsten Weise den Zusammenhang von Systemen zum reaktiven Programmieren. Auch aus diesem Grund hat Jonas Bonér einen weiteren Artikel⁸ verfasst, der sich dieser Thematik annimmt. Die grundlegenden Unterschiede wurde in dem Artikel wie folgt zusammengefasst.

- Reactive Programming eine Teilmenge von Reaktiven Systemen auf Implementierungsebene
- Reactive Programming liefert Leistung und effektive Ressourcennutzung auf Komponentenebene, speziell für Softwareentwickler

⁷[Bon] -> ebenso Bild von Manifest einfügen. Deutsches Manifest in Quellen verlinken

⁸[Bon14]

- Reaktive Systeme hingegen bieten Robustheit und Elastizität auf Systemebene zur Gestaltung von Cloud-kompatiblen oder verteilten Anwendungen, speziell für Softwarearchitekten oder DevOps
- Es ist von großem Vorteil reaktives Programmieren innerhalb der Bestandteile von reaktiven Systemen zu verwenden
- Es ist ebenso von Vorteil reaktive Systeme zur Interaktion zwischen reaktiv programmierten Komponenten zu verwenden

Wie aus diesen Punkten klar wird, liegt also der genaue Unterschied zwischen reaktivem Programmieren und reaktiven System aus welcher Perspektive die Betrachtung stattfindet. Architektonisch wird von Systemen gesprochen, die innerhalb und zur Interaktion mit anderen reaktiv reagiert. Betrachtet man aus Entwicklersicht eine Komponente, kann diese unter Zuhilfenahme von reaktivem Programmieren für ein asynchrones, paralleles Verhalten entwickelt werden. Wie Jonas Bonér schon schreibt, ist das Zusammenspiel beider sehr oft hilfreich um das gewünschte Ziel zu erreichen. Ein weiterer wichtiger Punkt ist die Unterscheidung von *ereignisorientiert* zu *nachrichtenorientiert*. Nachrichten werden auf Systemebene genutzt. im Gegensatz zu Ereignissen sind Nachrichten klar an einen Empfänger adressiert. Ereignisse treten auf und müssen beobachtet werden um das gewünschte Resultat zu erzielen. Nachrichten sind somit gut geeignet bekannte Empfänger in einem verteilten System, zum Beispiel über das Netzwerk, zu kontaktieren. Innerhalb der Komponente besteht die Funktion (man denke hier wieder an die Nutzeroberfläche) oft aus der Reaktion auf Ereignisse unterschiedlicher Art. Somit herrscht hier ein Ereignis-getriebenes Verhalten.

2.2 Reactive Programming vs. Functional Reactive Programming

FRP findet nun mal in Funktionalen Programmiersprachen statt. Java ist jedoch OO und mit Java 8 und dem Rx Frameworks werden funktionale Eigenschaften in der OO Sprache eingebracht. Unterschiede müssen noch genau belegt⁹ werden. Der grundlegende Gedanke reaktiver Systeme wurde schon im Jahre 1985 in einem Paper von D. Harel und A. Pnueli beschrieben¹⁰.

2.3 Reactive Programming - Ein neues Programmierparadigma?

Wie gliedert man einen Programmierstil ein? ¹¹

⁹[Loh16]

¹⁰[HP85]

¹¹[Bai13]

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.

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. Akka¹².

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.

¹²[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 Synchronität

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 Unterschied. Observable als Gegenpart zu Iterable - 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
- [Bon] BONÉR, Klang V. Jonas: Reactive Programming versus Reactive Systems. <https://info.lightbend.com/reactive-programming-versus-reactive-systems.html>
- [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