Hochschule Bremen

Exposè - Development of an MVI framework for the Android platform

Media Computer Science B.Sc.

Roman Quistler

June 3, 2019

Contents

1 Introduction

Das Entwickeln von Applikationen ist nicht einfach.

2 Problem and solution

2.1 Problem

Applications today have increased in complexity compared to its counterpart just a few years ago. Especially in the field of frontend development the amount 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. The same applies to the mobile world: social networks, navigation, sharing and editing files together is a common want by the user. A large part of applications are thus interacting with APIs, accessing local or external databases and communicating with the underlying operating system itself (eg the periodic recording of the location via GPS or Wi-Fi).

The challenge a developer faces is to come up with a general approach that bridges the gap between what the users sees and the source code of the application - mainly: How to structure the user facing parts of the codebase also known as the presentation layer [2, 3] in a layered architecture. [4].

Like many other problems in software development, this one is a problem that has existed for quite some time and concerns a large part of the developers. It is therefore a common problem. This often results in the endeavour to develop a universally valid concept that counteracts these problems and points out possible solutions. The sought out concept here is defined as an "architectural" pattern. [5, 6, 7] Its goal is to lay out a structure and architecture of source code in such a way that it is modular, flexible and reusable. At the same time, the developer is invited to follow a pattern and produce consistent and readable code. This promotes quality and maintainability of the application. Over time, many "architectural" patterns have evolved for structuring code within the presentation layer. These include, among others, the following, sorted in ascending order by year of publication: Model-View-Controller (MVC - 1979) [8, 9, 10, 11], Model-View-Presenter (MVP - 1996) [12, 13, 14], Model-View-ViewModel (MVVM) [15, 16, 14] und - relatively new to the group - Model-View-Intent (MVI - 2015) [17, 18, 19]. Each of the patterns listed here serves the same purpose: the strict separation of the user interface from the underlying (business) logic.

MVI also makes use of this idea. Brought to life by André (Medeiros) Staltz, MVI encapsulates the interaction between the user and the application as a cycle (as shown in figure 1), where the data flows unidirectionally [20, 21, 22]. It takes its inspiration from three concepts: The original MVC as introduced by Trygve Reenskaug in 1979, the JavaScript framework Redux [23] and the JavaScript library React [24]. The idea of

a cycle is based on the assumption that the output (e.g. a click) from a user resembles the input for the program. The program in turn produces an output which becomes the input for the user. This concept can be embodied as a chain of mathematical functions:

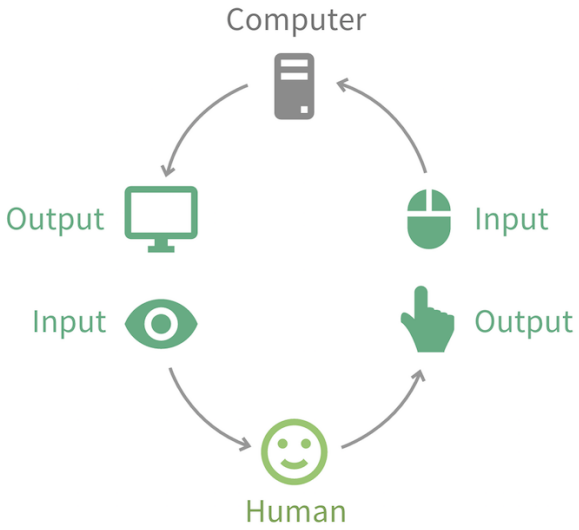

Figure 1: User and Computer as Input and Output

Source: https://cycle.js.org/dialogue.html

f(g(a())) or view(model(intent(event))). When the user interface registers an event (e.g. a click) it will be passed to the intent functions as a parameter. The purpose of the intent-function is to translate the user generated event into something the application can understand and work with. In addition, the event contains data that is necessary for further processing. This requires the event to be accurately described, unique and expressive. The model function then consumes the output (intent function) as its input. Its job is it to create a new model without changing the last one. That's what is referred to as immutability. [25, 26] It's a concept that belongs (but is not exclusive) to a programming paradigm called functional programming [27], where MVI adopts some of its ideas from.
Finally the view function receives the model as its input and takes care of rendering it (a visual representation of the model for the user). In order to achieve the cycle effect and the unidirectional data flow, MVI makes use of reactive programming [28, 29] and the underlying "Observer" pattern []. To quote Andre Staltz:

> *Model-View-Intent (MVI) is reactive, functional, and follows the core idea in MVC. It is reactive because Intent observes the User, Model observes the Intent, View observes the Model, and the User observes the View. It is functional because each of these components is expressed as a referentially transparent function over streams. It follows the original MVC purpose because View and Intent bridge the gap between the user and the digital model, each in one direction.* ([30], Andre Staltz)

Due to MVI being based on functional paradigms and hence using pure functions, it has to fight a common problem in applications called ”side effects”. As mentioned above, it is ofen a requirement to access APIs or a local database, but calling them inside of a pure function is technically not allowed.
Given this description of MVI and the mentioned problems, the questions that arise are: What does an event look like? How is immutability achieved? Since MVI does not have a ”Controller”, ”Presenter” or a ”ViewModel”: Where does the user user-interface logic belong? And how is separation of concerns possible? To summarize, it becomes a question of implementation and design. What does a developer have to do in order to implement or reap the benefits from this concept of MVI?

why reactive, what does reactive solve for MVI? Streams? state?

2.2 Solution

In order to overcome the mentioned problem, the aim will be to take a deeper look at what is important to understand and successfully implement Model-View-Intent. For this it is essential to get a better overview over archhitectural patterns in general. Before this can be done meaningfully the first step must be to analyze why they came to life in the first place and what the exact motivation was. Further the problem they try to solve shall be layed out in more detail to get the bigger picture. In order to do justice to that, it is a necessity to review the history of architectural patterns and to work out how they have evolved over time. By doing that it will also be possible to track down the issues that each of this architectural patterns brings with it.

After this is accomplished, the next step is to study the origins of Model-View-Intent. The beginning is made by the JavaScript library React and the application architecture Flux, which introduced the concept of the unidirectional flow for building user interfaces. It is necessary to clarify which idea is hidden behind it and what it does better than previous solutions. The important components of this framework shall be examined and shortly explained. Following on from this, a further improvement(?) of Flux will be discussed: Redux. Differences to the idea and terminology of Flux will be pointed out and evaluated.

Since Mode-View-Intent takes some of its inspiration regarding its structure from Mode-View-Cotroller, it is imperative (crucial?) to dedicate a section to it as well. In this section, the similarities and differences will be identified and illustrated to get a better grasp as to why it served as a source of inspiration.

It then will be set out how Model-View-Intent functions in detail, what mechanism it employs/utilizes and which components it is made up of. In addition, it is examined which ideas it has adopted from the topics discussed before, whey they were chosen and how they interoperate with each other.

Thereafter, two other major building blocks of Model-View-Intent will be investigated. The first one being ”functional” programming and the second one ”reactive” program-

ming. Since both are classified as more complex paradigms, they will be briefly explained and it will be outlined to what role they play in Model-View-Intent.

The penultimate step is to explore already available solutions and gain valuable knowledge for the own realization of a framework. The challenge is to determine how they implement Model-View-Intent and to comprehend what they do well, badly or not at all - also in the scope of the Android platform.

The next task and final objective is then to use the acquired knowledge to conceptualize and develop a framwork that simplifies the implementation of Model-View-Intent in Android. Its purpose is to reduce the cognitive burden for a developer who is interested in deploying this very same pattern in their application. Hence it is also important to address the peculiarities of the Android platform when it comes to creating the concept. This in turn makes it mandatory to check and assess to what extent Model-View-Intent can be implemented in the context of Android and what difficulties it migh entail.

At last an example application shall be developed to showcase and verfiy the intended behaviour of the framework.

3 Specific Tasks

The following tasks are necessary for the realization of the Bachelor Thesis:

- Research: Collection and comparison of further literature, for a deeper understanding of the topic.
- Requirement analysis:
- Design: Detailed design of an architecture for a framework to support the implementation of MVI in mobile (Android) apps.
- Prototypical implementation: Development of a framework in which the determined requirements are implemented. Prototypical use of the framework within an example application.
- Evaluation: Review and discussion of the results.
- Quality assurance: Verifying intended behaviour of the framework via tests.

4 Early Structure

Summary/Abstract
Statutory Affidavit/Declaration of Authorship

1. Introduction
   - 1.1. Problem area/Motivation
   - 1.2. Objective(s) of the work
   - 1.3. Task definition
   - 1.4. Solution (approach)
2. Fundamentals and related work
   - 2.1. Unidirectional data flow: Flux and Redux
   - 2.2. Functional programming
     - 2.2.1. Pure Functions
     - 2.2.2. Immutability
     - 2.2.3. Expressions (vs Statements)
     - 2.2.4. Function Composition
   - 2.3. Reactive Programming
     - 2.3.1. Observer Pattern
     - 2.3.2. Iterator Pattern
     - 2.3.3. Streams
3. MVI (ggf. eigenes Kapitel)
   - 3.1. Architecture Patterns
     - 3.1.1. MVC (Model-View-Controller)
     - 3.1.2. MVP (Model-View-Presenter)
     - 3.1.3. MVVM (Model-View-ViewModel)
   - 3.2. View
   - 3.3. Intent and Action
   - 3.4. Model and State
   - 3.5. Middleware
   - 3.6. Finite state machine
4. Related work
   - 4.1. Examination of existing MVI Frameworks/Libraries
     - 4.1.1. MVICore

5 Timetable

Planned start date: 15 June 2019
Processing time: 9 weeks
Table 1 shows the planned work packages and milestones:

M1	Official beginning of the work	15.06.2019
	• Setting up the word processor • Requirements analysis • Writing the Thesis: Chapter 2 (Requirements Analysis)	1.5 weeks
M2	completion of the analysis phase	22.06.2019
	• Research • Writing the Thesis: Chapter 3 (Fundamentals)	1.5 weeks
My	First version of the complete Thesis	22.06.2019
	• Proofreading • Printing and binding	1 week
Mx	Submission of the thesis	22.06.2019

List of Figures

Book References

[2] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, Jan. 13, 2013, pp. 19–22.

[3] Mark Richards. *Software Architecture Patterns*. O'Reilly Media, Feb. 2015, pp. 1–5.

[4] Donald Wolfe. *3-Tier Architecture in ASP.NET with C sharp tutorial*. SitePros2000.com, Jan. 13, 2013.

[5] Mark Richards. *Software Architecture Patterns*. O'Reilly Media, Feb. 2015.

[6] Douglas C. Schmidt Frank Buschmann Kevin Henney. *Pattern-Oriented Software Architecture, On Patterns and Pattern Languages*. John Wiley Sons, Apr. 30, 2007.

[7] Erich Gamma. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education India, Oct. 2000.

[11] Brad Wilson Jon Galloway, K. Scott Allen, and David Matson. *Professional ASP.NET MVC 5*. Pearson Education India, Aug. 4, 2014.

[14] Adam Freeman and Steven Sanderson. *Pro ASP.NET MVC 3 Framework*. Apress, June 26, 2011, pp. 68–70.

[18] Sergi Mansilla. *Reactive Programming with RxJS: Untangle Your Asynchronous JavaScript Code*. PRAGMATIC BOOKSHELF, Jan. 12, 2016, pp. 112–114.

[20] Adam Boduch. *Flux Architecture*. Packt Publishing, Jan. 30, 2017, pp. 27, 198, 312.

[21] Robin Wieruch. *Taming the State in React: Your journey to master Redux and MobX*. CreateSpace Independent Publishing Platform, June 5, 2018, p. 14.

[22] Ilya Gelman and Boris Dinkevich. *The Complete Redux Book*. Leanpub, Jan. 30, 2017, pp. 6–7.

[23] James Lee, Tao Wei, and Suresh Kumar Mukhiya. *Redux Quick Start Guide: A beginner's guide to managing app state with Redux*. Packt Publishing, Feb. 28, 2019.

[25] Joshua Bloch. *Effective Java - Third Edition*. Addison-Wesley Professional, Dec. 17, 2017, pp. 80–86.

[27] Pierre-Yves Saumont. *Functional Programming in Java: How functional techniques improve your Java programs*. Manning Publications, Jan. 27, 2017.

[28] Tomasz Nurkiewicz and Ben Christensen. *Reactive Programming with RxJava*. O'Reilly Media, Oct. 27, 2016.

Report References

[8] Trygve Reenskaug. *THING-MODEL-VIEW-EDITOR - an Example from a planningsystem*. The original MVC report. May 12, 1979. url: http://heim.ifi.uio.no/~trygver/1979/mvc-1/1979-05-MVC.pdf (visited on 05/23/2019).

[9] Trygve Reenskaug. *MODELS - VIEWS - CONTROLLERS*. Dec. 10, 1979. url: http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf (visited on 05/23/2019).

[10] Glenn E. Krasner and Stephen T. Pope. *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System*. ParcPlace Systems, Inc, 1988.

[12] Mike Potel. *MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java*. Taligent, Inc., 1996. url: http://www.wildcrest.com/Potel/Portfolio/mvp.pdf.

[16] Erik Sorensen and Marius Iulian Mihailes. *Model-View-ViewModel(MVVM)Design Pattern using Windows Presentation Foundation (WPF) Technology*. University of Southern Denmark, The Maersk Mc-Kinney Moller Institute, Titu Maiorescu University, Faculty Science, and Information Technology, 2010. url: http://megabyte.utm.ro/articole/2010/info/sem1/InfoStraini_Pdf/1.pdf.

[26] Christian Haack et al. *Immutable Objects in Java*. Technical. Apr. 28, 2006.

Thesis References

[24] Naimul Islam Naim. "ReactJS: An Open Source JavaScript Library for Frontend Developement". Bachelor. Metropolia University of Applied Sciences, May 30, 2017.

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).

[13] Martin Fowler. *Model-View-Presenter (MVP)*. July 18, 2006. url: https://martinfowler.com/eaaDev/uiArchs.html#Model-view-presentermvp (visited on 05/23/2019).

[15] John Gossman. *Introduction to Model/View/ViewModel pattern for building WPF apps*. Oct. 8, 2005. url: https://blogs.msdn.microsoft.com/johngossman/2005/10/08/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps/ (visited on 05/17/2019).

[17] Andre Staltz. *What if the user was a function? by Andre Staltz at JSConf Budapest 2015*. Youtube. June 4, 2015. url: https://www.youtube.com/watch?v=1zj7M1LnJV4 (visited on 05/18/2019).

[19] Hannes Dorfmann. *Model-View-Intent on Android*. Mar. 4, 2016. url: http://hannesdorfmann.com/android/model-view-intent (visited on 05/23/2019).

[29] Andre Medeiros (Staltz). *The introduction to Reactive Programming you've been missing*. 2014. url: https://gist.github.com/staltz/868e7e9bc2a7b8c1f754 (visited on 05/20/2019).

[30] Andre Staltz. *What MVC is really about*. url: https://cycle.js.org/model-view-intent.html#model-view-intent-what-mvc-is-really-about (visited on 05/23/2019).