

Inhaltsverzeichnis

| | | |
|----------|--|----------|
| 1 | Einleitung | 1 |
| 1.1 | Problemfeld | 1 |
| 1.2 | Ziel der Arbeit | 1 |
| 1.3 | Aufbau der Arbeit | 1 |
| 2 | Grundlagen | 2 |
| 2.1 | Unidirektionaler Datenfluss: Flux, Redux und React | 2 |
| 2.1.1 | Flux | 2 |

1 Einleitung

Bis zum Jahre 1973 und der Entwicklung des ersten Eingabegerätes mit Graphischer Oberfläche (GUI) (Xerox Alto [1]) erfolgte die Interaktion mit einem Computer im Wesentlichen über eine Konsole. Dies reduzierte die Ein- und Ausgabe eines Programms auf rein textuelle Elemente. Seither hat sich viel getan: Die Erschaffung des Internets leitet den Beginn von Webseiten ein, welche von einer anfänglich statischen Ausprägung zu der heutigen dynamisch und komplexen wuchsen. Dazu gesellten sich im Laufe der Zeit mobile Endgeräte - zu Beginn bestückt mit Tasten für die Eingabe, sowie einem primitiven Bildschirm für die Anzeige finden sich heutzutage vorwiegend leistungsfähige, auf einem kapazitiven Touchscreen basierende Smartphones wieder. Hierbei ist über die Jahre der Funktionsumfang von Betriebssystem und Applikationen im Allgemeinen gestiegen.

1.1 Problemfeld

Die Herausforderung für einen Entwickler ist es, die Nutzerschnittstelle (auch: Präsentation Layer [2]) , innerhalb einer "Drei-Schichtenarchitektur"[3] sinnvoll aufzubauen und dabei die Modellierung und Verwaltung des Zustandes innerhalb einer Applikation zu berücksichtigen. Hierfür müssen auch externe Vorkommnisse wie bswp. die Aktualisierung der zugrundeliegenden Datenbank miteinbezogen werden.

Da diese Problematik nicht erst seit kurzem sondern seit vielen Jahren besteht, wurden hierfür

1.2 Ziel der Arbeit

1.3 Aufbau der Arbeit

2 Grundlagen

In diesem Kapitel gilt es zu klären, auf welchen Grundlagen, Ideen und Konzepten Model-View-Intent beruht, wie diese miteinander fungieren und weshalb sie als Inspiration dienen.

2.1 Unidirektionaler Datenfluss: Flux, Redux und React

In einer Applikation existieren grundsätzlich zwei Komponenten: Eine, die der Nutzer wahrnehmen kann und eine, die für ihn unsichtbar bleibt. Bei ersterer handelt es sich meist um das, was der Nutzer(auf dem Bildschirm) sieht - die sogenannte „View“. Die zweite Komponente beschreibt die Ebene, welche das Geschehen observiert, darauf reagiert und den weiteren Verlauf (zum größten Teil) kontrolliert. Sie kann unter anderem als „Controller“ betitelt werden.

Der Zustand in dem sich eine Applikation befindet kann hierbei von beiden Seiten modifiziert und beobachtet werden. Ist dies der Fall, so handelt es sich um einen bidirektionalen Datenfluss. Bei dieser Variante entsteht die eventuelle Gefahr von kaskadierenden Updates (ein Objekt verändert ein anderes, welches wiederum eine Veränderung bei einem weiteren herbeiführt usw.) als auch in einen unvorhersehbaren Datenfluss zu geraten: Es wird schwer, den Datenfluss nachzuvollziehen. Des weiteren muss immer überprüft und sichergestellt werden, dass „View“ und „Controller“ synchronisiert sind, da beide den globalen Zustand darstellen. Schlussendlich verliert man zusätzlich die Fähigkeit zu entscheiden, wann und welcher Stelle der Zustand manipuliert wird.

Ein anderer Ansatz ist, den Datenfluss in eine Richtung zu beschränken und ihn damit unidirektional [4, 5] operieren zu lassen. Diese Variante erfreut sich an zunehmender Popularität seit der Bekanntmachung der „Flux“ [6] Architektur im Jahre 2015 von Facebook. [7]

2.1.1 Flux

Für die Einhaltung und Umsetzung eines unidirektionalen Datenfluss bedient sich „Flux“ bei zwei fundamentalen Konzepten: Der Zustand innerhalb einer Applikation wird als „single source of truth (SSOT)“ angesehen und darf keine direkte Änderung erfahren. Um dies zu Gewährleisten finden sich mehrere Komponenten in „Flux“ wieder:

Action: Eine Aktion beschreibt ein Ereignis, welches unter anderem vom Nutzer ausgelöst werden kann. Sie geben vor, wie mit der Anwendung interagiert wird. Jeder dieser Aktionen wird dabei ein Typ zugewiesen. Insgesamt sollte eine Aktion semantisch und deskriptiv bezüglich der Intention sein. Des weiteren können zusätzliche Attribute an eine Aktion gebunden werden.

```
{
  type: ActionTypes.INCREMENT,
  by: 2
}
```

Dispatcher: Er ist für die Entgegennahme und Verteilung einer Aktion an sogenannte „Stores“ zuständig. Diese haben die Möglichkeit sich beim ihm zu registrieren. Er besitzt die wichtige Eigenschaft der sequentiellen Verarbeitung, d.h., dass er zu jedem Zeitpunkt nur eine „Action“ weiterreicht. Sämtliche „Stores“ werden über alle Aktionen unterrichtet.

Store: Hier befinden sich die Daten, welche einen Teil des globalen Zustands einer Anwendung ausmachen. Die einzige Möglichkeit für eine Veränderung der dort hinterlegten Daten besteht durch eine Reaktion auf eine, vom „Dispatcher“ kommenden, Aktion. Bei jeder Modifikation der Daten erfolgt die Aussendung eines Events an eine „View“, das die Veränderung mitteilt. Ebenso findet sich hier ein Part der Anwendungslogik.

View: Die View ist für die Anzeige und Eingabe von Daten zuständig - sie ist die für den Nutzer sichtbare Komponente, mit welcher dieser interagiert. Ihre Daten erhält sie von einem „Store“, diesen sie abonniert und auf Änderungsereignisse hört. Erhält sie vom „Store“ ein solches Änderungsereignis, so kann sie die neuen Daten abrufen und sich selbst aktualisieren. Der View ist es nicht gestattet, den Zustand direkt zu verändern. Stattdessen generiert sie eine Aktion schickt diese an den Dispatcher.

Ein beispielhafter Ablauf bei einer Anwendung die einen Wert erhöht oder verringert kann wie folgt aussehen:

1. Die View bekommt einem Store zugewiesen, welcher für das inkre- und dekrementieren der angezeigten Zahl verantwortlich ist.
2. Sie erhält die Anfangszahl und stellt diese in einem leserlichen Format/einer Ansicht dar, welches es dem Nutzer ermöglicht, damit zu interagieren.
3. Betätigt dieser einen der Knöpfe welche die dargestellte Zahl verändern, so wird eine Action erstellt und an Dispatcher geschickt.
4. Dieser wiederum informiert alle Stores.
5. Jener Store der für die Verarbeitung dieser Aktion verantwortlich ist, modifiziert die Zahl in seiner internen Datenstruktur und kommuniziert dies über ein Änderungsereignis
6. Diejenige View, welche auf Änderungsereignisse lauscht, erhält die Daten und aktualisiert sich dementsprechend.

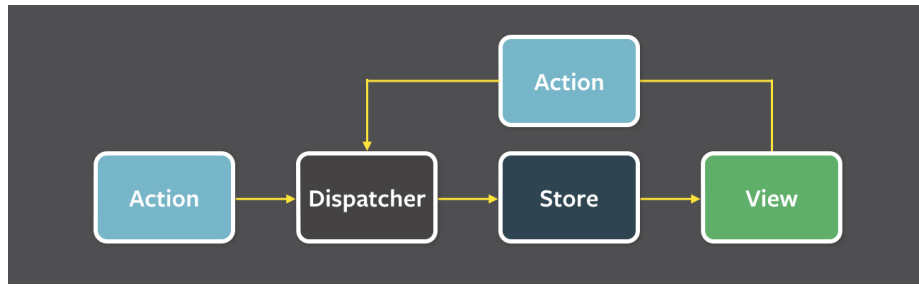


Abbildung 1: Datenfluss in der Flux Architektur

Anhand Abbildung 1 wird der unidirektionale Datenfluss deutlich erkennbar:

1. Die View schickt eine Aktion an den Dispatcher.
2. Dieser leitet diese an alle Stores weiter.
3. Der Store verarbeitet die Daten und informiert die View.

Abbildungsverzeichnis

| | | |
|---|--|---|
| 1 | Datenfluss in der Flux Architektur | 4 |
|---|--|---|

Book References

- [2] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 13. Jan. 2013, S. 19–22.
- [3] Donald Wolfe. *3-Tier Architecture in ASP.NET with C sharp tutorial*. SitePros2000.com, 13. Jan. 2013.
- [4] Adam Boduch. *Flux Architecture*. Packt Publishing, 30. Jan. 2017, S. 27, 198, 312.
- [5] Ilya Gelman und Boris Dinkevich. *The Complete Redux Book*. Leanpub, 30. Jan. 2017, S. 6–7.
- [6] Adam Boduch. *Flux Architecture*. Packt Publishing, 30. Jan. 2017.

Artikel Referenzen

- [1] Thomas A Wadlow. „The Xerox Alto Computer“. In: (Sep. 1982). URL: <https://tech-insider.org/personal-computers/research/acrobat/8109-e.pdf>.

Online References

- [7] Facebook Developers. *Hacker Way: Rethinking Web App Development at Facebook*. 4. Mai 2014. URL: <https://www.youtube.com/watch?v=nYkdrAPrdcw> (besucht am 08.07.2019).