Prototyp: Umlageverfahren auf der Ethereum Blockchain

Mizel, Paul
pmizel@asure.io
Asure Stiftung
https://asure.network

Raetz, Fabian
fraetz@asure.io
Asure Stiftung
https://asure.network

11. November 2019

Zusammenfassung

Die Asure Stiftung hat sich das Ziel gesetzt, die Sozialversicherungssysteme dieser Welt mit Hilfe der neusten Technologien wie Blockchain zu erforschen und zu verbessern. Um die Herausforderungen und die Anforderungen zur Modernisierung und Entwicklung von umlagebasierten Systemen besser verstehen zu können, wurde der Prototyp des deutschen Rentensystems auf der öffentlichen Ethereum Blockchain entwickelt. In diesem Dokument stellen wir das Ergebnis sowie die im Rahmen der Entwicklung gewonnenen Erkenntnisse vor.

Schlüsselwörter — Blockchain, Ethereum, Sozialversicherung, Rentensystem, Deutsche Rentenversicherung, Umlageverfahren

1 Einführung

Bitcoin ist das weltweit führende digitale Zahlungsmittel auf Basis eines dezentral organisierten Buchungssystems [7]. Die von Bitcoin verwendete Blockchain-Technologie bietet Vorteile im Bereich der Dezentralisierung, der Manipulationssicherheit und der Ausfallsicherheit. Diverse Branchen sind dabei und prüfen, wie sich IT-Systeme und Prozesse, durch die Verwendung der Blockchain-Technologie, optimieren lassen.

Die Asure Stiftung forscht, wie mit Hilfe der Blockchain-Technologie Sozialversicherungssysteme verbessert werden können. Hierzu wurde in Kooperation mit der ECHTNICE GmbH ein Prototyp des deutschen Rentensystems entwickelt. Die Ergebnisse dieser Arbeit werden im folgenden dargestellt.

1.1 Ziele

Ziel der Entwicklung des Prototypen zum deutschen Rentensystem ist es, die Blockchain-Technologie als Infrastrukturkomponente zu evaluieren und aufzuzeigen, wie sich diese z.B. von einer IBM Mainframe oder Java-Enterprise basierten Infrastruktur unterscheidet. Des weiteren sollen gewonnene Erkenntnisse in der Umsetzung von komplexen Softwaresystemen als Smart Contract System dokumentiert werden. Hierzu gehören u.a.:

1. Das Einspielen von Smart Contract Änderungen (z.B. Fehlerkorekturen, funktionale Erweiterungen).
2. Zugriff auf Drittsysteme aus dem Smart Contract System heraus.
3. Verwendung von Entwicklungswerkzeugen und Bibliotheken.
4. Umgang mit verschiedenen Infrastruktumgebungen für z.B. die lokale Entwicklung, Test und Produktion.
5. Entwicklung von modernen Oberflächen, welche Smart Contract Systeme verwenden.

Für die Entwicklung des Prototypen wird die Ethereum[1] Blockchain verwendet. Ethereum ist eine der bekanntesten Blockchain-Projekte, welche die Ausführung von Smart Contracts unterstützt.

1.2 Abgrenzung

Im folgenden betrachten wir Anforderungen von Rentensystemen, die im Rahmen dieses Projektes nicht weiter analysiert werden.

[1] `https://www.ethereum.org`

1.2.1 GDPR und der Datenschutz

Anwendungen, welche auf einer Blockchain betrieben werden und personenbezogene Daten verarbeiten, sind aus den folgenden Gründen problematisch:

> **Die Anonymisierung personenbezogener Daten.** Es gibt intensive Diskussionen und derzeit keinen Konsens darüber, was es braucht, um personenbezogene Daten zu anonymisieren, bis zu dem Punkt, an dem die resultierenden Ergebnisse möglicherweise in einem Blockchain-Netzwerk gespeichert werden können. Um ein Beispiel zu nennen: Das Hashing von Daten kann in vielen Situationen nicht als Anonymisierungstechnik betrachtet werden, aber es gibt Fälle, in denen die Verwendung von Hashing zur Erzeugung einzigartiger digitaler Signaturen von Daten, die außerhalb der Kette gespeichert sind, auf einer Blockkette denkbar ist.
>
> **Die Ausübung einiger Rechte der betroffenen Personen.** Wir weisen darauf hin, dass es bei der Erfassung personenbezogener Daten in einem Blockchain-Netzwerk schwierig sein kann, diese zu korrigieren oder zu entfernen. Die Definition, was als Löschung im Rahmen von Blockketten angesehen werden kann, wird derzeit diskutiert. [13]

Die GDPR konforme Implementierung des deutschen Rentensystems auf der öffentlichen Ethereum-Blockchain ist explizit kein Ziel dieser Arbeit.

Ob und wie die GDPR konforme Implementierungen des deutschen Rentensystems auf der öffentlichen Ethereum-Blockchain implementiert werden können, soll in einem zukünftigen Projekt erforscht werden. Vielversprechend scheint die Verwendung von sogenannten Zero-Knowledge-Beweisen[2] und die Überarbeitung bestehender Regelungen in Hinblick auf die Datensparsamkeit.

1.2.2 Blockchain-Skalierung

Die öffentliche Ethereum Blockchain kann 7-15 Transaktionen pro Sekunde (TPS) verarbeiten [12]. Pro Monat stehen demnach ca. 18-39 Millionen Transaktionen zur Verfügung. Auf der öffentlichen Ethereum Blockchain laufen diverse Anwendungen verschiedener Hersteller die sich dementsprechend auch gegenseitig beeinflußen. Benötigt eine Anwendung besonders viele Transaktionen, stehen diese dementsprechend nicht mehr für andere Anwendungen zur Verfügung [1].

2018 umfasste die deutsche Rentenversicherung ca. 37,5 Millionen aktiv Versicherte und ca. 21,04 Millionen Rentner. [9, 10] Für ein Blockchain-basiertes Rentensystem, welches für jede Ein-, und Auszahlung genau eine Transaktion benötigt, würden so ca. 58,04 Millionen Transaktionen pro Monat erforderlich sein.

Dies ist nur eine simple Betrachtung und je nach Implementierung kann die Anzahl der benötigten Transaktionen für ein Blockchain-basiertes Rentensystem für Deutsch-

[2] `https://de.wikipedia.org/wiki/Zero-Knowledge-Beweis`

land stark reduziert werden. Dennoch wird deutlich, dass insbesondere die öffentliche Ethereum Blockchain nicht über die entsprechenden Kapazitäten verfügt.

Das Problem der Blockchain-Skalierung ist ein bekanntes Problem zu welchem diverse Lösungsvorschläge erforscht und erarbeitet werden [8]. Im Rahmen dieser Arbeit wird das Thema Blockchain-Skalierung nicht weiter betrachtet.

1.2.3 Alternative Blockchain-Projekte

Im Rahmen dieses Projektes haben wir uns ausschließlich auf die öffentliche Ethereum Blockchain und die dazugehörige Programmiersprache Solidity fokussiert.

Neben der öffentlichen Ethereum Blockchain, kann Ethereum auch in privaten oder konsortium basierten Szenarien verwendet werden. Der Einsatz in privaten oder konsortium basierten Szenarien kann sich u.U. stark von dem öffentlichen Szenario hinsichtlich Skalierung, Performance und dem Umgang mit personenbezogenen Daten unterscheiden und wurde explizit nicht betrachtet.

Auch alternative Blockchain-Projekte wie z.B. Hyperledger Fabric[3], Qtum[4] oder EOS[5] werden im Rahmen dieser Arbeit nicht betrachtet.

2 Implementierung

Im folgenden Kapitel betrachten wir die Implementierung des Prototypen.

2.1 OpenSource

Der Quellcode des entwickelten Prototypen ist auf GitHub[6] veröffentlicht und kann unter der ISC Lizenz[7] im Rahmen weiterer Softwareprojekte genutzt werden.

Der Prototyp steht unter der Adresse `https://dapp.asure.io/` zum testen zur Verfügung.

2.2 Anwendungsfälle

Das deutsche gesetzliche Rentensystem basiert im Kern auf der Rentenformel, durch welche die Höhe der Rentenzahlung jedes Rentenempfänger ermittelt wird. Durch eine

[3] `https://www.hyperledger.org/projects/fabric`
[4] `https://qtum.org/en`
[5] `https://eos.io/`
[6] `https://github.com/AsureNetwork/asure-dapp`
[7] `https://github.com/AsureNetwork/asure-dapp/blob/master/LICENSE`

Vielzahl von Sonderregelungen gestaltet sich das Rentensystem jedoch sehr komplex, weshalb eine vollumfängliche Abbildung des Systems im Rahmen eines Prototypen nicht sinnvoll ist. Stattdessen wurden einige Anwendungsfälle ausgewählt, und im Umfang an die Anforderungen eines Prototypen angepasst.

Alle Anwendungsfälle zusammen, bilder den kompletten Zyklus eines Rentensystems ab - Von der Zahlung der Beiträge, bis zur Zahlung der Renten.

Die Anwendungsfälle sind wie folgt definiert:

1. Als Benutzer möchte ich mich als Versicherter registrieren.
2. Als Beitragszahler möchte ich mein geplantes Renteneintrittsdatum festlegen.
3. Als Beitragszahler möchte ich jeden Monat in das Rentensystem einzahlen.
4. Als Rentner möchte ich jeden Monat eine Rente auszahlen.
5. Als Versicherter möchte ich eine Übersicht über meine Beiträge / Rentenzahlungen angezeigt bekommen.
6. Als Versicherter möchte ich alle Aktionen via Smartphone Anwendung durchführen.
7. Als Administrator möchte ich das Backend (Smart Contracts) auf dem öffentlichen Ethereum Testnet bereitstellen.

2.3 Deutsche Rentenformel

Für die Berechnung der Höhe der Renten wurde die deutsche Rentenformel (Siehe 2.3) verwendet.
Dabei wurde im Rahmen des Prototypen nur die Berechnung der Entgeltpunkte (EP) implementiert.
Der Wert für den Zugangsfaktor (ZF) als auch für den Rentenartfaktor (RAF) ist immer 1. Hieraus resultiert, das die berechneten Renten immer einer Rente wegen Alters und regulären Rentenbeginn entsprechen.
Auch der Wert für den aktuelle Rentenwert (aRW) ist statisch und beträgt immer 29 €.
Der entsprechende Quellcode für die Ermittlung des aktuellen Rentenwerts (aRW) ist so strukturiert, das dieser pro Jahr im System mittels einer Lookup Tabelle hinterlegt werden kann.

$$Rente_{mtl} = EP \cdot ZF \cdot RAF \cdot aRW$$

- **$\text{Rente}_{\text{mtl}}$** ist die monatliche Bruttorente in Euro
- **EP** ist die Summe der Entgeltpunkte aufgrund des Versicherungsverlaufs
- **ZF** ist der Zugangsfaktor
- **RAF** ist der Rentenartfaktor
- **aRW** ist der aktuelle Rentenwert in Euro

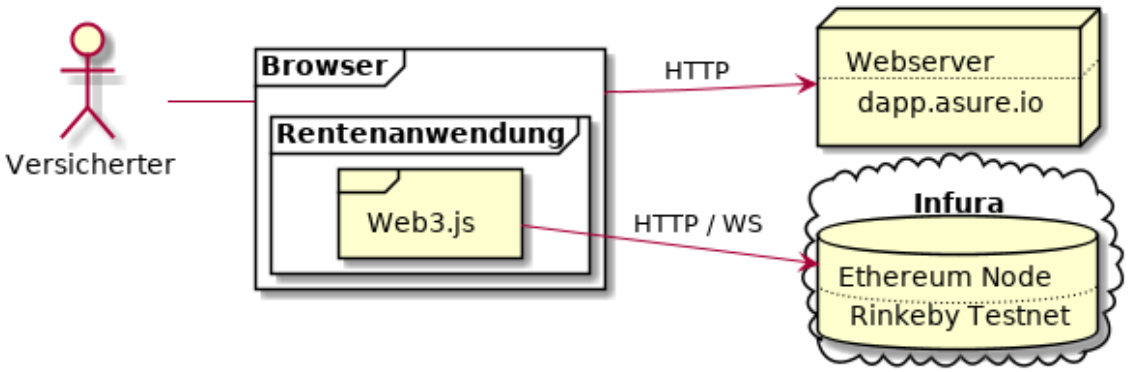

Abbildung 1: Komponenten des Prototyps

2.4 Anwendungsarchitektur

Die Benutzeroberfläche des Prototypen wurde als Progressive Web App (PWA) entwickelt und für die Verwendung auf Mobilgeräten optimiert. Ein Webserver liefert die Webanwendung mittels HTTP/HTTPS an den Browser.
Das Backend bildet ein Smart Contract-System, welches auf dem öffentlichen Ethereum Testnetz Rinkeby gehostet wird und mit dem die Benutzeroberfläche mittels HTTP/WebSockets kommuniziert. Für den Zugriff auf ein Ethereum Netz wird ein Ethereum Knoten (ggf. auch ein Ethereum Cluster für Ausfallsicherheit) benötigt. Im Rahmen des Prototypen verwenden wir hierzu Hosting Provider wie Infura und Cloudflare, welche entsprechende Ethereum Knoten hosten und zur Verfügung stellen.[8]

Smart Contract Entwicklungswerkzeuge

Im Rahmen der Backend Entwicklung wurde ein Smart Contract System für die Ethereum Blockchain entwickelt. Für die Entwicklung kamen die folgenden Tools zum Einsatz:

Solidity ist eine objektorientierte, anwendungsspezifische höhere Programmiersprache mit einer JavaScript-ähnlichen Syntax zum Entwickeln von Smart Contracts für Blockchain-Plattformen wie Ethereum oder Tron.
Sämtliche Smart Contracts des Prototypen wurden in Solidity programmiert.

OpenZeppelin Contracts ist ein Framework aus modularen, wiederverwendbaren und sicheren Smart Contracts für das Ethereum-Netzwerk, geschrieben in Solidity.

[8] Die Dezentralisierung und somit das Betreiben eigener Ethereum Knoten ist ein entscheidener Vorteil der Blockchain-Technologie. Durch die Verwendung von gehosteten Ethereum Gateways (Infura / Cloudflare) wird dieser Vorteil nicht genutzt.

Truffle ist ein Entwicklungsframework für Ethereum. Es bietet u.a. Funktionen für das Erstellen von automatisierten Tests, Deployments und Migrationen von Smart Contracts. Zudem können mit Truffle verschiedenen Ethereum Netwerke wie z.B. ein lokales Entwicklungsnetz, oder das öffentliche Ethereum Testnetz Rinkeby verwaltet werden.

Ganache ist eine Ethereum Software, welche speziell für die Entwicklung von Smart Contracts entwickelt wurde. Ganache bietet Erweiterungen um z.B. innerhalb von Unit Tests die Uhrzeit zu Manipulieren und somit auch uhrzeitabhängige Funktionalitäten zu testen.

Infura bietet einen skalierbaren API-Zugriff auf öffentliche Ethereum-Netzwerke.

2.5 Smart Contract Architektur

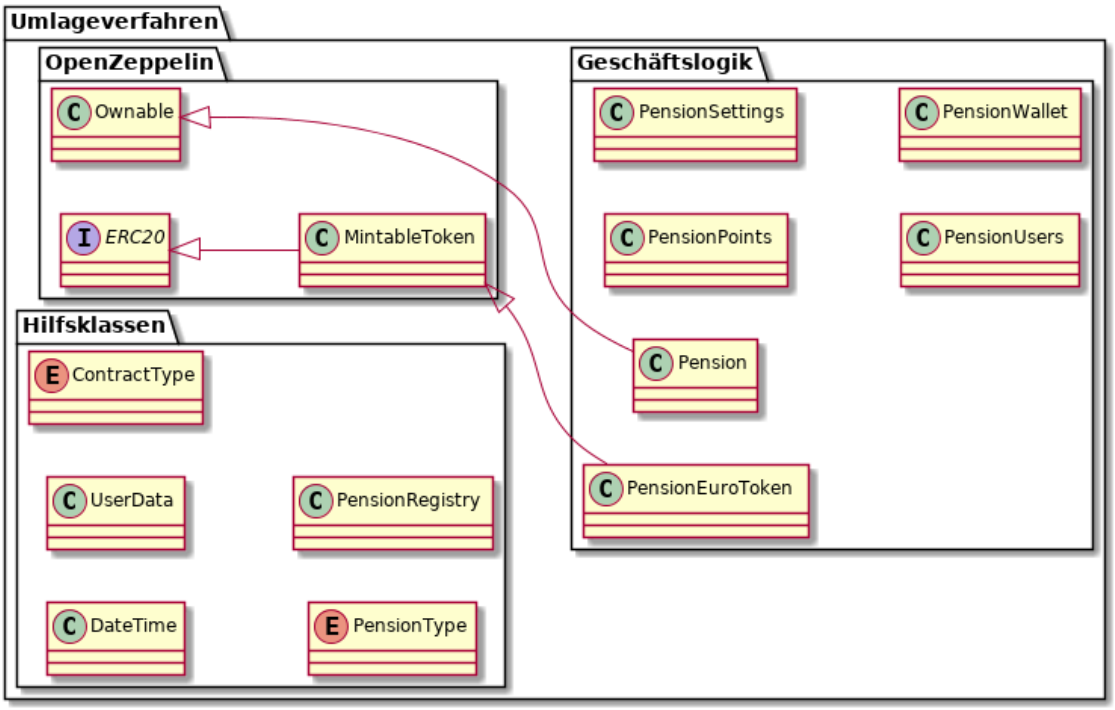

Abbildung 2: Smart Contracts Übersicht

PensionUsers: Registrierung und Verwaltung von Benutzern und deren Stammdaten.

PensionWallet: Beitrags-, und Rentenzahlungen - Verwaltet sämtliche Geldmittel.

PensionPoints: Datenhaltung Beitragszahlungen und Entgeltpunkte.

Pension: Berechnung der Höhe der Rentenzahlungen.

PensionSettings: Verwaltung diverser Parameter (z.B. Rentenartfaktor (RAF) / Rentenwerte (aRW)).

2.5.1 Anwendungsfall: Als Beitragszahler möchte ich jeden Monat in das Rentensystem einzahlen

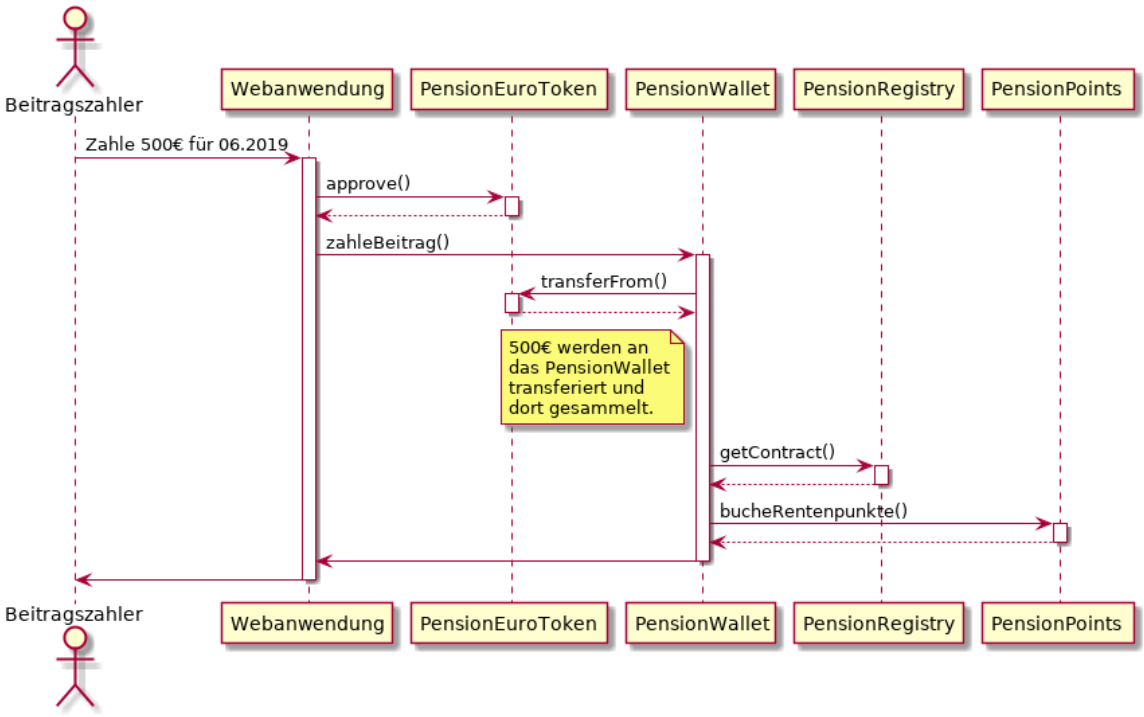

Abbildung 3: Sequenzdiagramm: Beitragszahlung

2.5.2 Anwendungsfall: Als Rentner möchte ich jeden Monat eine Rente auszahlen

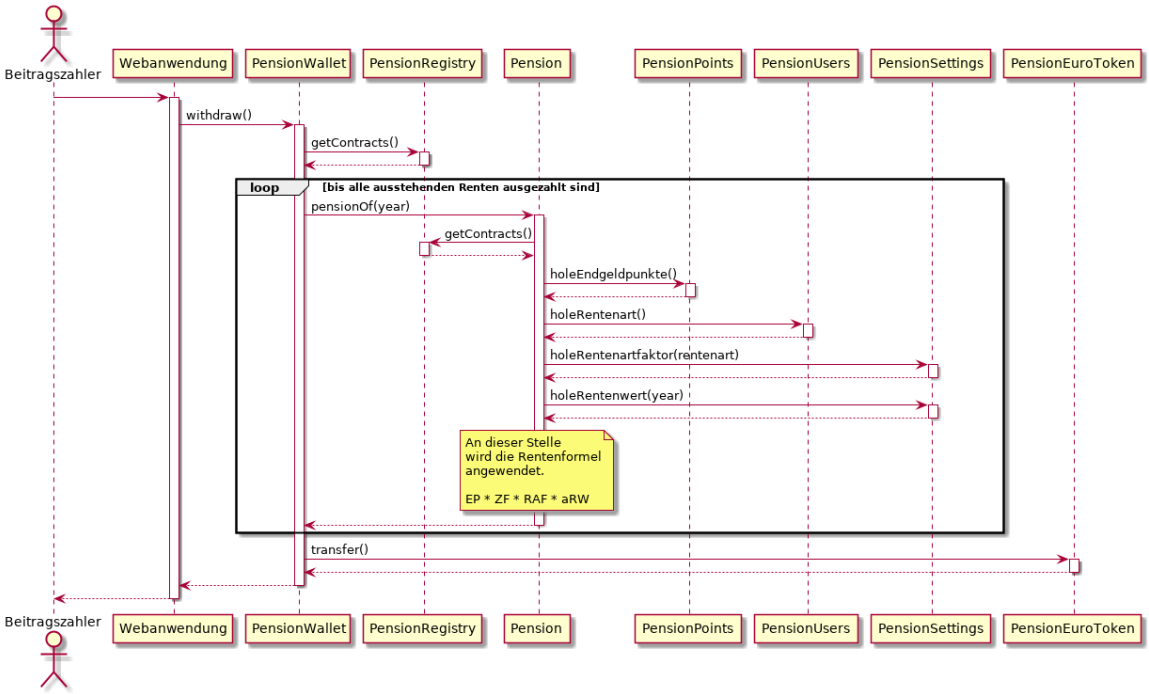

Abbildung 4: Sequenzdiagramm: Rentenzahlung

3 Gewonnenen Erkenntnisse

3.1 Mathematische Operationen mittels SafeMath

Arithmetische Operationen in Solidity werden bei einem Überlauf und Unterlauf umgebrochen. Dies kann leicht zu Fehlern führen, da Programmierer normalerweise davon ausgehen, dass ein Überlauf einen Fehler auslöst, was das Standardverhalten in höheren Programmiersprachen ist.

Das folgende Listing 1 demonstriert das Problem des Überlauf und Unterlauf einer Integer Variable.

```
contract Test {
  function overflow() returns (uint8) {
```

```
    uint8 a = 255;
    a = a + 1;
    return a; // a = 0
  }

  function underflow() returns (uint8) {
    uint8 a = 0;
    a = a   1;
    return a; // a = 255
  }
}
```

Listing 1: Beispielhafter Überlauf und Unterlauf

Lösungsansatz

Durch das explizite prüfen können Überlaufe und Unterläufe erkannt werden und die Transaktion zurückgesetzt werden. Das OpenZeppelin-Framework stellt eine solche Prüfung mittels der SafeMath API zur Verfügung. SafeMath stellt diese Intuition wieder her, indem die Transaktion zurückgesetzt wird, wenn ein Vorgang überläuft. [3]

```
contract Test {
 using SafeMath for uint8;

 function multiplySafe(uint8 a) returns (uint8) {
   return a.mul(a); // revert() transaction on overflow / underflow
 }
}
```

Listing 2: Beispielhafte Multiplikation mittels SafeMath zurück

3.2 Gleitkommaoperationen

Die Programmiersprache Solidity unterstützt Gleitkommaoperationen nach IEEE 754 nicht. Es gibt jedoch teilunterstützung für Bruchwerte die bestimmten Einschränkungen unterworfen sind. [6]

Lösungsansätze

Anstelle von Gleitkommaoperationen, können alternativ entsprechend große Ganzzahlen verwendet werden. Als Beispiel dient z.B. die native Kryptowährung ETH der Ethereum Blockchain. Ein ETH ist eine 19-stellige Ganzzahl. Die kleinste Einheit ist

1 WEI. Ein ETH ist gleich 1.000.000.000.000.000.000 WEI. Durch diese Darstellung können auch Bruchteile eines ETH transferiert werden.

Der Rentenartfaktor (RAF) der Rentenformel ist als Gleitkommazahl definiert und kann Werte von 0 bis 1 annehmen. [5]

Gleitkommazahlen können auch in diesem Fall durch entsprechend große Ganzzahlen repräsentiert werden, indem im SmartContract immer mit dem Faktor 1000 multipliziert wird. Externe Systeme wie z.B. eine Webanwendung können die Höhe der monatlichen Rente errechnen, indem der Wert entsprechend durch den Devisor 1000 geteilt wird.

```
contract Rentenformel {
 function rechneBeispiel() returns (uint256) {
   uint256 ep = 40;
   uint256 zf = 1;
   uint256 raf = 250; // kleine Witwenrente = 0.25 * 1000
   uint256 arw = 30;

   uint256 mtlRente = ep * zf * raf * arw; // mtlRente = 300000;

   return mtlRente;
   // Extern z.B. in Webanwendung: mtlRente / 1000 = 300
 }
}
```

Listing 3: Verzicht von Gleitkommazahlen durch Multiplikation

Alternativ kann auf Bibliotheken wie z.B. DS-Math [9] zurückgegriffen werden, welche entsprechende Funktionalitäten bieten

3.3 Zeitkomplexität von Smart Contract Methoden

Jede Ethereum-Transaktion (somit auch jede Ausführung einer Smart Contract Methode) verbraucht "Gas". Pro Transaktion steht nur eine bestimmte Menge an Gas zur Verfügung. Somit sind die Ausführungszeiten und die Aufgaben, welche inerhalb einer Smart Contract Methode ausgeführt werden können, ebenfalls begrenzt. Verbraucht eine Ethereum-Transaktion zu viel Gas, wird die Transaktion abgewiesen und nicht in die Blockchain mit aufgenommen.

[9] `https://github.com/dapphub/ds-math`

Lösungsansatz

Die Zeitkomplexität[10] von SmartContract Methoden sollte immer konstant sein (O(1)). Zeitkomplexität kann reduziert werden, in dem Daten vorberechnet und aggregiert werden.

Statt wie in Listing 4 über eine Liste aller Beitragszahlungen zu iterieren, kann die Gesamtsumme aller Beitragszahlungen schon in zahleBeitrag aggregiert werden, sodass die Zeitkomplexität immer O(1) ist.

```
contract Rente {
 uint256[] beitragszahlungen;

 function zahleBeitrag(uint256 beitrag) public {
        beitragszahlungen.push(beitrag);
 }

 function summeBeitragszahlungen() public returns (uint256) {
        uint256 summe = 0;
        for (var i = 0; i < beitragszahlungen.length; i++) {
      summe += beitragszahlungen[i];
        }
        return summe;
 }
}
```

Listing 4: Schlecht - Zeitkomplexität von summeBeitragszahlungen() ist O(n)

```
contract Rente {
 uint256[] beitragszahlungen;
 uint256 summe;

 function zahleBeitrag(uint256 beitrag) public {
        beitragszahlungen.push(beitrag);
        summe += beitrag;
 }

 function summeBeitragszahlungen() public returns (uint256) {
        return summe;
 }
}
```

Listing 5: Gut - Zeitkomplexität von summeBeitragszahlungen() ist O(1)

[10] http://www.inf.fu-berlin.de/lehre/SS12/ALP2/slides/V6_Rekursion_vs_Iteration_ALP2.pdf

3.4 Weitere Beschränkungen bzgl. Smart Contract Programmierung

Solidity unterstüzt nur statische Zeichenketten. Dynamische Operationen wie das Zusammensetzen von Zeichenketten ist somit ohne weiteres nicht möglich.

Die größe von Smart Contract Quellcode, welche mit einer Transaktion veröffentlicht werden kann, ist beschränkt. Ggf. müssen Zusammenhängende Smart Contract Systeme auf mehrerer Transaktionen aufgeteilt und veröffentlicht werden.

Das Hinterlegen von Smart Contract Quellcode in Blockchain Explorer (Etherscan.io) ist noch nicht in Truffle integriert und muss händisch durchgeführt werden.

Codeänderungen (Bugfixes und Features)

Bereitgestellter Smart Contracts Code ist unveränderbar auf der Blockchain gespeichert Änderungen müssen im Zuge von Fehlerbehebungen und Anforderungsänderungen durchgeführt werden können.

Lösungsansatz: Upgradeability Proxy Muster

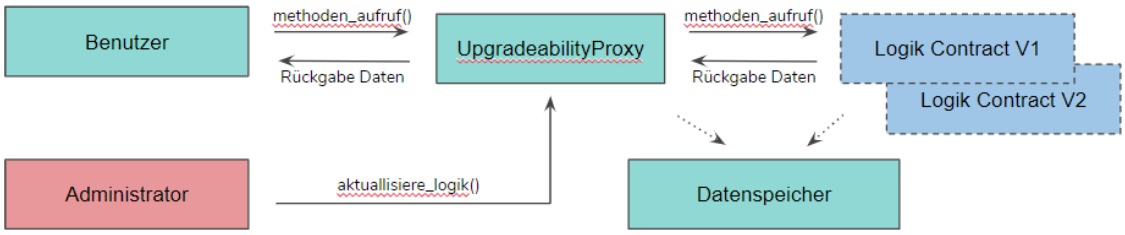

Abbildung 5: Smart Contract Upgrade Pattern

Abfragen von Blockchain Daten

Ethereum Smart Contracts bieten nur eine primitive Datenhaltung und rudimentäre Datenabfrage durch Drittanwendungen. Benutzer Masken benötigen viele HTTP-Anfragen um die benötigten Daten aus Ethereum in die Anwendung zu laden.

Eine weitere Herausforderung hinsichtlich der Benutzerführung sind die langen Ausführungszeiten von Transaktionen. Im Durchschnitt wird alle 15 Sekunden ein neuer Ethereum Block in die Blockkette aufgenommen. Dies hat zur Folge, das Transaktionen entsprechend 15 Sekunden oder länger benötigen bis Sie in die Blockchain aufgenommen werden.

Ladeanimationen über einen Zeitraum von mehreren Sekunden sind aus Benutzerperspektive eher suboptimal.

Lösungsansatz

Auf das direkte Abfragen von Blockchain-Daten mittels RPC/HTTP-Schnittstelle verzichten. Stattdessen kann z.B. eine besser geeignete Abfragesprache wie z.B. GraphQL[11] verwendet werden. Die Ethereum Software Geth unterstützt die Abfragesprache GraphQL z.B. seit der Version 1.9.0 nativ [2].

Alternativ gibt es spezialisierte Abfrage Proxies z.B. mittels GraphQL wie EthQL[12] und TheGraph[13].

Durch die Verwendung des Anwendungsmusters Optimistic UI geht die Oberfläche davon aus, dass die entsprechende Schreiboperation erfolgreich war, bevor die Antwort bekannt ist. [4] Hierdurch entfällt das Anzeigen einer Ladeanimation. Für den Fall, dass die Operation nicht erfolgreich war, muss der Nutzer entsprechend informiert werden.

Eine weitere sinnvolle Lösung kann darin bestehen, Daten niemals direkt von der Blockchain zu lesen, sondern stattdessen eine für Lesezugriffe optimierte Datenbasis aufzubauen. Dieses Vorgehen ist auch als Command-Query-Responsibility-Segregation (CQRS) - Entwurfsmuster bekannt [11].

Batch-Job Verarbeitung

Ethereum bietet keine native Unterstützung für "zeitgesteuerte" Transaktionen. Ethereum Smart Contracts können nicht direkt auf Smart Contract Ereignisse reagieren.

Lösungsansatz

Externe Systeme können zeitgesteuert und auf Smart Contract Events reagieren - Die Lösung setzt voraus, dass einem externen System "vertraut" wird.

Anbindung externer Datenquellen

Smart Contracts können nicht direkt auf externe Systeme zugreifen (z.B. mittels HTTP)

[11] `https://graphql.org/`
[12] `https://github.com/ConsenSys/ethql`
[13] `https://thegraph.com/`

Lösungsansatz

Als Lösung bieten sich sogenannte Orakel (Oracles) an. Diese stellen die Daten auf der Blockchain zur Verfügung und stellen auf Anfrage weitere bereit. Es gibt Oracle-Anbieter[14], die dies notwendige Infrastruktur bereitstellen und welche als Dienstleistung genutzt werden können. Entwicklung projektbezogener Orakel ist eine weitere Option.

4 Fazit

Im Rahmen dieses Projektes wurde ein Prototyp zum deutschen Rentensystem auf der öffentlichen Ethereum Blockchain entwickelt. Der Prototyp ist im Funktionsumfang stark reduziert und implementiert die Berechnung der Entgeltpunkte (Faktor EP der Rentenformel) als Smart Contract System. Der Zugriff auf das Smart Contract System erfolgt durch eine mobile Webanwendung. Durch diese können Rentenbeiträge mittels der Kryptowährung ETH gezahlt, Renten ausgezahlt und allgemeine Parameter wie z.B. der Zeitpunkt des Renteneintritts spezifiziert werden.

Diese Machbarkeitsstudie des Umlageverfahrens der deutschen Rente auf Ethereum Blockchain hat aufgezeigt, dass der Durchsatz von 7-15 Transaktionen pro Sekunde in Kombination mit anderen Anwendungsfällen für ein Rentensystem in Deutschland nicht ausreicht. Bei Erweiterungen und vollständiger Umsetzung aller Anforderungen, wäre das öffentliche Ethereum Netzwerk heute nicht in der Lage die Verarbeitung ohne großen Aufwand zu unterstützen.

Die Blockchain-Technologie ist Stand 2019 noch in diversen Bereichen limitiert. Es gilt, die Stärken der Blockchain im Kontext von Sozialversicherungen sinnvoll zu nutzen, sodass diese die Schwächen überwiegen. Grundlegende Probleme wie die Skalierung oder der richtige Umgang mit personenbezogenen Daten werden durch diverse Projekte und Institutionen erarbeitet und wir sind optimistisch, dass entsprechende Lösungen in naher Zukunft zur Verfügung stehen.

Literatur

[1] CryptoKitties craze slows down transactions on Ethereum. `https://www.bbc.com/news/technology-42237162`. 2017-12-05 Abgerufen: 2019-08-14.

[2] Geth v1.9.0. `https://blog.ethereum.org/2019/07/10/geth-v1-9-0/`. 2019-07-10 Abgerufen: 2019-08-15.

[3] OpenZeppelin Documentation - SafeMath. `https://docs.openzeppelin.com/contracts/2.x/api/math`. Abgerufen: 2019-08-14.

[4] Optimistic UIs in under 1000 words. `https://uxplanet.org/optimistic-1000-34d9eefe4c05?gi=93adffd521a8`. Abgerufen: 2019-08-15.

[14] z.B. Chainlink `https://chain.link/` und Provable `http://provable.xyz/`

[5] Rentenformel - Retenartfaktor. https://de.wikipedia.org/wiki/Rentenformel#Rentenartfaktor_(RAF). Abgerufen: 2019-08-14.

[6] Solidity Documentation - Fixed Point Numbers. https://solidity.readthedocs.io/en/v0.5.3/types.html#fixed-point-numbers. Abgerufen: 2019-08-14.

[7] Wikipedia Bitcoin. https://de.wikipedia.org/wiki/Bitcoin. Abgerufen: 2019-08-16.

[8] John Adler. The State of Layer-2 Protocol Development. https://media.consensys.net/the-state-of-ethereum-layer-2-protocol-development-2-f22b2603abd6. Abgerufen: 2019-08-15.

[9] Deutscher Rentenversicherung Bund. Rentenversicherung in Zahlen 2019. https://www.deutsche-rentenversicherung.de/SharedDocs/Downloads/DE/Statistiken-und-Berichte/statistikpublikationen/rv_in_zahlen_2019.pdf?__blob=publicationFile&v=3. Abgerufen: 2019-08-14.

[10] Deutscher Rentenversicherung Bund. Versichertenbericht 2018. https://www.deutsche-rentenversicherung.de/SharedDocs/Downloads/DE/Statistiken-und-Berichte/Berichte/versichertenbericht_2018.pdf?__blob=publicationFile&v=1. Abgerufen: 2019-08-14.

[11] Martin Fowler. CQRS Command Query Responsibility Segregation. https://martinfowler.com/bliki/CQRS.html. Abgerufen: 2019-08-16.

[12] Kieran Smith. Vitalik — Ethereum en route to a million transactions per second. https://bravenewcoin.com/insights/vitalik-ethereum-en-route-to-a-million-transactions-per-second. Abgerufen: 2019-08-14.

[13] Ken Timsit Tom Lyons, Ludovic Courcelas. Eu blockchain observatory and forum Blockchain and the GDPR. https://www.eublockchainforum.eu/sites/default/files/reports/20181016_report_gdpr.pdf. Abgerufen: 2019-08-15.

Glossar

API Die API ist eine Schnittstelle, die ein Softwaresystem bereitstellt, um dieses in andere Programme einzubinden.

Blockchain A system in which a record of transactions are maintained across several computers that are linked in a peer-to-peer network.

ETH Standard Währungseinheit im Ethereum Netzwerk.

Gas Gas ist eine Einheit der Rechenarbeit und der Kosten von Transaktionen oder Smart Contracts die im Netzwerk ausgeführt werden.

WEI Kleinste Währungseinheit im Ethereum Netzwerk.

GDPR Die Datenschutz-Grundverordnung ist eine Verordnung der Europäischen Union, mit der die Regeln zur Verarbeitung personenbezogener Daten durch die meisten Datenverarbeiter, sowohl private wie öffentliche, EU-weit vereinheitlicht werden.