



universität
wien

BACHELORARBEIT

IMPLEMENTIERUNG EINES ÖFFENTLICHEN BLOCKCHAIN SYSTEM

Verfasser

Nemanja Srdanovic

angestrebter akademischer Grad

Bachelor of Science (BSc)

Wien, 2022

Studienkennzahl lt. Studienblatt: UA 033 526

Fachrichtung: Wirtschaftsinformatik

Betreuer: Dr. techn. Belal Abu Naim

Inhaltsverzeichnis

1	Einführung	5
2	Verwandte Architekturen	8
3	System Design.....	9
3.1	Network Komponente	9
3.2	Node Komponente.....	11
3.3	Wallet Komponente.....	13
4	Systemfunktionalitäten.....	14
4.1	Übertragung und Verifikation von Transaktionen.....	14
4.2	Übertragung und Verifikation von Blöcken	16
5	Implementierung	19
6	Auswertung.....	23
7	Conclusio.....	24
8	Referenzen	25
9	Abbildungsverzeichnis	26

Danksagung

An erster Stelle möchte ich meinem Betreuer Dr. techn. Belal Abu Naim danken, der mich richtungsweisend und mit viel Engagement während meiner Arbeit begleitet hat.

Ein herzliches Dankeschön geht auch an die Kolleginnen und Kollegen, die mit mir mitgefiebert haben und mich immer gefördert haben.

Mein größter Dankt gilt vor allem meinen Eltern, meiner Schwester und meiner Freundin. Vielen Dank für die Unterstützung und den motivierenden Beistand während meines gesamten Studiums.

Nemanja Srdanovic

Wien, 31.01.2022

Abstract

Um die Technologie und die Hauptbausteine der Blockchain-Technologie zu verstehen, zielt dieses Projekt auf die Implementierung eines Blockchain-Systems ab. Das Ziel besteht darin die Datenmodelle der Blöcke zu definieren, Hashing-Algorithmen zu verwenden, eine Funktion, die den Merkle-Tree erstellt zu implementieren und den Proof-of-Work Algorithmus zu entwerfen und einzusetzen. Des Weiteren soll ermöglicht werden neuen Blöcke zu schürfen und Transaktionen auszuführen. Zusätzlich ist die Anforderung ein vereinfachtes Peer-to-Peer Protokoll zu erstellen, welches die Kommunikation zwischen Komponenten des Systems ermöglicht. Die Arbeit ist für jene Personen geeignet, die die Grundsteine der Blockchain-Technologie zu verstehen versuchen, da sie durch Modelle sowie deren Realisierung die Struktur und Verbindungen einzelner Komponenten des Systems präsentiert.

1 Einführung

Das Konzept der Peer-to-Peer-Version eines dezentralen elektronischen Zahlungssystems, welches ermöglicht Zahlungen direkt von einer Partei an eine andere zu senden, ohne dass es die Mitwirkung eines vertrauenswürdigen Finanzinstituts benötigt, wurde 2008 von Satoshi Nakamoto vorgestellt. [1]

Die Motivation dahinter ist es bestehende Modelle elektronischer Zahlungen durch ein verbessertes Model abzulösen, da bestehende Modelle unter den Schwächen des auf Vertrauen basierenden Modells leiden, wodurch es zu erhöhten Transaktionskosten, Beschränkungen der praktischen Mindesttransaktionsgröße und weiteren Komplikationen kommt. Dieses Zahlungssystem basiert auf einem kryptografischen Beweis anstatt auf Vertrauen, wodurch Parteien direkt miteinander arbeiten können und das Bedürfnis für eine dritte Partei, welche die Transaktion zwischen den Parteien abwickelt, entfällt. [2]

Diese Idee eines elektronischen Zahlungssystems löste das aus, was wir heute als Blockchain bezeichnen. Die Blockchain ist eine verteilte Datenstruktur aus Datenblöcken, in der ein Block durch seinen Hashwert mit einem anderen Block verknüpft ist [3]. Blöcke enthalten mehrere Transaktionen, einen Zeitstempel, den Hashwert des vorherigen Blocks und ein Nonce, bei der es sich um eine Zufallszahl zur Überprüfung des Hashs handelt, außerdem können Blöcke vom Netzwerk mit kryptografischen Mitteln validiert werden. Durch dieses Konzept wird die Integrität der gesamten Blockchain sichergestellt, da Hashwerte einzigartig sind und die Änderung eines Blocks in der Kette auch den Hashwert dieses Blocks ändern würde, wodurch die ganze Blockchain invalidiert wäre. [4] [5]

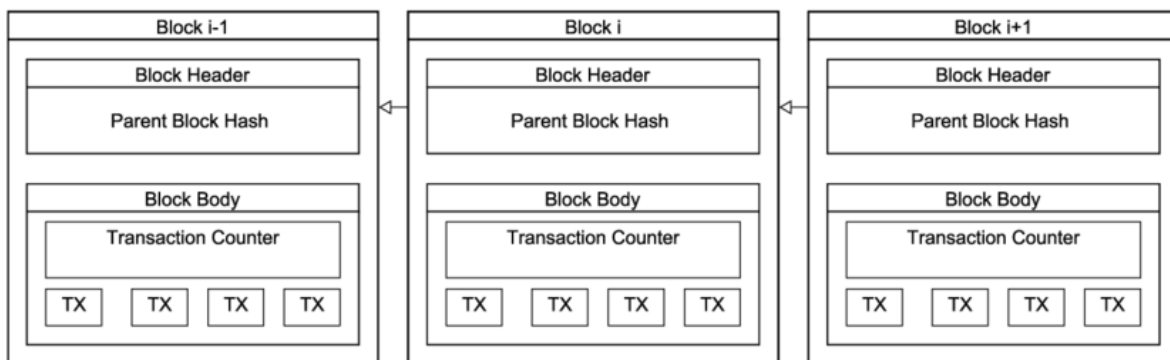


Abbildung 1: Sequence-of-blocks-in-a-blockchain-Zheng-et-al-2017

Ein Hashwert wird mithilfe einer kryptologischer Hashfunktionen erzeugt. Dabei werden beliebig lange Datensätze an die Hashfunktion übergeben, welche einen Ausgabewert liefert, der immer die gleiche Größe hat. Würde man nur ein Bit in diesem Datensatz ändern, bekäme man einen komplett anderen Hashwert, was bedeutet, dass keinen Rückschluss auf den Inhalt und die Länge des übergebenen Datensatz möglich ist, wodurch sich Hashes bestens dafür eignen, die Integrität von Daten zu garantieren. [6]

Um die Validität des Blockchains zu erhalten, halten sich die Knoten an einen Konsensmechanismus, der besagt, dass sich die Mehrheit der Knoten im Netzwerk auf die Gültigkeit der Transaktionen und des Blocks selbst einigen muss, damit der Block in die Blockchain hinzugefügt werden kann.

Das öffentliche Blockchain-System enthält eine Vielzahl an Konsensmechanismen, die sich mit der Zeit etabliert haben, wobei die bekanntesten der Proof-of-Work (PoW) und Proof-of-Stake (PoS) sind. Der PoW Mechanismus erfordert das Knoten gegeneinander antreten und eine bestimmte Menge an Rechenaufwand aufwenden, um eine komplizierte kryptografische Aufgabe zu lösen. Der Knoten welcher es als erster schafft und den Hash für den neuen Block berechnet, bekommt die Möglichkeit den neuen Block zur Blockchain hinzuzufügen [7]. Beim PoS Konsensalgorithmus handelt es sich um eine Alternative mit geringem Energieverbrauch im Vergleich zum PoW. Damit man das Recht bekommt einen neuen Block in die Blockchain hinzuzufügen muss man seine Tokens einem Wallet Zuweisen (Staking). Dabei ist die Chance das Recht den neuen Block hinzuzufügen proportional groß mit der Anzahl der zugewiesenen Tokens. [8]

Zusätzliche Verifikation der Datenstruktur wird durch das Konzept des Merkle Tree ermöglicht, da dadurch Blockchain-Daten effizienter und sicherer verschlüsselt werden können.

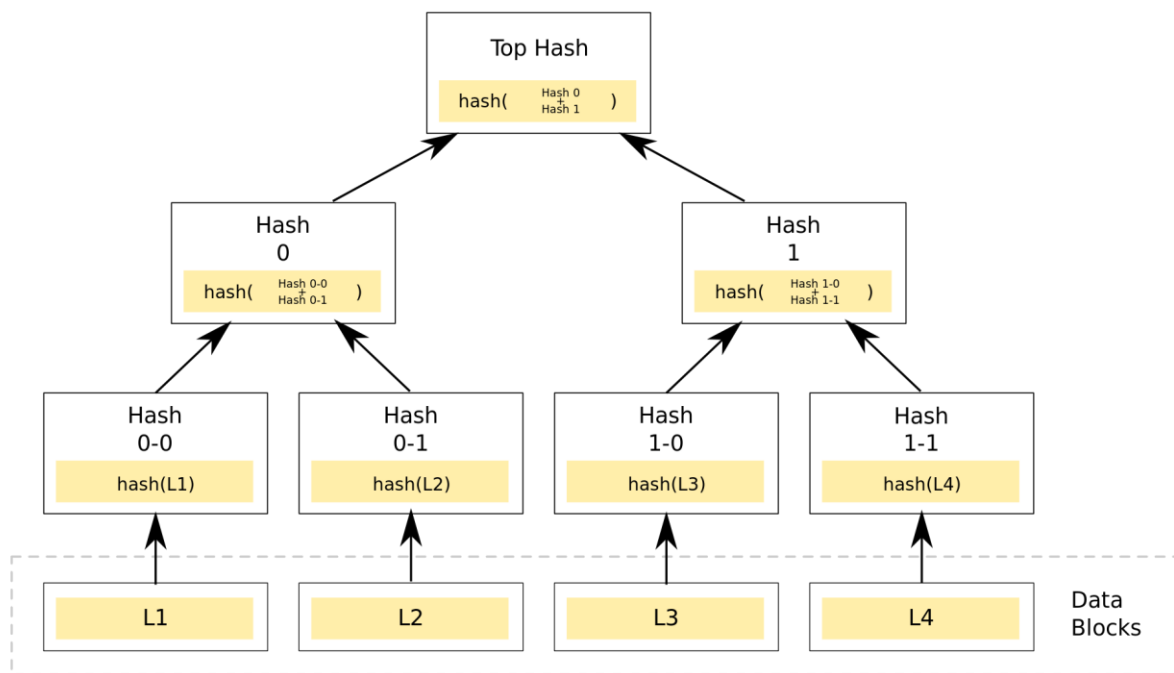


Abbildung 2: Hash_Tree Original illustration by David Göthberg, Sweden

Im Merkle Tree werden die Hashes in der unteren Reihe als Blätter bezeichnet, die übergeordneten Zwischen-Hashes als Zweige und der oberste Hash als Wurzel (Merkel Root). Um zum Merkle Root zu kommen, werden beginnend von der untersten Ebene Paare von Blättern und Zweigen miteinander verkettet und gehasht, um übergeordnete Zweige zu erhalten. Dieser Prozess wird so lange weitergeführt bis nur noch ein Paar von Zweigen übrigbleibt, der Verkettet und gehasht wird, um den Merkle Root zu erhalten. Sollte die unterste Ebene eine ungerade Anzahl an Blättern erhalten, wird ein Blatt verdoppelt und sein Hash mit sich selbst verkettet. Da die Hashes für jedes Blatt einzigartig sind würden sich beim Ändern eines einzigen Blattes auch die Hashes der übergeordneten Zweige

ändern was zur Invalidierung des Merkle Root führen würde, was das wichtigste Sicherheitsmerkmal dieses Konzepts darstellt. [9]

Merkle Trees werden zum Beispiel in der Blockchain von Bitcoin eingesetzt, um einen Transaktionsblock mit diesem Konzept zu sichern und ein Hash zu generieren, mit dem die Richtigkeit der Transaktionsliste bestätigt werden kann. Dafür wird jede Transaktion gehasht und anschließend wird jedes Transaktionspaar verkettet und miteinander gehasht, bis es nur einen Hash für den gesamten Block gibt, der zur Sicherung der Transaktionsliste eingesetzt werden kann. [10]

In einem Blockchain System sind Knoten über ein Peer-to-Peer Netzwerk verbunden, in dem alle Daten repliziert, geteilt und synchron zwischen den Netzwerken verteilt werden. Jedoch sind nicht alle Netzwerke, auf denen die Blockchain Plattformen betrieben werden gleich, sondern unterscheiden sich hinsichtlich ihrer Konfigurationen. Die Peer-to-Peer Netzwerke können Hinsichtlich ihrer Berechtigungen und ihrer Netzwerkzugänglichkeit kategorisiert werden. [11]

Die Berechtigungen reichen von Erlaubnis wo jeder schreiben, lesen und am Konsensmechanismus teilnehmen kann bis zu nur einer dieser Operationen. Aus der Perspektive der Netzwerkzugänglichkeit unterscheidet man zwischen öffentlichen und privaten Blockchains sowie einigen Hybriden dieser zwei Netzwerkarten. Öffentliche Blockchains haben einen offenen Netzwerkzugriff, wodurch jeder dem Netzwerk beitreten darf. Im Gegensatz dazu gehören private Blockchains einer Organisation und der Zugriff auf sie ist beschränkt. [12]

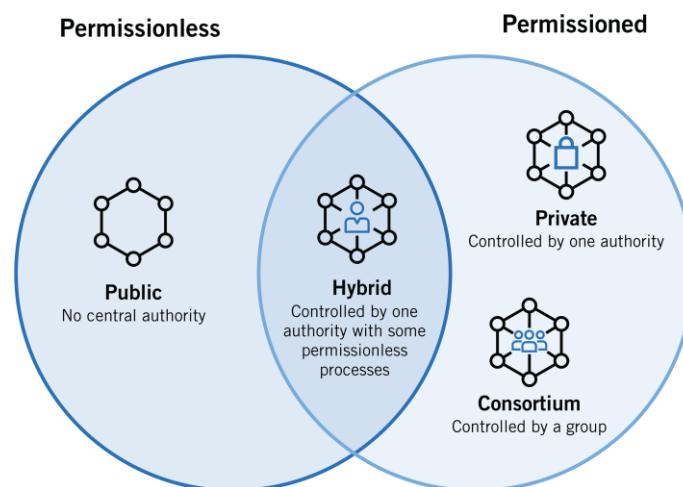


Abbildung 3: Kathleen E. Wegrzyn Eugenia Wang, Types of Blockchain - Public, Private, or Something in Between

Um die Blockchain Technologie besser zu verstehen, wurden die in diesem Kapitel kurz erwähnten Konzepte, Modelle und Mechanismen detaillierter analysiert und vereinfachte Versionen dieser Komponenten mit der Java Programmiersprache implementiert. Dabei wurde wie in den folgenden Kapiteln zu sehen ist, als erstes ein Konzept der zukünftigen Implementierung erstellt welcher UML Diagramme enthält, die der Visualisierung von Software-Teilen und anderen Systemen dienen sollen, um so die tatsächliche Ausführung zu erleichtern.

2 Verwandte Architekturen

Das Blockchain Ökosystem ist seit dem Bitcoin-Whitepaper im Jahr 2008 immens gewachsen. Die Anzahl der aktiven Blockchains wächst täglich in immer schnellerem Tempo. Zum Zeitpunkt des Verfassens dieser Arbeit gibt es mehr als 10.000 aktive Blockchains die sich auf Kryptowährungen basieren und mehrere hundert Blockchains anderer Art. [13]

Einige Frameworks und Systeme implementieren die öffentliche Blockchain, welche wir in dieser Arbeit näher betrachten werden. Zu den in der Branche am meisten Anerkannten und Akzeptierten gehören vor allem Bitcoin und Ethereum. Beide Blockchains benutzen zurzeit den PoW Konsensmechanismus, um eine Einigung über den Status des Netzwerks zu erzielen und neue Blocks für die Blockchain zu kreieren [14].

Die Art in welcher neue Blöcke erzeugt werden unterscheidet sich jedoch um einiges, weswegen das Bitcoin-Blockchain-System architektonisch am kompatibelsten mit dem in dieser Arbeit zur implementierten Blockchain-System ist. Daher wurden dessen Architektur, Konzepte und Mechanismen als Referenz für diese Arbeit genommen.

Bitcoin ist ein Online-Kommunikationsprotoll, welches die Verwendung einer virtuellen Währung einschließlich elektronischer Zahlungen erleichtern soll [15]. Damit doppelte Ausgaben vermieden werden, wird die Transaktion mit einem privaten Schlüssel signiert und bekommt einen Zeitstempel. Die Transaktionen werden in Blocks gepackt die mit der SHA-256 Funktion so lange gehasht wird, bis ein angemessener CPU-Aufwand aufgebraucht ist. Da spätere Blöcke mit diesen Hashs verkettet werden, würde die Arbeit zum Ändern des Blocks das Wiederholen aller nachfolgenden Blöcke umfassen [1]. Das Netzwerk ist so aufgebaut, dass neue Transaktionen und Blöcke nicht notwendigerweise alle Knoten erreichen müssen. Transaktionen werden trotzdem in den nächsten Block geraten und Knoten können beim Empfang des nächsten Blocks erkennen, dass ein Block verpasst wurde und ihn anfordern. [16]

Die Schritte zum Ausführen des Netzwerks lauten wie folgt [1]:

- Neue Transaktionen werden an alle Knoten gesendet.
- Jeder Knoten sammelt neue Transaktionen in einem Block.
- Jeder Knoten arbeitet daran, einen PoW für seinen Block zu finden.
- Wenn ein Knoten den PoW findet, sendet er den Block an alle Knoten.
- Knoten akzeptieren den Block nur dann, wenn alle darin enthaltenen Transaktionen gültig und nicht bereits verbraucht sind.
- Knoten drücken ihre Akzeptanz des Blocks aus, indem sie an der Erstellung des nächsten Blocks in der Kette arbeiten und den Hash des akzeptierten Blocks als den vorherigen Hash verwenden.

Alle angeführten Schritte sind daher auch im implementierten öffentlichen Blockchain-System vorhanden. Lediglich wurde die Akzeptanz des Blocks um einiges vereinfacht und ein Vergleich der Zeitstempel eingeführt, sodass Schürfen eines neuen Blocks um einiges schneller durchgeführt werden kann.

3 System Design

Die folgenden Diagramme zeigen die Struktur des implementierten Blockchain-Systems, indem sie die Klassen des Systems, ihre Attribute, Operationen und Beziehungen zwischen Objekten darstellen. Das System besteht aus drei Komponenten, der Netzwerk Komponente, welche die Kommunikation zwischen den Bestandteilen des Systems ermöglicht, der Node Komponente, die die Logik des Blockchain Konzepts enthält, sowie alle Funktionen und Sicherheitsmechanismen inkorporiert und der Wallet Komponente durch die Transaktionen signiert und in das Netzwerk gesendet werden können sowie der Zustand der Blockchain grafisch manifestiert werden kann.

3.1 Network Komponente

Die Netzwerk-Komponente repräsentiert die Implementierung eines P2P Protokolls, welches eine vereinfachte P2P Netzwerkfunktionalität bereitstellt, die von allen Knoten verwendet wird. Folgende Funktionalitäten werden vom Protokolldesign bereitgestellt [17]:

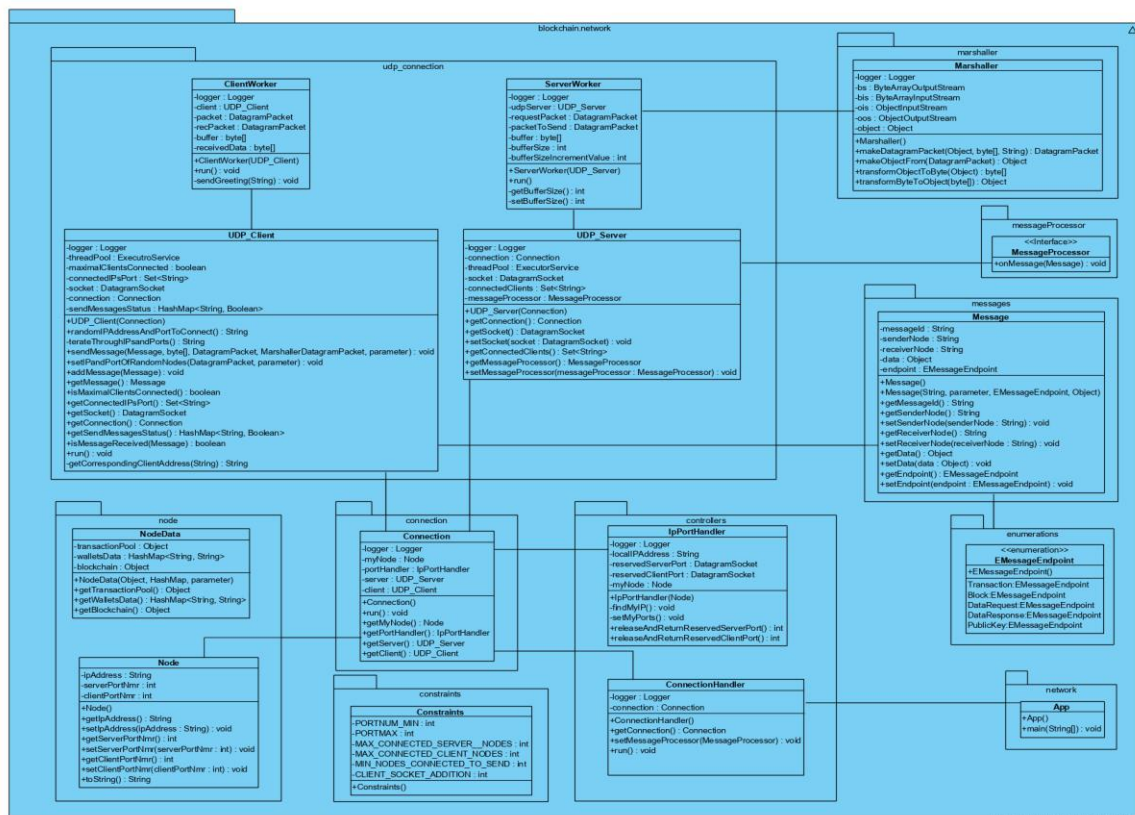


Abbildung 4:Klassendiagramm – Network Komponente des Blockchain System

Knoten Entdeckung: Die grundlegendste Art, andere Peers (Knoten) und deren bereitgestellte Funktionalität durch sogenanntes Scannen zu entdecken. Dabei beginnt ein Peer nach dem Start systematisch IP-Adressen und Ports, aus der Liste zugelassener Verbindungen, zu scannen [17]. Anschließend werden Nachrichten nur an diese Peers gesendet und weitergeleitet, welche den Aufruf zu Verbindung zugestimmt haben. Jeder Peer im P2P-Netzwerk ist zu jedem gegebenen Zeitpunkt mit höchstens sechs verschiedenen Peers verbunden. Davon agiert der Peer für maximal zwei Peers als Server (kann von ihnen Nachrichten empfangen) und für maximal vier verschiedene Peers als Client (kann an diese Nachrichten senden). Dadurch gibt es im Netzwerk immer mehr Sender als Empfänger, wodurch gewährleistet ist, dass jeder neu hinzugekommene Peer die Verbindung mit den bestehenden Peers aufbaut und nicht vom Netzwerk ausgelassen wird. Sollte nicht die maximale Anzahl an Peers verbunden sein, scannt der Peer ununterbrochen das Netzwerk nach neuen Verbindungen, sodass beim Ausfall einer bestehenden Verbindung schnellstmöglich eine neue hergestellt werden kann, sollte es freie Peers im Netzwerk geben.

Kommunikationstechnologie: Das Netzwerk verwendet das verbindungslose UDP (User Datagram Protocol) Protokoll, um Nachrichten zu übertragen. Dies ermöglicht die volle Kontrolle über die angewandte Netzwerkfehlerbehandlung, genau wie es echte P2P-Protokolle tun [17]. Um Datagram Pakete senden und empfangen zu können verbinden sich einzelne Knoten an zwei vordefinierte lokale UDP Sockets. Dafür werden Nachrichten durch den Marshaller von Objekten in Datagram Pakete und umgekehrt umgewandelt.

Kommunikations-Arbeiter: Client und Server instanziierten Threadpools, die eine gewisse Anzahl an Arbeitern initialisieren, damit mehrere Nachrichten gleichzeitig bearbeitet werden können. Der Client Arbeiter übernimmt dabei die Aufgabe Verbindungsaufrufe in das Netzwerk zu senden und die Liste der ungesendeten Nachrichten auf neue Nachrichten zu durchsuchen. Ungesendete Nachrichten werden dabei aus der Liste entfernt und an alle verbundenen Knoten gesendet. Server Arbeiter scannen den verbundenen Port auf eingehende Datagram Pakete, welche in Nachrichtenobjekte umgewandelt und bearbeitet werden. Dabei werden Verbindungsaufrufe akzeptiert und Knoten in die Liste der Verbindungen gespeichert. Bei sonstigen Nachrichten wird geprüft, ob der Sender in der Liste der Verbindungen ist und die Nachricht an den Node/Wallet Layer übergeben sowie an die restlichen Verbindungen weitergeleitet.

Nachrichten Schnittstelle: Das MessageProcessor Interface und seine Funktion wird von jedem Knoten implementiert, welcher den Netzwerk Layer benutzt, um mit anderen Knoten zu kommunizieren. Über diese Schnittstelle werden vom Server-Arbeiter umgewandelte Nachrichten zu weiteren Bearbeitung weitergeleitet.

Knoten Daten: Die Netzwerk Komponente definiert die genaue Art der Datensynchronisation zwischen Knoten, indem sie das Datamodel für das Speichern und Senden der Daten zur Verfügung stellt.

3.2 Node Komponente

Diese Komponente definiert die Datenmodelle der Blöcke, verwendet Hashing-Algorithmen und beinhaltet die Funktion, um den Merkle-Tree zu erstellen. Außerdem implementiert sie den Proof-of-Work Algorithmus und verifiziert neue Blöcke und Transaktionen. Zusätzlich werden von der Komponente die Netzwerk und Wallet Komponenten implementiert, die ihr es ermöglichen kryptografische Signaturen mit Hilfe des öffentlichen Schlüssels zu verifizieren und mit allen Knoten im Netzwerk zu kommunizieren.

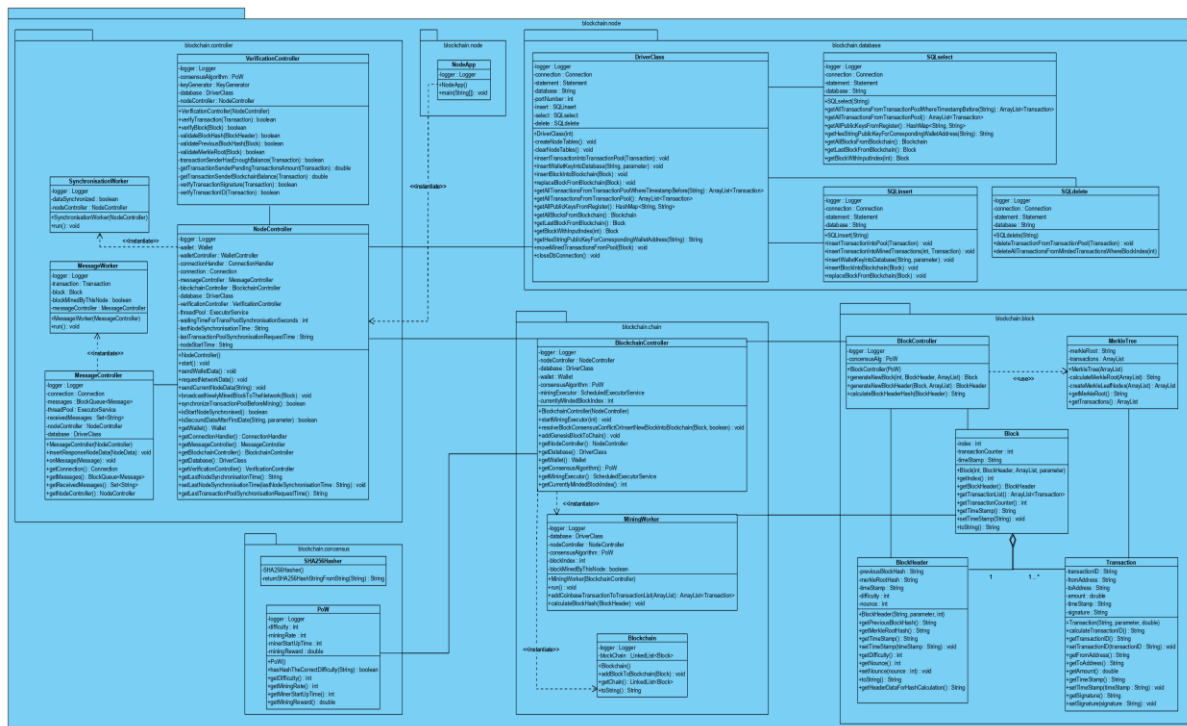


Abbildung 5: Klassendiagramm – Node Komponente des Blockchain System

Datenmodelle: Alle Abbildungen der für die Blockchain relevanten Objekte inklusive deren Attribute und Beziehungen sind in dieser Komponente inkorporiert. Damit sind vor allen die Modelle des Blocks, Block-Headers und der Transaktion gemeint.

Block: Besteht aus einem Index, der seine Ordnung in der Blockchain repräsentiert, dem Zähler, der die Anzahl der Transaktionen im Block angibt, sowie dem Block-Header Objekt und einer Liste aller Transaktionen im Block. Außerdem hat jeder Block einen Zeitstempel, welcher den Zeitpunkt aufzeichnet, wenn der Block gemined wurde.

Block-Header: Enthält den Hash des vorherigen Blocks in der Blockchain, den Merkle Root, sowie den Nounce. Alle Daten, welche zur Validierung des Bloks und für die Berechnung des Blockhash benutzt werden, sind im BlockHeader enthalten. Zusätzlich enthält der Block-Header auch einen Zeitstempel und den Schwierigkeitsgrad, der zum Zeitpunkt des mining vorgegeben war.

Transaktion: Jede Transaktion bekommt eine einzigartige Identifikationsnummer, welche durch das hashing der Transaktion mit der SHA-256 Funktion erstellt wird. Zudem enthält sie eine Absenderadresse und Empfängeradresse, welche die Adressen der Wallets speichert, die den Betrag senden bzw. empfangen. Abschließend bekommt jede Transaktion auch einen Zeitstempel und eine Signatur, welche mit dem privaten Schlüssel des Senders erstellt wurde.

Hashing: Um einzigartige Hashwerte zu erzeugen bedient sich die Implementation der Hashing Hilfsklasse welche als Singleton Pattern implementiert wird, wodurch es nur eine Instanz des Objekts gibt, dessen Funktion von allen Klassen aufgerufen werden kann. Die Hashwerte werden dabei mit Hilfe der Google Guava-Bibliothek erstellt. Der durch die Bibliothek bereitgestellte SHA-256 Algorithmus generiert eine 256-bit (32-byte) Signatur für die übergebene Text Eingabe.

Merkle Tree: Merkle Roots werden durch das Instanzieren neuer Merkle Tree Klassen erzeugt. Die dafür benötigte Liste der Transaktionen wird dem neuen Objekt im Konstruktor übergeben und es wird automatisch ein Merkle Root erzeugt, der im Objekt gespeichert wird.

Mining (Proof-of-Work): Der BlockchainController erstellt ein Service, der beim Starten des Knoten eine Startzeit für die Synchronisation und Validierung der Netzwerkdaten zur Verfügung stellt und danach in einem vom PoW vorgegebenen Tempo mit dem Mining neuer Blöcke startet. In der PoW Klasse befinden sich außerdem auch Parameter die den Rechenaufwand, welcher für das Erzeugen eines neuen Blocks, sowie die Belohnung für das Erzeugen eines neuen Blocks definieren. Ob ein angemessener Rechenaufwand erbracht wurde, wird durch Funktionen der PoW Klasse bestimmt. Bei jedem Miningversuch wird eine neue Instanz der MiningWorker Klasse erzeugt, welche Funktionen enthält, die einen gewissen Rechenaufwand betreiben und versuchen den Hash für den neuen Block zu berechnen, um die Möglichkeit zu bekommen den neuen Block zur Blockchain hinzuzufügen.

Verifikation: Eingehende Nachrichten welche neue Transaktionen oder Blocks enthalten werden an den VerificationController weitergeleitet und auf deren Korrektheit geprüft. Dabei wird bei neuen Transaktionen geprüft, ob der Sender der Transaktion genügend Mittel hat diese Transaktion zu tätigen, sowie ob die eingegangene Transaktion manipuliert wurde. Dafür wird die Identifikation der Transaktion geprüft und kontrolliert, ob die Transaktion auch wirklich vom angegebenen Sender mit seinem privaten Schlüssel signiert ist. Bei neuen Blöcken wird der im Block gespeicherte Hash des vorherigen Blocks sowie der Merkle Root und der Hash des neuen Block auf Korrektheit geprüft.

Datenbank: Alle Funktionalitäten der Datenbank werden in der DriverClass Klasse zusammengefasst. Sie ermöglicht es neue Daten in der Datenbank zu speichern sowie bestehende Daten abzurufen, zu aktualisieren und zu löschen. Die bereitgestellten Funktionalitäten ermöglichen die Speicherung neuer validierter Transaktionen im Transaktionspool, der Blockchain als Ganzes und der öffentlichen Schlüssel der Wallets mit den Transaktionen validiert werden.

3.3 Wallet Komponente

Diese Komponente importiert Logik aus der Netzwerk Komponente und Node Komponente, um Zugriff auf das bestehende Netzwerk zu haben, sowie von den vordefinierten Datenmodellen Gebrauch zu machen. Durch der in dieser Komponente enthaltenen Logik werden folgende Funktionalitäten bereitgestellt:

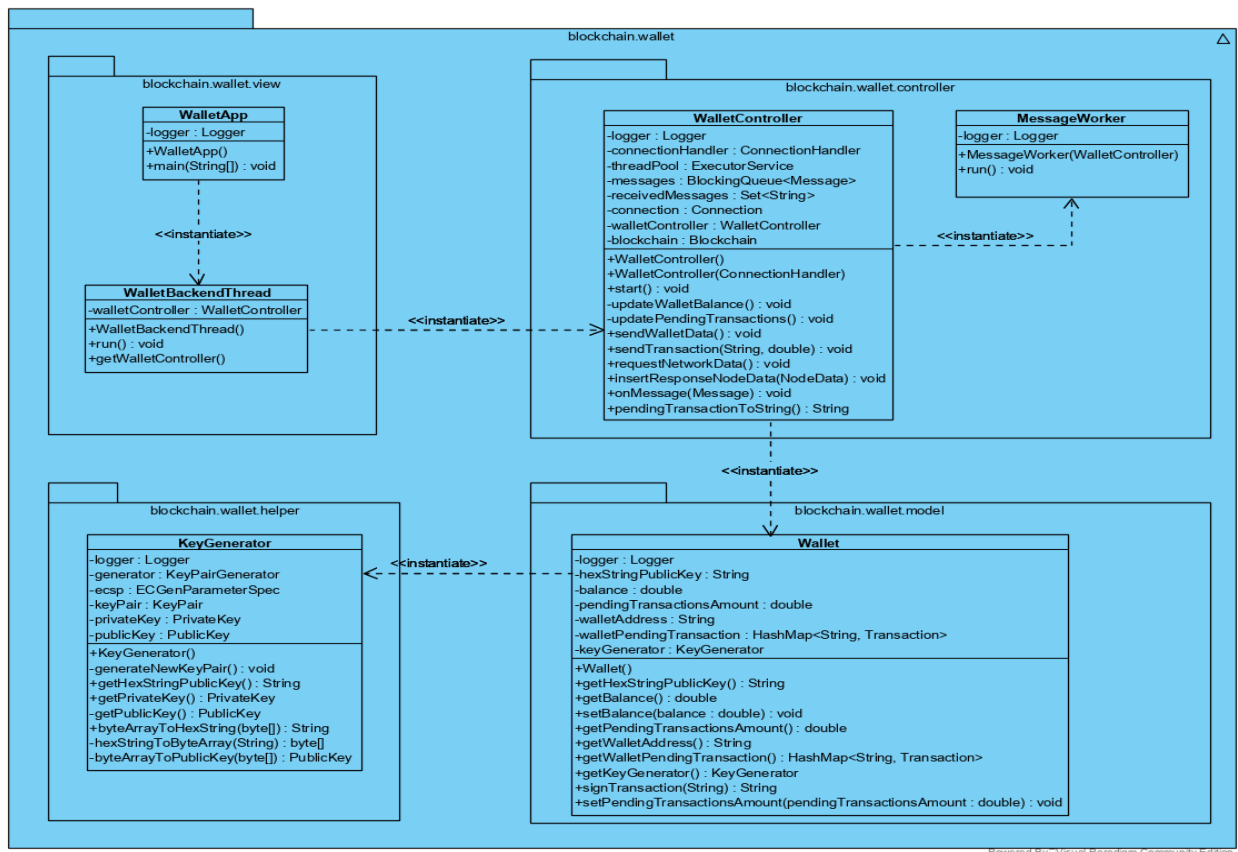


Abbildung 6:: Klassendiagramm – Wallet Komponente des Blockchain System

Befehlszeilenschnittstelle (CLI): Textbasierte Benutzeroberfläche, welche vom Benutzer Eingaben entgegennimmt und Programmzustände zurückliefert. Der Benutzer kann über die CLI Transaktionen senden und sich die Blockchain, ausstehende Transaktionen und Kontostand für diesen Wallet anzeigen lassen.

Schlüsselgenerierung: Unter Verwendung der Java Security Bibliothek werden Parameter angegeben, die zum Generieren der elliptischen Kurve und somit auch Schlüsselpaare verwendet werden. Dabei wird der `secp256k1` Parameter übergeben, welcher auch in der Bitcoin Schlüssel Kryptographie verwendet wird. [18]

Transaktion Generierung: Inputdaten für die Instanziierung des Transaktionsobjekts werden über die Benutzeroberfläche eingelesen und eine neue Transaktion erstellt, die mit Hilfe des Netzwerk Layers an alle Peers im Netzwerk gesendet wird.

4 Systemfunktionalitäten

Die Anforderungen an diese Implementation sind vor allem die Verwendung von Hashing-Algorithmen, Merkel-Trees sowie den Proof-of-Work Algorithmus, um die Blockchain Technologie besser verstehen zu können. Alle diese Konzepte und Mechanismen kommen beim Senden und Empfangen neuer Transaktionen, sowie beim Schürfen neuer Blöcke zum Einsatz. Daher wird im folgendem der gesamte Prozess dieser zwei Funktionalitäten inklusive der Validierung dieser Daten beschrieben.

4.1 Übertragung und Verifikation von Transaktionen

Neue Transaktionen werden vom Benutzer mit Hilfe der Benutzeroberfläche erstellt. Dabei überprüfen Hintergrundprozesse, ob einige Kriterien erfüllt sind und erstellen eine neue Transaktion, welche an alle Knoten im Netzwerk gesendet wird. Benutzeraktionen sowie Hintergrundprozesse der Wallet Komponente werden in folgender Abbildung dargestellt. Die Benutzerinteraktion mit der CLI ist für uns nicht von Interesse, weswegen wir uns bei dieser Funktion nur auf die Prozesse der Wallet Komponente fokussieren werden.

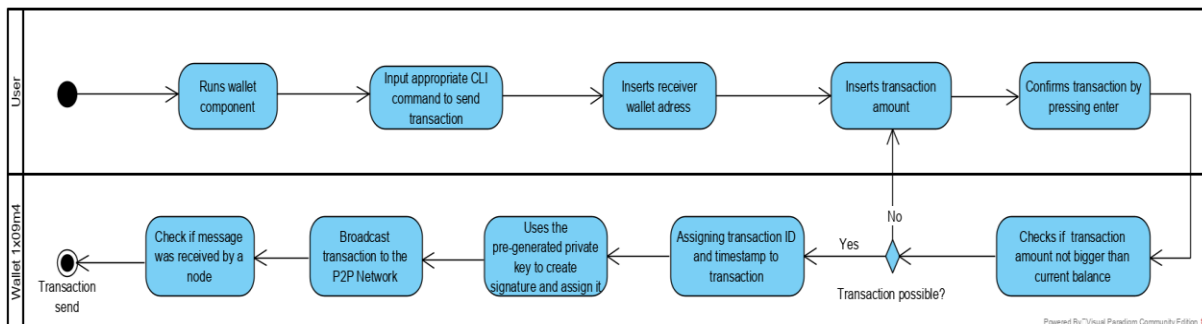


Abbildung 7: Aktivitätsdiagramm– Transaktion Erstellung

Guthaben Verifizierung: Bevor eine neue Transaktion erstellt wird, muss zuerst geprüft werden, ob diesem Wallet genügend Mittel zugeordnet sind, um den eingegebenen Betrag zu senden. Dafür werden alle Transaktionen in der Blockchain iteriert und die Sender- und Empfängeradressen mit der Adresse des Wallets verglichen. Sollte die Senderadresse übereinstimmen, wird der Gesamtbetrag verringert bzw. falls die Empfängeradresse übereinstimmt, der Betrag erhöht. Der dadurch kalkulierte Betrag repräsentiert das Guthaben für die bestätigten Transaktionen von dem nur noch

der Gesamtbetrag der ausstehenden Transaktionen, der in jedem Wallet lokal gespeichert ist, subtrahiert werden muss, um das finale Guthaben zu berechnen. Sollte das berechnete Guthaben höher als der übergebene Transaktionsbetrag sein, wird eine neue Transaktion erstellt und für den nächsten Schritt freigegeben.

ID und Zeitstempel Zuweisung: Jeder Transaktion wird automatisch beim Erzeugen eine Identifikation und ein Zeitstempel der Erstellung zugeordnet. Die ID wird in weiteren Schritten benutzt, um im Zusammenhang mit dem Zeitstempel die Einzigartigkeit der Transaktion zu gewährleisten. Durch diese Einzigartigkeit dient die Identifikation als perfekter Datensatz für die Signierung der Transaktion.

Signierung: Die mittels SHA-256 Funktion erstellte Identifikation der Transaktion wird mit dem vorgeneriertem privaten Schlüssel signiert, wodurch ein Textkette zurückgegeben wird, über die mit dem öffentlichen Schlüssel des Wallets geprüft werden kann, ob die Transaktion auch wirklich mit dem privaten Schlüssel signiert wurde.

Senden und bestätigen: Nachdem die Transaktion mit den Inputdaten initialisiert und signiert wurde bedient sich die Wallet Komponente mit den Funktionen der implementierten Netzwerk Komponente um die Nachricht, welche die Transaktion enthält an das Netzwerk zu übergeben.

Wir angedeutet werden beim Erstellen und Senden einer neuen Transaktion schon einige Konzepte der Blockchain-Technologie wie Hashing und kryptographische Verschlüsselung verwendet. Nachdem eine neue Transaktion den Knoten erreicht, wird sie mit anderen Transaktionen auf die Seite gelegt, um mit mehreren anderen Transaktionen in einen Block zusammengefasst werden zu können.

Bevor die Transaktion jedoch temporär gespeichert werden kann, muss ihre Validität geprüft werden. In weiterer Folge betrachten wir die in der folgenden Abbildung dargestellten Schritte der Verifikation dieser Transaktion vom Netzwerk Knoten.

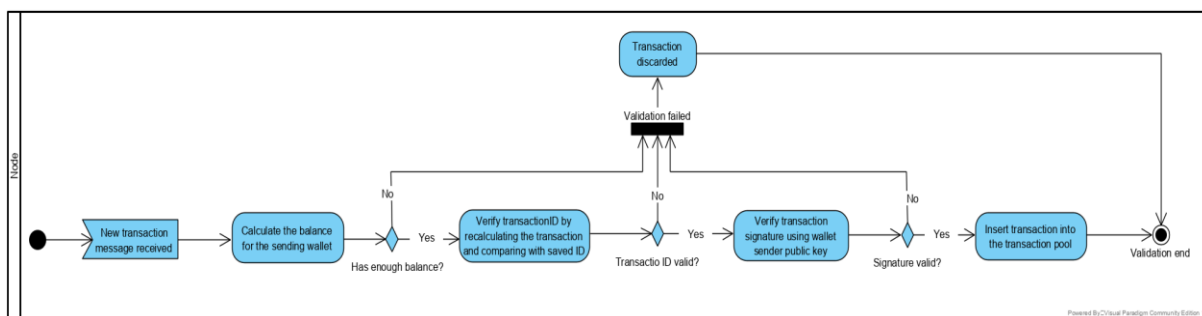


Abbildung 8: Aktivitätsdiagramm– Transaktion Validierung

Guthaben: Dieser Schritt beinhaltet dieselben Prozesse, welche schon von der Wallet Komponente durchgeführt wurden. Es unterscheidet sich lediglich der Zugriff auf die Liste ausstehender Transaktionen. Wallets senden neue Transaktionen, sodass sie immer eine lokale Liste der gesendeten und nicht bestätigten Transaktionen bereitstellen haben. Um den Betrag der

ausstehenden Transaktionen zu berechnen, greifen Node Komponenten auf die Transaktionen im Transaction Pool zu. Damit sich die Transactions pools bei verschiedenen Knoten nicht unterscheiden und so unterschiedliche Beträge berechnet werden, synchronisieren alle Knoten ihren Transactions pool in regelmäßigen abständen.

ID Verifizierung: Da bei weiteren Schritten der Verifikation die Identifikation der Transaktion zum Einsatz kommt, wird die ID für die gegebenen Transaktionsdaten erneut mittels SHA-256 berechnet und mit der in der Transaktion angegebenen ID verglichen. Dadurch wird auch teilweise sichergestellt, dass die Transaktionsdaten nicht manipuliert wurden, da andere Daten eine unterschiedliche ID ergeben wurden.

Verifizierung der Unterschrift: In diesem Schritt werden die von der Java Security Bibliothek zur Verfügung gestellten Funktionen benutzt, um mit Hilfe des öffentlichen Schlüssels des Senders und der Transaktion ID zu bestätigen, dass die Signatur aus der Transaktion auch wirklich vom Senderwallet erstellt wurde. Damit jeder Knoten den Zugriff auf die öffentlichen Schlüssel aller Knoten im Netzwerk hat, senden Wallets beim Start ihren öffentlichen Schlüssel allen Knoten im Netzwerk. Diese Schlüssel werden regelmäßig synchronisiert und neuen Knoten im Netzwerk gesendet, sodass alle Knoten den gleichen Stand haben.

Nachdem die Transaktion alle validierungsschritte erfolgreich absolviert hat, wird sie in dem Transaktionspool gespeichert. Alle Transaktionen aus dem Pool werden anschließend aus dem Pool entnommen und in einen neuen Block eingefügt.

4.2 Übertragung und Verifikation von Blöcken

Neue Blöcke werden in dem Miningprozess erstellt. Dabei handelt es sich um einen regelmäßig wiederholenden Prozess, in dem eine bestimmte Anzahl an Transaktionen aus dem Transaktionspool entnommen werden und versucht wird einen neuen Block, welcher diese Transaktionen beinhaltet zu kreieren.

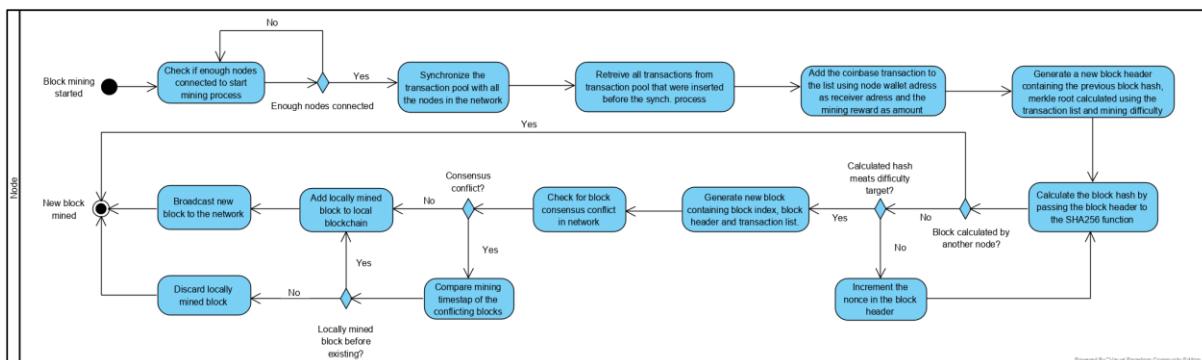


Abbildung 9: Aktivitätsdiagramm – Block Mining

Alle Mining Aktivitäten im Sinne des Proof-of-Work Algorithmus sind in der vorherigen Abbildung ersichtlich und werden im folgendem genauer betrachtet.

Netzwerkverbindungen: Den Miningprozess zu starten, ohne dass der Knoten mit weiteren Knoten im Netzwerk verbunden wird wäre nutzlos, da der Knoten nicht den Transaktionspool synchronisieren könnte bzw. nicht mit dem Rest des Netzwerks abstimmen kann, welche Transaktionen der nächste Block beinhalten soll und der Knoten außerdem nicht den neuen Block mit dem Netzwerk teilen kann, falls er als erster den korrekten Hash ausrechnet. Daher wird der Prozess so lange nicht gestartet, bis der Knoten nicht mit einer angemessenen Anzahl an Knoten verbunden ist und bekommt somit Zeit bis zum nächsten Mining Zyklus sich mit anderen Knoten im Netzwerk zu verbinden bzw. Netzwerkdaten zu synchronisieren.

Transactionpool Synchronisation: Beim Beginn jedes Mining Prozesses bekommen die Knoten ein gewisses Zeitintervall die Netzwerkdaten zwischeneinander auszutauschen und vor allem den Transaktionspool zu synchronisieren. Knoten die es nicht in vorgegebener Zeit schaffen die Daten mit dem Rest des Netzwerks zu synchronisieren nehmen nicht am Mining Prozess teil und müssen bis zum nächsten Zyklus warten.

Transaktionspool Transaktionen: Wenn alle Transactionpools synchronisiert sind und den gleichen Stand haben, übernehmen Knoten die Liste der Transaktionen aus der Datenbank. Jeder Knoten gibt zu den bestehenden Transaktionen noch die Coinbase Transaktion hinzu. Die Coinbase Transaktion enthält als Empfängeradresse die Wallet Adresse des Knoten, welcher versucht den neuen Block zu schürfen und als Betrag die vom Proof-of-Work vorgegebene Belohnung, für das erfolgreiche hinzufügen eines neuen Block in die Blockchain. So wird gewährleistet, dass der Knoten welcher als erster den neuen Block hinzufügt auch die Belohnung enthält.

BlockHeader Erzeugung: Für die Liste der zu enthaltenen Transaktionen wird der Merkle Root berechnet, der mit anderen Daten Bestandteil des BlockHeader wird. Weiteres enthält der BlockHeader den Hash des vorherigen Blocks sowie einen automatisch generierten Zeitstempel, Nounce die von null beginnt und die vom Proof-of-Work vorgegebene Schwierigkeit für das Errechnen des Block Hashes.

Mining Prozess: Um den Block-Hash zu berechnen und einen angemessenen Rechenaufwand einzubringen wird der BlockHeader so lange mit der SHA-256 Funktion gehasht, bis ein Hash entsteht, der für den PoW eine angemessene Schwierigkeit hat. Da die Hash Funktion für dieselbe Eingabe auch immer dieselbe Ausgabe liefert wird nach jedem unzureichendem Hash das Nounce inkrementiert, um eine neue Ausgabe zu erhalten. Wird ein angemessener Hash berechnet, bevor ein anderer Knoten es schafft, kann eine neue Block Instanz generiert werden, welche den BlockHeader mit dem korrekten Nounce und die Liste der Transaktionen enthält.

Konsenskonflikt: Entsteht, wenn mehrere Knoten zu ungefähr gleichen Zeit den Block Hash berechnen und ihre Blöcke ins Netzwerk senden. Im Falle eines Konflikts vergleichen die Knoten die Zeitstempel der Blöcke und fügen den Block, der früher erstellt wurde, in die Blockchain hinzu. Da die Zeitstempel automatisch beim Erstellen einer neuen Blockinstanz eingefügt werden und Einheiten von Nanosekunden enthalten, ist die Wahrscheinlichkeit, dass zwei Blöcke den gleichen Zeitstempel enthalten, sehr gering.

Sobald ein neuer Block den Knoten erreicht, wird er validiert, bevor überhaupt Konsenskonflikte gelöst und der Block in die Blockchain eingefügt wird. Die einzelnen Schritte der Block(chain) Validierung sind in folgender Abbildung dargestellt und werden des Weiteren näher erläutert.

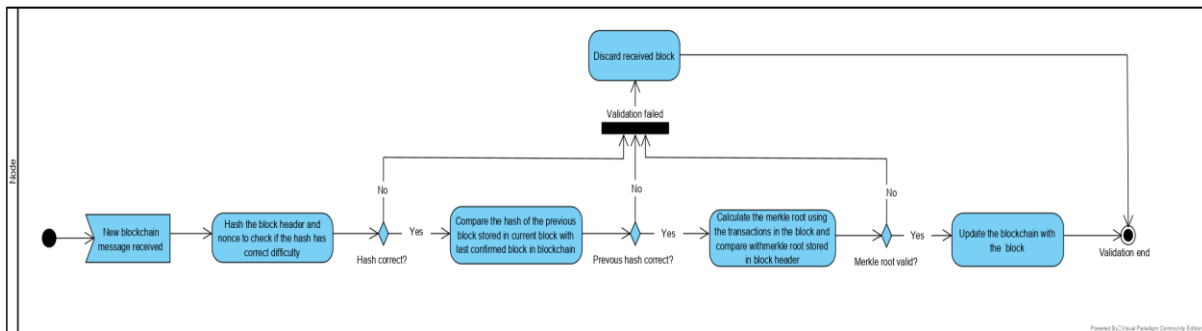


Abbildung 10: Aktivitätsdiagramm – Block(chain) Validierung

Hash-Validierung: Um zu validieren, dass der Hash des neuen Blocks korrekt ist bzw. dass er unter Beachtung der vom Proof-of-Work vorgegebenen Schwierigkeit berechnet wurde, wird mit den im Block angegebenen Daten der Hash des Blocks erneut berechnet. Dafür wird der im Block enthaltene BlockHeader und der angegebene Nounce erneut an die SHA-256 Funktion übergeben. Der von dieser Funktion zurückgegebene Hashwert wird an die Funktion der PoW Instanz weitergeleitet, die die Richtigkeit des Hashes zurückliefert. Erst wenn bestätigt wurde, dass ein angemessener Rechenaufwand eingebracht wurde, werden weitere Daten aus dem Block geprüft.

Validierung des vorherigen Hashwerts: Nachdem der Block Hash validiert wurde, werden die Angaben im Block geprüft. Dafür wird erstens der angegebene Hash des vorherigen Blocks in der Blockchain geprüft. Es wird der letzte Block in der Blockchain aus der Datenbank ausgelesen, sein Hashwert berechnet und mit dem im Block angegebenen Hashwert verglichen.

Merkle Root Validierung: Als nächstes wird der im Block angegebene Merkle Root geprüft, wofür die Liste der Transaktionen aus dem Block ausgelesen und für die Liste der Merkle Root neu berechnet wird. Der berechnete Merkle Root wird mit dem im Block angegebenen verglichen.

Blockchain-Aktualisierung: Wurden alle Validierungsschritte erfolgreich durchgeführt und die Korrektheit des Blocks bestätigt, fügt der Knoten diesen Block in die Blockchain hinzu. Dieser Block bzw. sein Hashwert wird dann benutzt, um den nächsten Block in der Blockchain zu berechnen. Sollte der Block nicht korrekt sein, wird er im Laufe der Validierung verworfen, wodurch andere Knoten immer noch die Chance haben ihren Block in die Blockchain hinzuzufügen.

5 Implementierung

Das öffentliche Blockchain-System wurde mittels Java 11 realisiert wofür die Entwicklungsumgebung von Eclipse die notwendigen Entwicklungstools zur Verfügung gestellt hat. Zusätzlich wurde dabei das Build-Werkzeug Maven benutzt, um Abhängigkeiten wie z.B. die Bibliothek, welche Hashing ermöglicht, automatisch zu verwalten und bei Bedarf herunterzuladen.

Ein relationales Datenbanksystem lieferte die SQLite Programm-bibliothek, wodurch der Node-Komponente ermöglicht wurde die Transaktionen, Blocks und Wallet-Schlüssel lokal zu speichern. Abschließend wurde wie schon beim beschreiben der Netzwerk-Komponente erwähnt das UDP Kommunikationsprotokoll benutzt, um Datagramme zwischen einzelnen Knoten zu senden und so die Kommunikation zu ermöglichen.

Der dabei entstandene Quellcode wurde auf der von der Universität Wien zur Verfügung gestellten Webanwendung GitLab gesichert und regelmäßig aktualisiert.

Um die Beziehungen zwischen einzelnen Objekten und das Verhalten des Systems bei Ausführen von Hauptfunktionen darzustellen, bedienen wir uns im folgendem des UML Sequenzdiagramms. Durch das Verhaltensdiagramm werden alle Details grafisch dargestellt ohne Seitenlangen Quellcode präsentieren zu müssen. Sollte es weiterhin Bedarf zur ausführlicher Untersuchung des Systems geben, ist der Quellcode auf GitLab (<https://git01lab.cs.univie.ac.at/spba/2021ws/01576891-nemanja-srdanovic>) jederzeit aufrufbar.

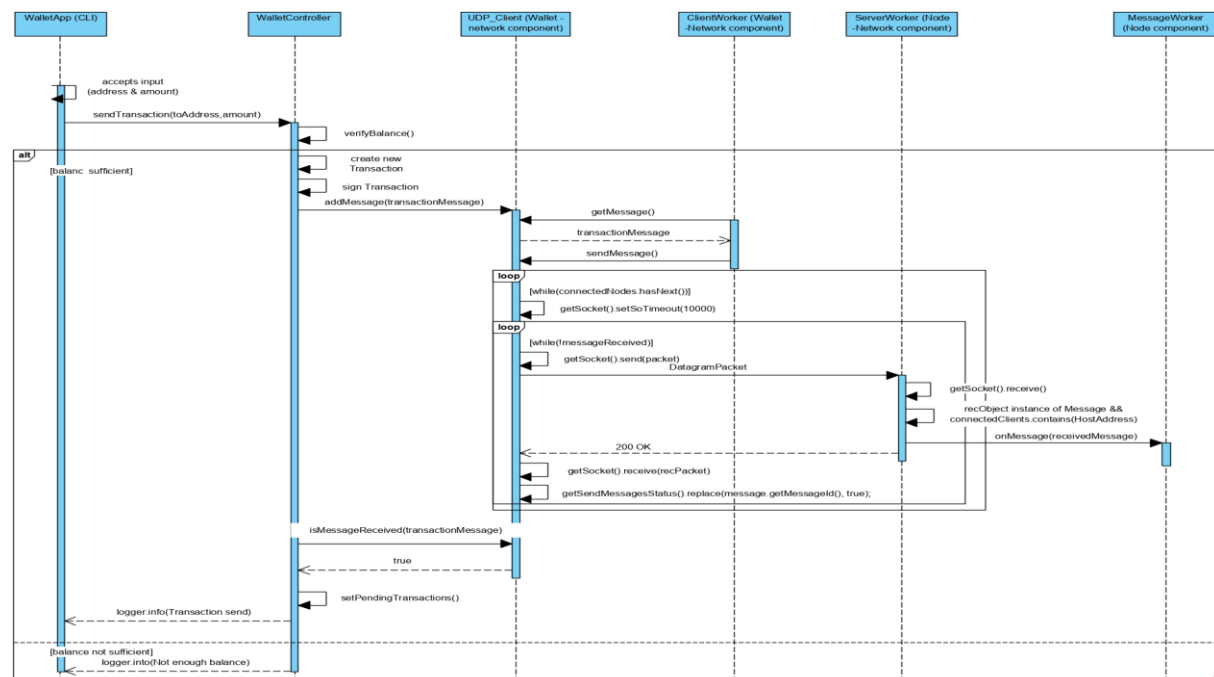


Abbildung 11: Sequenzdiagramm– Transaktion Erstellung innerhalb der Wallet Komponente

Die Erstellung und Versendung einer Transaktion wird von der Wallet-Komponente durchgeführt, die dafür sechs Objekte einsetzt. Inputdaten werden über die CLI übernommen und Resultate zurückgegeben. Vom WalletController wird das Guthaben geprüft und neue Transaktionen erstellt, die über von Netzwerk-Komponente zur Verfügung gestellte UDP_Client und ClientWorker an die verbundenen Knoten weitergeleitet werden. Nachrichten werden von MessageWorker der Node-Komponente empfangen und die Empfangsbestätigung an den Senderknoten zurückgesendet. Sobald die Bestätigung eines verbundenen Knotens eintrifft, wird der Benutzer über das erfolgreiche Senden der Transaktion informiert. Beim zweiten Anwendungsfall, in dem der Benutzer nicht genügend Guthaben hat, um die Transaktion zu tätigen, wird er über die CLI informiert und angefordert einen neuen Betrag anzugeben oder die Sendung abubrechen.

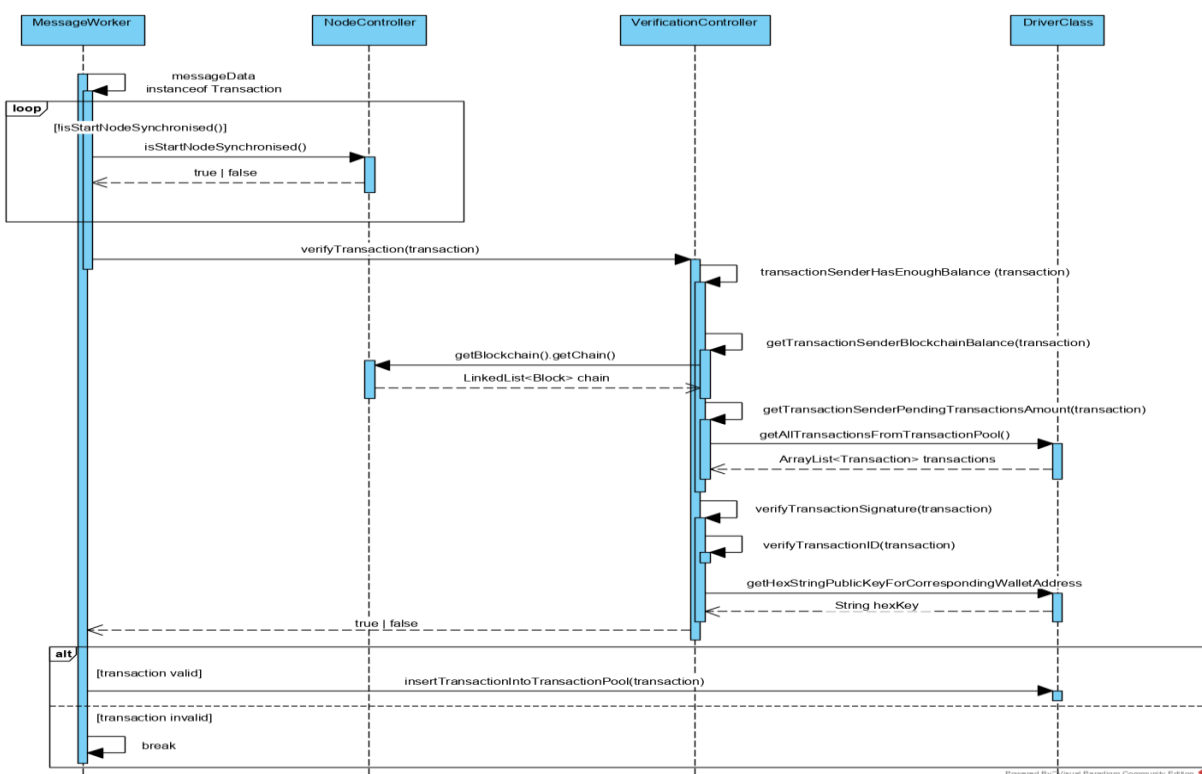


Abbildung 12:Sequenzdiagramm– Transaktion Validierung innerhalb der Node Komponente

Bei Eintreffen einer Transaktionsnachricht wird von der Node-Komponente zuerst geprüft, ob schon alle Netzwerkdaten aktualisiert wurden. Die Synchronisation aller Daten ist notwendig, um die Transaktion ordnungsgemäß validieren zu können. Sobald die Daten erfolgreich synchronisiert wurden, übergibt man das Transaktionsobjekt an den VerificationController der alle Schritte der Validierung durchführt, wofür er mehrmals die Datenbank abfragt. Wird die Validität einer Transaktion bestätigt, fügt man sie in die Transaktionspool Tabelle der Datenbank ein.

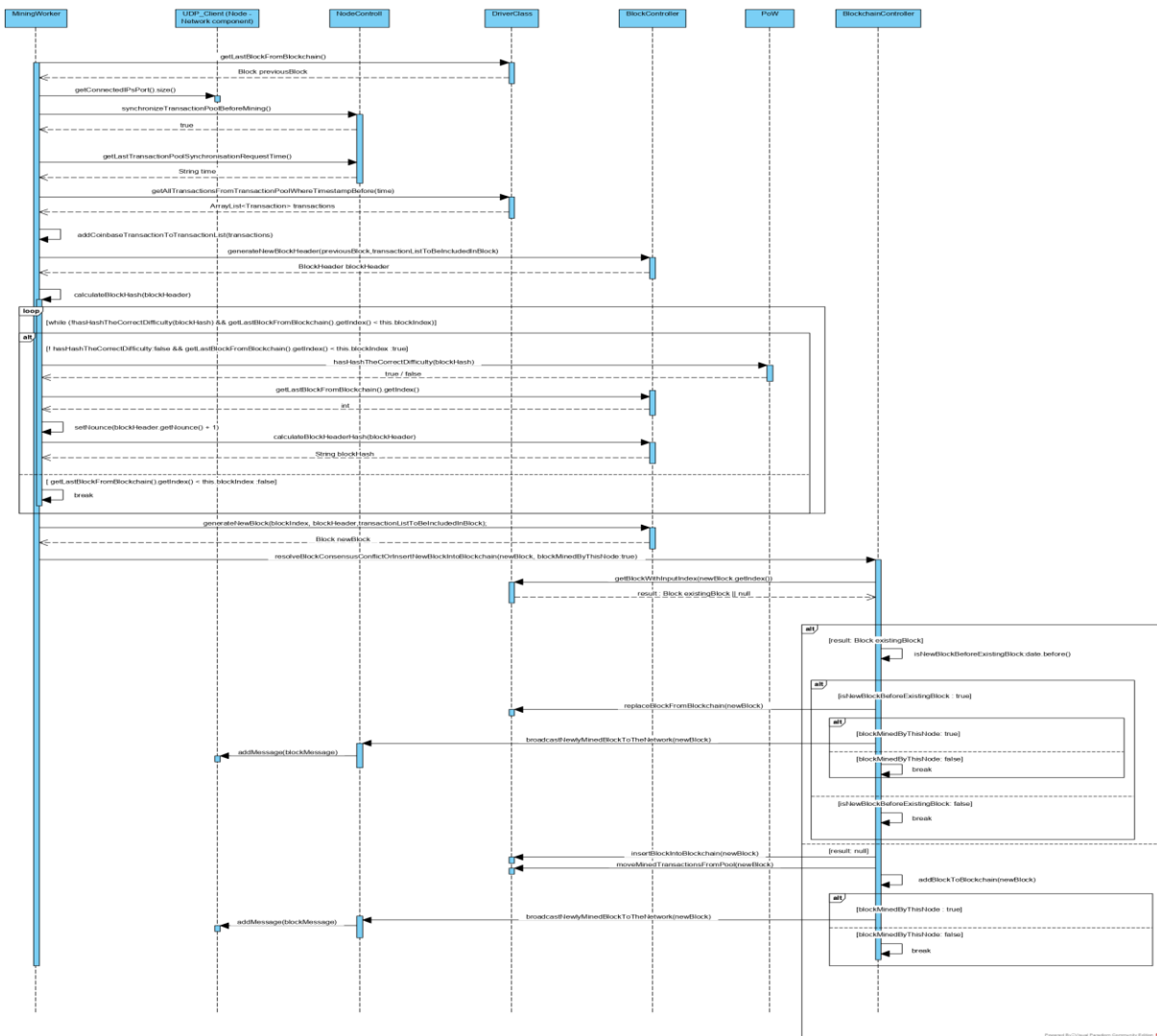


Abbildung 13: Sequenzdiagramm– Bitcoin-Mining innerhalb der Node Komponente

Schritte, um einen neuen Block zu erstellen wurden schon detailliert im Kapitel 4.2 beschrieben. Die oben angegebenen Objekte und Funktionen sollen diesen Prozess noch genauer erläutern, werden aber nicht erneut beschrieben, da das Konzept sich nicht von dem im Kapitel 4.2 unterscheidet. Jeweils wichtig zu erwähnen ist dabei nur, dass der Miningprozess in einem selbstständigen Thread läuft, welcher von einem Threadpool-Service, in dem vom PoW vorgegebenem Intervall ausgeführt wird. Das eigentliche Berechnen des Blockhash ist dabei ein simpler Prozess, welcher in dem Thread läuft. Dafür wird in einer while-Schleife der BlockHeader und das ständig inkrementierende Nounce so lange gehasht, bis die Funktion der PoW Instanz zurückliefert, dass der Hashwert ein angemessenes Format (Difficulty) hat.

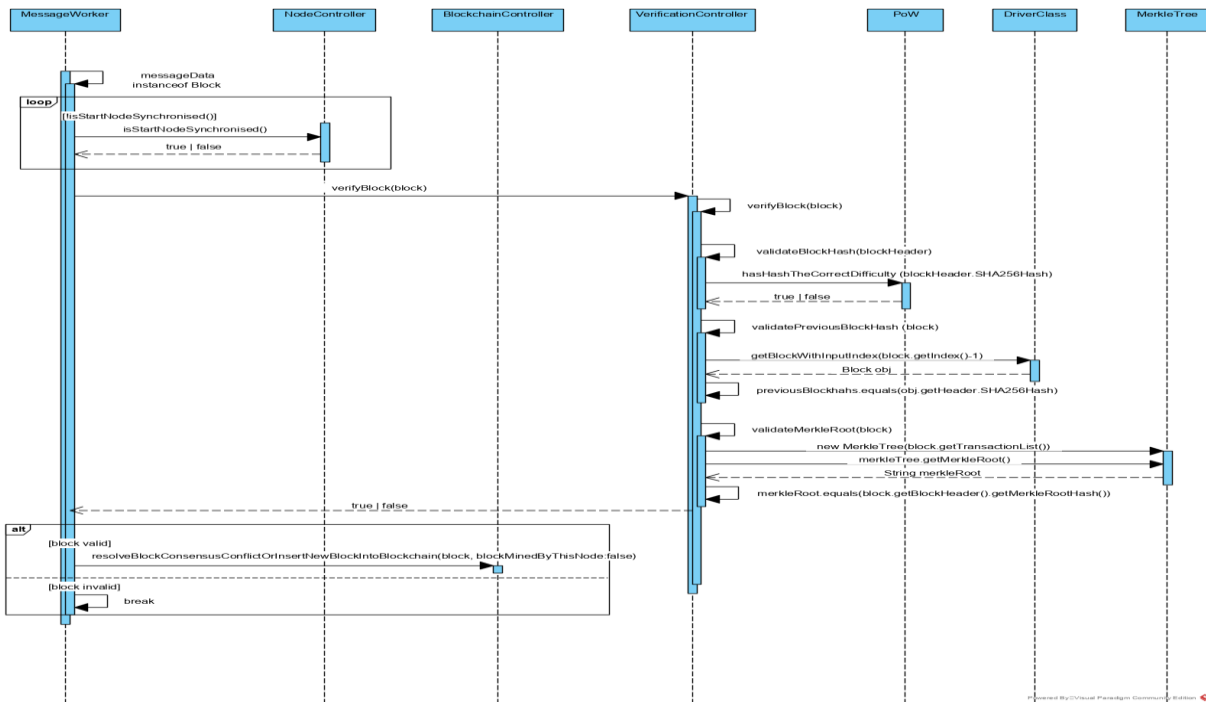


Abbildung 14: Bitcoin-Validierung innerhalb der Node Komponente

Genau wie bei der Validierung einer Transaktion wird auch bei Blocknachrichten zuerst die Korrektheit aller Netzwerkdaten geprüft. Anschließend wird das Blockobjekt an die Validierungsfunktion des VerificationController weitergeleitet, der die Korrektheit des neuen Blocks prüft. Dafür werden Funktionen der PoW Instanz aufgerufen, Daten aus der Datenbank abgerufen, sowie neue Instanzen des MerkleTree erstellt. Valide Blöcke werden danach in die Datenbank gespeichert, um nur wieder abgerufen zu werden, falls es zu einem Konsenskonflikt kommen sollte, oder der nächste Block in der Blockchain erstellt wird und ungültige Blöcke verworfen.

Wie schon am Anfang des Kapitels erwähnt, kann man den kompletten Quellcode von GitLab heruntergeladen, um das System laufen zu lassen. Dafür müssen vor allem die Wallet-Komponente und Node-Komponente in eine Entwicklungsumgebung wie z.B. Eclipse importiert und die Maven Abhängigkeiten installiert werden. Anschließend sollten mehrere Instanzen der NodeApp und WalletApp in der Entwicklungsumgebung gestartet werden. Die dabei erstellten Knoten verbinden sich automatisch miteinander und tauschen Nachrichten aus. In der Console der Entwicklungsumgebung kann der Nachrichtenaustausch verfolgt werden und über die CLI der WalletApp Console neue Transaktionen erzeugt, sowie der Stand der Blockchain nachverfolgt werden.

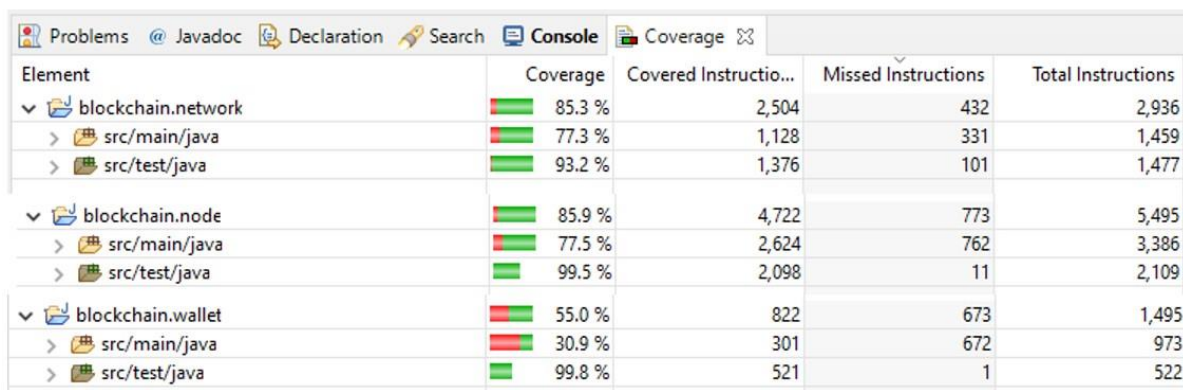
Beim Start hat jeder Knoten der Node-Komponente eine Synchronisationszeit, um Daten mit dem Rest des Netzwerks auszutauschen, bevor er mit dem Mining neuer Blöcke beginnt. Diese Zeit endet nachdem die zweite volle Minute ab Startzeit der NodeApp vollendet ist. Danach beginnt der Knoten jede anschließende Minute den Miningprozess.

6 Auswertung

Noch vor der Implementierung des Projekts wurde eine umfangreiche Literaturanalyse durchgeführt und anhand dieser ein Konzept erstellt, welcher alle folgenden Meilensteine sowie den Zeitplan definiert hat. Das Konzept ist anschließend vom Betreuer ausgewertet und ein finaler Plan für die Realisierung des Projekts festgelegt worden.

Um den Fortschritt und die im Konzept definierten Ziele auszuwerten, wurden alle zwei Wochen mit dem Betreuer die Meilensteine evaluiert. Basierend auf dieser Evaluation sind anschließend Code und Meilensteinanpassungen vorgenommen wurden.

Zusätzlich wurde Code-Sicherung sowie testgetriebene Entwicklung mit dem JUnit-Testframework ermöglicht. Dabei wurde als Ziel gesetzt alle wichtigen Methoden der drei Komponenten mit JUnit-Tests zu sichern. Wie in nächster Abbildung zu sehen ist wurde dabei eine sehr hohe Abdeckung des gesamten Quellcodes erreicht.



Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
▼ blockchain.network	85.3 %	2,504	432	2,936
> src/main/java	77.3 %	1,128	331	1,459
> src/test/java	93.2 %	1,376	101	1,477
▼ blockchain.node	85.9 %	4,722	773	5,495
> src/main/java	77.5 %	2,624	762	3,386
> src/test/java	99.5 %	2,098	11	2,109
▼ blockchain.wallet	55.0 %	822	673	1,495
> src/main/java	30.9 %	301	672	973
> src/test/java	99.8 %	521	1	522

Abbildung 14: JUnit Test – Abdeckung des Quellcodes

Vor allem wurde bei der Auswertung der Komponenten, sowie des Systems als ganzes die Minimum Viable Product Strategie herangezogen. Im Sinne dieser Strategie wurde die erste funktionsfähige Iteration des Produkts geschaffen.

7 Conclusio

Durch die Analyse diverser Literatur und Projekte aus dem Bereich der Blockchain Technologie und der Implementation eines vereinfachten öffentlichen Blockchain-Systems ist es gelungen Verständnis über die Bausteine und Funktionsweise der Blockchain-Technologie zu erlangen. Dabei wurde vor allem das Verständnis über die Konzepte und Mechanismen des Proof-of-Work sowie Absicherung von Datenstrukturen mittels Hash-Algorithmen und Digitalen Signaturen erweitert.

Einige Konzepte gingen jedoch über den Rahmen dieses Projekts hinaus, sodass im Sinne der Minimum Viable Product Strategie entschieden wurde sie für zukünftige Implementationen zu berücksichtigen. Ein solches Konzept wäre die Longest chain rule welches bei Konsenskonflikten zum Einsatz kommt und bei Erweiterungen dieses Projekts implementiert werden könnte.

8 Referenzen

- [1] S. Nakamoto, „Bitcoin: A Peer-to-Peer Electronic Cash System,“ *Cryptography Mailing list at <https://metzdowd.com>*, 2009.
- [2] M. Betancourt, „Bitcoin,“ *Ctheory*, 2018.
- [3] V. Y. Kemmoe, W. Stone, K. Jeehyeong, K. Daeyoung und S. Junggab, „Recent advances in smart contracts: A technical overview and state of the art,“ 2020.
- [4] M. Nofer, P. Gomber, O. Hinz und D. Schiereck, „Blockchain,“ 2017.
- [5] M. Faizan, F. Förster, T. Brenner und C. Wittwer, „Decentralized bottom-up energy trading using Ethereum as a platform,“ 2019.
- [6] F. Jinhua, Q. Sihai, H. Yongzong, L. Bin und Y. Chao, „A Study on the Optimization of Blockchain Hashing Algorithm,“ 2020.
- [7] A. Porat, P. Avneesh, S. Parth und A. Vinit, „Blockchain Consensus: An analysis of Proof-of-Work and its applications“.
- [8] W. Wenbo, T. H. Dinh, H. Peizhao, X. Zehui, N. Dusit, W. Ping, W. Yonggang und K. I. Dong, „A Survey on Consensus Mechanisms and Mining,“ 2019.
- [9] R. C. Merkle, „A Digital Signature Based on a Conventional Encryption Function,“ 1988.
- [10] M. Bosamia und P. Dharmendra, „Current Trends and Future Implementation Possibilities of the Merkel Tree,“ *International Jurnal of computer sciences and engineering*, pp. 294-301, 2018.
- [11] I. Khan, „Using blockchain technology for file synchronization,“ in *OP Conference Series: Materials Science and Engineering*, 2019.
- [12] J. B. Bert , D. A. Tamburri und J. Willem, „Blockchains: A Systematic Multivocal Literature Review,“ 2020.
- [13] „CoinGecko,“ 5 February 2022. [Online]. Available: <https://www.coingecko.com/>.
- [14] F. Casino, T. K. Dasaklis und C. Patsakis, „A systematic literature review of blockchain-based applications: Current status, classification and open issues,“ *Telematics and Informatics*, pp. 55-81, 2019.
- [15] R. Böhme, N. Christin und B. Edelman, „Bitcoin: Economics, Technology, and Governance,“ *Journal of Economic Perspectives*, Bd. 29, p. 213–238, 2015.
- [16] N. Pranav, P. Dhiren, B. Yann, L. Romaric, K. Saru und M. K. Khan, „Dissecting bitcoin blockchain:

Empirical Analysis of Bitcoin network (2009-2020),“ 2020.

[17] University of Vienna - Faculty of Computer Science, Research Group Software Architecture, „Connected and Autonomous Vehicle Mesh, 2020SS - 052500 VU - Distributed Systems Engineering,“ 2020.

[18] D. Etzold, „etzold.medium.com,“ 23 Dezember 2020. [Online]. Available: <https://etzold.medium.com/elliptic-curve-signatures-and-how-to-use-them-in-your-java-application-b88825f8e926>. [Zugriff am November 2021].

9 Abbildungsverzeichnis

Abbildung 1: Sequence-of-blocks-in-a-blockchain-Zheng-et-al-2017	5
Abbildung 2:Hash_Tree Original illustration by David Göthberg, Sweden.....	6
Abbildung 3:Kathleen E. Wegrzyn Eugenia Wang, Types of Blockchain - Public, Private, or Something in Between.....	7
Abbildung 4:Klassendiagramm – Network Komponente des Blockchain System	9
Abbildung 5:Klassendiagramm – Node Komponente des Blockchain System	11
Abbildung 6:: Klassendiagramm – Wallet Komponente des Blockchain System.....	13
Abbildung 7:Aktivitätsdiagramm– Transaktion Erstellung	14
Abbildung 8:Aktivitätsdiagramm– Transaktion Validierung.....	15
Abbildung 9:Aktivitätsdiagramm– Block Mining	16
Abbildung 10:Aktivitätsdiagramm– Block(chain) Validierung	18
Abbildung 11:Sequenzdiagramm– Transaktion Erstellung innerhalb der Wallet Komponente	19
Abbildung 12:Sequenzdiagramm– Transaktion Validierung innerhalb der Node Komponente	20
Abbildung 13:Sequenzdiagramm– Bitcoin-Mining innerhalb der Node Komponente.....	21
Abbildung 14:Bitcoin-Validierung innerhalb der Node Komponente.....	22