UNIVERSITÄT HAMBURG
FAKULTÄT FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

Softwareunterstützte Dezentralisierung von Geschäftsprozessen in B2B-Anwendungen mittels der Blockchain-Technologie

---

eingereicht beim Fach-Promotionsausschuss Informatik von

Wolf Dietmar Posdorfer

Dissertation
Zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
Dr. rer. nat.

Hamburg, September 2021

Vorsitz der Prüfungskommission	Prof. Dr. Janick Edinger
Erstgutachter	Prof. Dr. Winfried Lamersdorf
Zweitgutachter	Prof. Dr. Walid Maalej
Tag der Disputation	22.02.2022

Much work complete.
Allow demonstration.

– Abathur

Kurzzusammenfassung

Die heutige Informationstechnik erfährt eine laufende Zentralisierung, sowohl in ihrer Infrastruktur als auch in Geschäftsprozessen mit mehreren Akteuren. Um für Vertrauen in Geschäftsprozessen mit teils kontrahierenden Akteuren zu sorgen, werden notwendigerweise Dritte für Vertrauen und Konsens hinzugezogen. Ein Bereich, der besonders stark von Zentralisierung betroffen ist, ist das globale Bank- und Geldsystem, mit wenigen Anbietern und Millionen Kunden.

Ausgelöst durch die Finanzkrise 2008, entstand als Gegenpol die erste vollkommen dezentrale digitale Währung, die ohne dritte Parteien oder Intermediäre Vertrauen zwischen allen teilnehmenden Akteuren herstellt: *Bitcoin*. Vereinfacht dargestellt, besteht Bitcoin technisch aus kryptografischen Prüfsummen über seine Daten gepaart mit einem verteilten Konsensalgorithmus, der auf dem Beweis einer erbrachten Arbeitsleistung beruht. Aufbauend auf den Errungenschaften von Bitcoin sind über die Jahre viele weitere Blockchain-Technologien entstanden, die weitere Funktionen bieten, jedoch in den meisten Fällen als einziges Anwendungsgebiet ausschließlich digitale Währungen umfassen. Sowohl die technische Umsetzung der Technologien als auch deren hauptsächliche Auslegung auf Währungen machen sie ungeeignet für den Einsatz in heutigen Geschäftsprozessen, unter Anderem wegen ihrer begrenzten Skalierbarkeit.

Zur dezentralen Koordination von Geschäftsprozessen mithilfe von Blockchain-Technologie bedarf es entsprechend einiger Anpassungen und Weiterentwicklungen bisheriger Konzepte und existierender Lösungen. Die vorliegende Dissertation identifiziert zunächst diesen Anpassungsbedarf und zeigt anschließend Problemstellungen als auch entsprechende Lösungsstrategien auf. Der Fokus liegt dabei auf der Untersuchung, Konzeption und Erarbeitung von weiterführenden Konzepten für die Integration von Geschäftsprozessen im B2B-Bereich mithilfe von Blockchain-Technologien unter besonderer Berücksichtigung der *dezentralen Koordination ohne intermediäre Entscheidungsfindung*.

Als Ergebnis dieser Arbeit wird hierzu ein Konzept und eine komponentenbasierte Architektur für eine generalisierte Blockchain zur Abbildung von Geschäftsprozessen entwickelt. Deren exemplarische Umsetzung wird anhand ausgewählter Anwendungsszenarien evaluiert, wobei die entwickelten Optimierungen, wie die Minimierung der Blockchain-Historie anhand eines modifizierten BFT-Algorithmus und Abhängigkeitsgraphen zwischen Transaktionen, besondere Berücksichtigung erfahren.

Abstract

Today's IT is experiencing an ongoing centralization not only in its infrastructure, but also in its business processes with multiple actors. To cater to the needs of trust in business processes with multiple opposing actors, it quickly becomes necessary to include third parties to generate trust and consensus. One of the sectors that is highly affected by centralization is the global bank and (fiat-) currency system, with only a few providers and millions of customers.

Triggered by the financial crisis in 2008, the first fully decentralized digital currency was created, which manages to facilitate trust amongst its participating actors without the need for third parties or intermediaries: *Bitcoin*. Put simply, Bitcoin technologically consists of cryptographic checksums over its data combined with a decentralized consensus algorithm, which relies on the proof of computational work. Based on the accomplishments of bitcoin, a multitude of other blockchain technologies were developed through the years, which offer further functionalities. However, these technologies also only target digital currencies as its sole use case. The technological choice of implementation and their targeted primary use case in currencies prevent their usage in todays business processes, due to, among other things, the limited ability to scale.

In order to facilitate a decentralized coordination of business processes with blockchain technology a number of adjustments and technological advancements of the current concepts and existing solutions must be achieved. This dissertation will firstly identify the necessary adaptions and subsequently present the problem statement and illustrate solution strategies. The main focus will be placed upon the analysis, conception and development of concepts for the integration into business processes in the b2b-sector by using blockchain technologies with particular regard to the *decentralized coordination without the need for decision making by intermediaries*.

As a result, this work aims to develop the concept and a component-based architecture of a generalized blockchain-framework targeting the realization of business processes. The exemplary implementation of this framework will be evaluated through selected application scenarios. Specific attention will also be placed upon the optimized blockchain concepts, such as the minimization of the blockchain-history by using a modified BFT-algorithm and dependency graphs between transactions.

Danksagung

Diese Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter im Arbeitsbereich VSYS und wurde von vielen Personen begleitet, die alle in unterschiedlicher Form und Maße zu deren Gelingen beigetragen haben.

Zuerst möchte ich meinem Betreuer Prof. Lamersdorf danken, dass er diese Promotion überhaupt ermöglicht und während der gesamten Durchführung tatkräftig unterstützt hat. Auch meinem Zweitbetreuer Prof. Maalej möchte ich an dieser Stelle für seine langjährige Unterstützung auf dem Weg hierher danken.

Für Inspiration, Motivation, Denkanstöße und den gelegentlichen Schabernack haben täglich meine Kollegen Alex, Christopher, Dirk, Gabriel, Heiko, Julian, Kai, Kristof, Lars, Philipp und Volker von VSYS sowie Fabian, Felix^2, Mareike, Martin, Steffen und Wolfram von DBIS jeder für sich auf seine eigene Weise gesorgt. Vielen Dank dafür!

Ein besonderer Dank gilt an dieser Stelle Anne, denn ohne ihre kompetente Unterstützung läuft in Stellingen praktisch nichts. Auch muss ich mich entschuldigen, dass ich dich nun mit Heiko und Philipp allein lasse, aber es wird für mich jetzt Zeit das Kapitel Dissertation abzuschließen ;-)

Während meiner Projekttätigkeit bei Ponton konnte ich wertvolle Erfahrungen in der „echten Welt“ außerhalb des Elfenbeinturmes sammeln. Hierfür möchte ich mich besonders bei Michael, Manuel, Stephan, Thomas und dem Rest des Teams für die $2\frac{1}{2}$ Jahre bei Ponton bedanken.

Auch muss ich ausdrücklich bei Emily, Joana, Lukas und Torsten bedanken, dass sie mich während der häufigen Lockdown-Maßnahmen aufgenommen und mit sozialem Kontakt versorgt haben.

Zuletzt möchte ich meinen Eltern sowie Ama, Derik und Dominik danken, die mich stets bei allen meinen Unterfangen und vor allem ohne Erwartungen unterstützt haben, damit ich meinen Weg so gehen konnte wie ich wollte, getreu dem Motto:
„Er wird’s wohl machen“.

Hamburg, September 2021
Wolf Dietmar Posdorfer

Inhaltsverzeichnis

1. Einleitung

Vertrauen ist das Fundament, auf dem die heutige Zivilisation aufbaut. Aus der Notwendigkeit für ein solches Vertrauen innerhalb verschiedener Prozesse, sei es geschäftlicher oder *technischer* Natur, entwickelten sich über die Jahre immer mehr, vor allem zentralisierte Systeme. Gerade ein zentrales System schafft eine (zentrale) Quelle der Wahrheit und einen sicheren Ort der Verwaltung und Buchführung. Potentielle Dispute zwischen an derartigen Prozessen beteiligten Parteien können in einem zentralen System so immer eindeutig gelöst werden.

Besonders stark von Zentralisierung betroffen war und ist beispielsweise das globale Banken- und (Fiat-) Geldsystem. Hier stehen viele Millionen Kunden einigen wenigen vertrauenswürdigen Instanzen gegenüber, zumeist Banken oder andere Kreditinstitutionen. Gerade bei derartigen Anwendungen gestaltet sich unter anderem die starke Zentralisierung und Vernetzung dieser Intermediäre besonders dann für deren Kunden problematisch, wenn das Vertrauen in die zentralen Instanzen sinkt, wie es im Fall der globalen Finanzkrise im Jahr 2007 beobachtet werden konnte [McKibbin und Stoeckel 2010].

Das anschließende Aufkommen digitaler Währungen, Verträge oder ganzer Wertschöpfungsketten, die aus unterschiedlichem Bedürfnis ohne einen zentralen, vertrauenswürdigen Koordinator auskommen, rückt nun den Bedarf an *dezentralen* Prozessen und Systemen in den Mittelpunkt einer Vielzahl aktueller Entwicklungen und Trends.

In dezentralen Systemen ist es jedoch oft nicht trivial eine vertrauensvolle Zusammenarbeit und Entscheidungsfindung eindeutig, effizient und auch möglichst fehlertolerant zu organisieren. So bedarf es in solchen Systemen mit mehreren Parteien immer unabhängiger Dritter, die eine effektive Koordination ermöglichen und Vertrauen herstellen. Diese Beteiligung von Intermediären verursacht jedoch einerseits Kosten für die Herstellung von Konsens unter allen Teilnehmern, andererseits entstehen sog. *Single Points of Failure*, die dann aber das Gesamtsystem derartig stark einschränken können, dass keinerlei Aktionen mehr möglich sind.

Hieraus wird der Bedarf für eine *dezentrale Koordination ohne intermediäre Entscheidungsfindung* erkennbar. Diese aber existiert bisher nur in sehr begrenzten Ansätzen und deshalb steht deren Weiterentwicklung auch im Fokus der vorliegenden Dissertation.

1.1. Motivation

Die globale Finanzkrise im Jahr 2008 war mit ein Auslöser für die Entstehung einer ersten dezentralen Lösungsstrategie zur sicheren Durchführung verteilter Transaktionen ohne die Notwendigkeit einer zentralen Instanz, eine frühe Blockchain-ähnlichen Technologie. Unter dem Pseudonym *Satoshi Nakamoto* wurde dazu ein Whitepaper veröffentlicht, welches ein dezentrales Register für digitale Währungen vorstellt: *Bitcoin* [Nakamoto 2008]. Das Besondere an dieser digitalen Währung ist im Vergleich zu anderen digitalen Währungen wie beispielsweise „eCash“ [Panurach 1996], dass es ohne zentrale Instanzen auskommt. Das größte Problem bei digitalen Währungen ist das Verhindern des sogenannten *Double-Spending* (dt. doppeltes Ausgeben). Bei dessen Vermeidung geht es vorrangig darum, allen anderen Teilnehmern glaubhaft zu versichern, dass sie bei einer Transaktion von Währung auch tatsächlich die jeweils einzigen Empfänger sind und bleiben, und dass dieselbe digitale Münze nicht erneut an jemand anderen ausgegeben werden kann. In eCash und anderen Systemen wird das Double-Spending Problem durch eine

zentrale Instanz gelöst. In Bitcoin hingegen basiert das Vertrauen ineinander auf Rechenleistung und Prüfsummen. Durch das Lösen eines „kryptografischen Rätsels“ (Brute-Force Aufgabe) kann einerseits dargelegt werden, dass eine bestimmte Rechenleistung erbracht wurde (Proof-of-Work). Gleichzeitig wird durch die Bildung von Blöcken und deren Verkettung durch Prüfsummen erreicht, dass im Nachhinein keine Daten verändert werden können. Die Verifikation der Inhalte lässt sich entsprechend durch Prüfung aller Transaktionen gewährleisten.

Dieser Proof-of-Work als Vertrauensbeweis, oder manchmal auch Nakamoto-Konsens genannt, ist speziell ausgelegt für vollständig dezentralisierte Systeme in Umgebungen, in denen sich die Teilnehmer nicht vertrauen [Nakamoto 2008]. Das Vertrauen der Teilnehmer untereinander beruht hierbei komplett auf dem Vertrauen in den Proof-of-Work-Algorithmus. Wenn dabei ein Teilnehmer mutwillig ungültige Aktionen ausführt, wie beispielsweise ein Double-Spend, aber das Kryptorätsel löst, wird dies von den anderen Teilnehmern nicht akzeptiert und ignoriert. Das Gleiche passiert bei gültigen Aktionen, aber ungültiger Lösung des Rätsels. Das korrekte Berechnen eines Rätsels benötigt mittlerweile (Stand 2021) immensen Rechenbedarf und führt bei ungültigen Aktionen zu einer „Verschwendung“ dieser. Denn die Verifikation der Lösung und der Aktionen ist trivial, die Erzeugung dieser hingegen nicht.

Basierend auf den Ideen von Bitcoin sind in den vergangenen Jahren viele Weiterentwicklungen der Blockchain-Technologien entstanden. Mit der Vorstellung von *Ethereum* wurde die Möglichkeit geschaffen, auch andere Prozesse und Workflows als nur den Austausch von Währungen durch nutzergenerierte Programmlogik umzusetzen [Buterin 2014]. Diese sog. Smart Contracts erben dabei die Transparenz und Unveränderbarkeitseigenschaften der Blockchain-Technologie, unterliegen jedoch vielen weiteren Einschränkungen, die sie nicht ohne Weiteres für Geschäftsprozesse geeignet machen.

Entsprechend beschränken sich auch existierende Mechanismen zur dezentralen Koordination ohne intermediäre Entscheidungsfindung oft nur auf die

Validierung von Transaktionen und bringen für diesen Zweck wichtige, aber insgesamt negative Seiteneffekte mit sich, beispielsweise den Verbrauch von Energie zur Herstellung von Sicherheit.

1.2. Problemstellung

Eine vielfach kritisierte Eigenschaft an Blockchains ist der Proof-of-Work-Konsensalgorithmus. Dieser erzeugt in vielen Blockchain-Technologien enormen Rechenaufwand und Energieverbrauch, der ihn für den Einsatz in vielen Geschäftsprozessen ungeeignet macht [O'Dwyer und Malone 2014]. Weiterhin gibt es in Bitcoin u.a. nicht die erstrebenswerte „sofortige Finalität" der Zustände aufgrund der *Longest-Chain-Rule* und der temporären Entstehung von konkurrierenden Zuständen (Forks). Die Finalität des Zustands kann hier erst nach einer bestimmten Zeit oder bestimmten Anzahl von nachfolgenden Blöcken garantiert werden (eventual consistency), was viele Geschäftsprozesse in ihren Abläufen verlangsamen würde [Anceaume u. a. 2020]. Diese Wartezeit, von mehreren Minuten bis hin zu einer Stunde, ist zudem mit echtzeitnahen Geschäftsprozessen nicht vereinbar, sodass insbesondere hier Koordinationsmechanismen auf Basis des Proof-of-Work-Algorithmus nicht in Betracht gezogen werden können.

Eine weitere Einschränkung vorherrschender Koordinationsmechanismen betrifft die Skalierbarkeit des Systems. Mit steigender Anzahl von Nutzern beziehungsweise Transaktionen muss sichergestellt werden, dass das System weiterhin performant nutzbar bleibt. Viele der existierenden Lösungen sind allerdings nicht darauf ausgelegt, beliebig zu skalieren. So beschreibt das sogenannte „Blockchain-Trilemma" die Unfähigkeit, dass Skalierbarkeit, Sicherheit und Dezentralisierung unverändert bleiben, sollten zwei der Eigenschaften verbessert werden. Beispielsweise leidet bei der Erhöhung der Skalierbarkeit und Sicherheit notgedrungenerweise die Dezentralisierung. Für komplett offene Systeme mit unbekannten Teilnehmern ist dies oft keine Option. Bei Geschäftsprozessen hingegen ist eine vollständige Dezentralisierung

nicht zwingenderweise notwendig, da viele der Teilnehmer untereinander bereits bekannt sind.

Entsprechend bedarf es für die Integration von Blockchain-Technologien in Geschäftsprozesse und der damit resultierenden Dezentralisierung einiger Anpassungen der bisher bestehenden Technologien. Weiterhin stellen viele Geschäftsprozesse zusätzliche Herausforderungen, welche über das Double-Spend-Problem hinausgehen. Welche Herausforderungen die Blockchain-basierte Umsetzung von Geschäftsprozessen maßgeblich beeinflussen, wie diese sich lösen lassen und welche daraus resultierenden nötigen Anpassungen an den bestehenden Technologien vorzunehmen sind, soll Gegenstand und Forschungsfrage der vorliegenden Dissertation sein, die nachfolgend präzisiert wird.

1.3. Zielsetzung

In verteilten Systemen ist es von größter Wichtigkeit, dass alle Teilnehmer zu jeder Zeit die gleiche Sicht auf den gesamten Zustand besitzen. Nur so kann gewährleistet werden, dass keine ungültigen Prozesse gestartet werden, die noch auf einem alten oder ungültigen Datenstand basieren. Einen Konsens unter mehreren, auch mit fehlerhaften oder mutwillig falsch agierenden, Teilnehmern herzustellen, ist eines der fundamentalen Probleme der Informatik.

Die Zentralisierung von Prozessen und IT-Infrastruktur kann leicht helfen, dieses Konsensproblem zu lösen, denn hier wird der Gesamtzustand des Systems durch eine einzelne Instanz diktiert. Um jedoch in einem verteilten System einen Konsens herzustellen, bedarf es eines Konsensalgorithmus. Wenn dabei zusätzlich auf eine unbeteiligte Vertrauensinstanz verzichtet werden soll, der Konsens also allein zwischen den beteiligten Teilnehmern hergestellt wird, bedarf es weitergehender softwaretechnischer Unterstützung, um die dezentrale Koordination ohne Intermediäre zu ermöglichen. Existierende

Koordinationsalgorithmen, die zur Lösung dieses Problems beitragen, beispielsweise Verfahren der Schwarmintelligenz oder der Selbstorganisation, unterliegen dabei ebenfalls den vorgenannten Restriktionen und können oft nur Dezentralität oder Vertrauen gewährleisten.

Entsprechend soll im Rahmen dieser Arbeit die dezentrale Koordination mit Hilfe von Blockchain-Technologie untersucht werden, um Koordinationsstrategien auch zu ermöglichen, dass Daten unveränderlich, ausfallsicher und transparent gespeichert sowie an alle beteiligten Teilnehmer gleichermaßen diskriminierungsfrei verteilt werden, ohne dass diese sich gegenseitig vertrauen müssen. Statt einer zentralen Instanz, mit all ihren Einschränkungen, sorgt hierbei ein dezentral ablaufender verteilter Konsensalgorithmus für die notwendige Übereinstimmung der ablaufenden Prozesse zwischen den Teilnehmern.

Entsprechend ergibt sich die folgende Hauptforschungsfrage dieser Arbeit:

- Wie kann eine Softwareunterstützung für die Dezentralisierung von Geschäftsprozessen in B2B-Anwendungen auf der Basis einer geeignet angepassten Verwendung der Blockchain-Technologie so realisiert werden, dass dabei keine der spezifischen Anforderungen von Geschäftsprozessen vernachlässigt werden?

Zur Beantwortung dieser Forschungsfrage bedarf es der Beantwortung weiterer Teilforschungsfragen:

- Was sind geeignete Technologien für die softwareunterstützte Dezentralisierung von Geschäftsprozessen in B2B-Anwendungen?

- In wie weit stellt die Blockchain-Technologie dafür ein geeignetes Mittel dar? Was sind ihre technischen Grenzen und welche ihrer bisherigen Eigenschaften müssen dafür noch verallgemeinert bzw. erweitert und verbessert werden, um ihren Einsatz in derartigen realitätsnahen Anwendungen zu ermöglichen?

- ▷ Welche allgemeinen Anforderungen haben Geschäftsprozesse bei einer dezentralen Ausführung und welche zusätzlichen speziellen Anforderungen entstehen durch die Verwendung einer Blockchain-Technologie?

- ▷ Welche Einschränkungen ergeben sich bei der Verwendung von aktuell existierenden Blockchain-Technologien zur Umsetzung von Geschäftsprozessen?

- ▷ Welche Funktionalitäten muss eine Blockchain-Technologie beinhalten, um damit möglichst viele Geschäftsprozesse abdecken zu können?

- ▷ Wie muss eine Blockchain-Technologie softwaretechnisch umgesetzt werden, damit sie eine generelle Unterstützung für jegliche Arten von Geschäftsprozessen bietet?

- ▷ Welche Möglichkeiten zur Optimierung bietet eine im B2B-Bereich eingesetzte Blockchain, um Probleme der Sicherheit oder Skalierbarkeit zu minimieren?

Ziel dieser Arbeit ist es folglich, ausgehend von der aufgestellten Hauptforschungsfrage und den Teilforschungsfragen, herauszufinden, wie die Blockchain-Technologie für realistische Geschäftsprozesse genutzt und geeignet erweitert und auch technisch realisiert werden kann. Dabei gilt es sowohl zu untersuchen welche Technologien bereits existieren und wie diese dafür verwendet oder gar umfunktioniert werden können, aber auch, welche Anforderungen nicht oder nur teilweise abgedeckt werden. Weiterhin soll auch softwaretechnisch untersucht werden, welche Schlüsselkomponenten eine Blockchain-Technologie benötigt und wie diese zusammenarbeiten müssen, damit sie die speziellen Abläufe in Geschäftsprozessen unterstützen. Als weiteres Ziel der Arbeit soll ein generalisiertes Konzept entwickelt werden, um jegliche Art von Geschäftsprozessen Blockchain-basiert umzusetzen zu können und damit auch auf die Teilforschungsfragen zur Softwarearchitektur einzugehen, beziehungsweise den Einschränkungen des Blockchain-Trilemmas entgegenzuwirken.

1.4. Aufbau der Arbeit

Das nachfolgende Kapitel 2 führt zuerst in die Grundlagen von verteilten (Geschäfts-) Prozessen ein. Dabei wird einerseits dargestellt, was unter derartigen Geschäftsprozessen zu verstehen ist und wie solche auch softwaretechnisch modelliert und ausgeführt werden können. Zur fehlerfreien Verteilung von Geschäftsprozessen auf mehrere Unternehmen und Serverinfrastrukturen sind vor allem geeignete Entscheidungsfindungs- bzw. Konsensalgorithmen nötig, welche dementsprechend im zweiten Teil des Kapitels vorgestellt werden.

Im darauffolgenden Kapitel 3 werden die weiteren notwendigen technischen Grundlagen für diese Arbeit dargestellt. Unter anderem wird hier die Blockchain-Technologie ausführlicher analysiert und bewertet - wobei insbesondere ihre technischen Komponenten, Prozesse und Anwendungsgebiete dargestellt werden. Dazu werden vor allem Blockchain-spezifische Konsensalgorithmen der verschiedenen, bereits bestehenden Technologien vorgestellt und miteinander verglichen. Abschließend werden aktuelle Herausforderungen von Blockchain-basierten Umsetzungen für Geschäftsprozesse dargestellt und erläutert.

Auf diesen Grundlagen werden in Kapitel 4 existierende Blockchain-Technologien genauer auf ihre internen Funktionalitäten untersucht. Im Speziellen werden dabei ihre Eignung zur Umsetzung von verteilten Geschäftsprozessen mit besonderen Anforderungen an Vertrauens- und Sicherheitsaspekte bewertet und die Grenzen bestehender Ansätze herausgearbeitet.

Das folgende Kapitel 5 widmet sich der Analyse der Anforderungen an die Integration einer Blockchain-Technologie in Geschäftsprozessen im B2B-Bereich mit verteilten Akteuren in dezentralen Systemen. Anhand zweier Beispiele werden hierbei konkrete Anforderungen aufgezeigt, eine dazu notwendige Prozessadaption dargestellt und mögliche Lösungsansätze vorgestellt und beschrieben.

Aufbauend auf der Analyse bestehender Ansätze und Technologien sowie den in Kapitel 5 dargestellten Anforderungen von Geschäftsprozessen wird anschließend in Kapitel 6 ein generalisiertes Konzept für ein allgemeines Blockchain-Framework erarbeitet, das einen besonderen Fokus auf die Unterstützung für verteilte Geschäftsprozesse legt.

Kapitel 7 befasst sich schließlich mit der exemplarischen Umsetzung des zuvor entwickelten, erweiterten Konzeptes eines generalisierten Blockchain--Frameworks. Dazu werden zuerst verschiedene Entwicklungsparadigmen vorgestellt und daraus wird dann ein für diese Umsetzung besonders geeignetes Paradigma ausgewählt. Basierend auf dieser Auswahl wird anschließend die softwaretechnische Realisierung dargestellt, so wie sie im Rahmen der konkreten Entwicklungen dieser Dissertation exemplarisch erfolgte.

Anschließend wird in Kapitel 8 die Evaluation des vorgestellten und exemplarisch realisierten, technischen Konzepts beschrieben. Anhand ausgewählter praktischer Anwendungsszenarien und deren exemplarischer softwaretechnischer Umsetzung mithilfe des im Rahmen dieser Arbeit entwickelten Prototyps erfolgt hierzu eine qualitative Evaluation. Hierzu wird die Eignung der Unterstützung dezentraler Geschäftsprozesse in B2B-Anwendungen auf der Grundlage einer erweiterten und generalisierten Blockchain-Technologie als Koordinierungshilfsmittel in verteilten Systemen untersucht.

Kapitel 9 fasst die Inhalte, Beiträge und Diskussionen zusammen und beleuchtet dabei die in dieser Arbeit erzielten Ergebnisse und dabei jeweils auch den konkreten Beitrag zur Forschung. Abschließend werden mögliche Fortführungen dieser und verwandter Arbeiten diskutiert.

2. Grundlagen verteilter Prozesse

Der Begriff des Prozesses beschreibt einen Verlauf, Ablauf oder eine Entwicklung von Ereignissen in einem gemeinsamen Zusammenhang. Je nach der wissenschaftlichen Disziplin wird dabei aber der Prozessbegriff jeweils semantisch unterschiedlich verwendet. Deshalb soll auch zunächst geklärt werden, was im Zusammenhang dieser Arbeit unter einem Prozess verstanden wird.

In Unternehmen handelt es sich bei allen Abläufen um „Geschäftsprozesse“, welche erfüllt werden müssen, um die Unternehmensziele zu erreichen. Mithilfe von entsprechenden Prozess-Management-Systemen können Geschäftsprozesse überwacht sowie die Erreichung der Ziele verbessert werden. Sobald ein Geschäftsprozess zumindest teilweise auch automatisiert abläuft, wird dieser Teil als sogenannter „Workflow“ beschrieben, modelliert und durch geeignete Workflow-Management-Systeme unterstützt und (teil-) automatisiert ausgeführt.

Die rasante Entwicklung von Informationssystemen und die globale Vernetzung haben inzwischen dazu geführt, dass Geschäftsprozesse und Workflows nicht mehr nur noch innerhalb eines Unternehmens ausgeführt werden, sondern sie betreffen zunehmend auch mehrere Unternehmen oder Unternehmensteile; teils auch an unterschiedlichen Orten. Zudem ergeben sich bei verteilten Geschäftsprozessen und zugehörigen Prozess- und Workflow-Management-Systemen weitere Herausforderungen, u.a. hinsichtlich der Konditionen der Ausführung, der Datenerhebung, des Monitorings sowie einheitlicher globaler Zustände.

Bei der Koordination und der Erhaltung von validen Systemzuständen können u.a. verteilte Konsensalgorithmen derartige Management-Systeme unterstützen. Dabei sorgen diese Konsensalgorithmen einerseits für Fehlertoleranz, indem sie bei ausfallenden Systemen eine Einigung über den wahren Zustand herbeizuführen helfen. Im Falle von mutwillig falsch agierenden Systemen, helfen Konsensalgorithmen dabei die dann auftretenden Falschmeldungen zu identifizieren.

In diesem Kapitel werden nachfolgend zunächst die Grundlagen von Geschäftsprozessen und Workflows sowie den dazugehörigen Management-Systemen und Modellierungssprachen dargestellt. Dabei wird insbesondere auf die Herausforderungen eingegangen, die eine ständige Weiterentwicklung von Prozessen erfordern, als auch auf Herausforderungen, die bei der verteilten Ausführung von Prozessen auftreten.

Aufbauend werden dann die Grundlagen von verteilten Konsensalgorithmen erläutert. Hierbei wird zunächst auf die generellen Ziele von Konsensalgorithmen eingegangen. Hierzu werden verschiedene Stufen von Fehlertoleranzen beschrieben sowie dazugehörige ausgewählte Konsensalgorithmen. Zur Modellierung von fehlertoleranten Systemen werden schließlich Zustandsautomaten und Primary-Backup Replikationen vorgestellt.

2.1. Geschäftsprozesse

Der Begriff des *Geschäftsprozesses* beschreibt eine Menge von Aktivitäten oder Arbeitsschritten, die innerhalb einer Organisation zu einem bestimmten Zweck ablaufen. Folgende Definition wird in dieser Arbeit für Geschäftsprozesse verwendet [Weske 2007]:

> *A business process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted*

by a single organization, but it may interact with business processes performed by other organizations.

Es handelt sich bei einem Geschäftsprozess also um eine logische und zeitliche Abfolge von Schritten, die in der Vollendung einer innerbetrieblichen Aufgabe endet. Zusammengenommen formen alle Geschäftsprozesse das Geschäftsziel. Die konkrete Ausprägung der Geschäftsprozesse hängt wiederum stark von den Zielen und den innerbetrieblichen Strukturen der Unternehmen ab, wie beispielsweise die Einbindung von Informationssystemen, externen Partnern, aber auch von den Qualifikationen der ausführenden Mitarbeiter. Zusätzlich zu den innerbetrieblichen Aspekten beeinflussen auch externe Faktoren die Gestaltung der Prozesse, wie etwa rechtliche Auflagen zur Belegpflicht, Controlling oder Abrechnung. Abbildung 2.1 zeigt eine Übersicht über mögliche Einflussfaktoren auf derartige Geschäftsprozesse.

Abbildung 2.1.: Einflussfaktoren von Geschäftsprozessen [Allweyer 2005]

Mit fortschreitender Digitalisierung ist auch der Automatisierungsgrad innerhalb derartiger Geschäftsprozesse angestiegen. Der Automatisierungsgrad beschreibt den Anteil der Schritte im Prozess, die rein maschinell, also ohne menschliche Eingriffe erfolgen. Grundsätzlich wird in vielen Bereichen, besonders mit einem großen Anteil an standardisierten Prozessen, eine vollständige Automatisierung angestrebt [Staud 2006].

Sobald mehrere parallel ablaufende Prozesse im selben oder auch in anderen Unternehmen ablaufen, die voneinander abhängen oder sich gegenseitig beeinflussen, spricht man von *verteilten Geschäftsprozessen*. Ein verteilter Geschäftsprozess kann als ein dynamischer und temporärer Zusammenschluss von Geschäftsprozessen zum Zweck der Erfüllung eines Endproduktes gesehen werden. Der Zusammenschluss mehrerer Unternehmen und derer Prozesse wird in diesem Kontext auch als *Virtuelles Unternehmen* (engl. Virtual Enterprise) bezeichnet [Klen u. a. 1999].

2.1.1. Herausforderungen

Die Herausforderungen, die durch die Globalisierung und die daraus resultierenden Bedingungen für die Herstellung von Waren und das Anbieten von Dienstleistungen aufkommen, haben sich in den letzten Jahren stark verändert. Damit derartige Unternehmen wettbewerbsfähig bleiben, müssen deshalb die entsprechenden Geschäftsprozesse ständig weiterentwickelt und an neue Herausforderungen angepasst werden. Zu diesen Herausforderungen zählen unter anderem die nachfolgenden [Allweyer 2005]:

Produktzyklen Kurze Produktzyklen sind erforderlich, um in vielen Bereichen kompetitiv zu bleiben. Entwicklungs- und Einführungsprozesse müssen beschleunigt, teils auch parallelisiert werden. In vielen Bereichen, wie beispielsweise Prozessoren, erscheinen fast wöchentlich neue Modelle mit höherer Leistungsfähigkeit.

Kundenanforderungen Durch steigende Kundenanforderungen wird zusätzlicher Druck ausgeübt. Produktzyklen müssen nicht nur verkürzt, sondern auch die Qualität muss ständig erhöht werden. Auch die angestiegene Nachfrage nach individualisierten Produkten wirkt sich auf die Geschäftsprozesse aus.

Gesetze Auch gesetzliche Anforderungen oder Handels- und Produktnormen beeinflussen die Prozesse. Wenn international agiert wird, müs-

sen je nach Markt teils unterschiedliche Normen eingehalten werden. Dies erfordert, dass teilweise der gesamte Herstellungsprozess eines Produktes genauestens dokumentiert ist.

IT Entwicklung Die rasante Digitalisierung und globale Vernetzung erfordert auch eine Abstimmung der diversen Softwaresysteme, die an vielen Stellen in der Prozesskette eingesetzt werden. Eine notwendige Änderung im Prozess kann eine Anpassung in allen Softwaresystemen entlang der Lieferkette erfordern.

2.2. Geschäftsprozessmanagement

Geschäftsprozessmanagement (kurz BPM, engl. Business Process Management) ist die systematische Gestaltung, Steuerung, Überwachung und Weiterentwicklung der Geschäftsprozesse eines Unternehmens. Es umfasst das strategische Prozessmanagement, den Prozessentwurf, die Prozessimplementierung und das Prozesscontrolling [Allweyer 2005].

BPM beinhaltet also für die Ausgestaltung und den Ablauf aller betrieblichen Abläufe die Konzepte, die verwendeten Technologien für deren Design, die Implementierung, die Analyse der Prozesse und die Steuerung der Geschäftsprozesse. Im BPM wird die Gesamtheit des Prozesses betrachtet, dazu zählen auch die menschlichen Akteure, die diese ausführen, die beteiligten Systeme und Funktionen sowie externe Firmen. Die Ziele des BPM sind die Analyse, Überwachung, Optimierung, Automatisierung und das Schaffen von Transparenz in die Geschäftsprozesse, um das Unternehmensziel stetig zu verbessern [Staud 2006].

Um Tätigkeiten wie Identifizierung, Formalisierung und Analyse von Geschäftsprozessen aus der analogen Welt in die digitale zu überführen, kann ein BPM-System verwendet werden. Ein BPM-System ist erstmal eine generische Software, die durch explizite Prozessrepräsentationen zur Koordination der Ausführung von Geschäftsprozessen getrieben wird. Diese verringern

den Aufwand von bisherigen Papier- und diskussionsgetriebenen Abläufen hin zu einer digitalen Version. An die Geschäftsprozesse selbst werden dabei zunächst keine Anforderungen bezüglich ihres Automatisierungsgrades gestellt [Weske 2007].

2.2.1. Workflow Management

Wenn ein Geschäftsprozess teilweise oder vollständig automatisiert abläuft, wird dieser Ablauf auch als *Workflow* (dt. Arbeitsablauf) bezeichnet. Ein derartiger Workflow beschreibt eine spezielle Reihenfolge von Schritten und wiederholbaren Mustern von Aktivitäten. Physische oder digitale Ressourcen werden systematisch in Prozessen abgebildet, um Materialien zu transformieren, Dienste anzubieten oder Daten zu verarbeiten [EESPA 2020].

Workflows stellen also die Verfeinerung eines Geschäftsprozesses bezüglich der konkreten Einbindung von IT-Systemen dar. Da es sich bei Workflows um die technische Repräsentation von Geschäftsprozessen handelt, liegt es nahe, dass auch hier die Begrifflichkeiten und Systeme analog dazu gestaltet werden. Dementsprechend wird das *Workflow-Management* durch *Workflow-Management-Systeme* unterstützt.

Das Workflow Management wiederum ist die explizite Representation von Prozessstrukturen durch Modelle und die kontrollierte Ausführung von Geschäftsprozessen basierend auf diesen Modellen. Softwaresysteme, die das Erstellen, Bearbeiten und Ausführen von Workflows ermöglichen, werden als Workflow-Management-Systeme (WMS) bezeichnet [Weske 2007]. Die inhaltliche Kopplung von Geschäftsprozessen und Workflows sorgt auch dafür, dass sobald ein Modell eines Geschäftsprozesses besteht, ein Modell des Workflows für deren Systeme erstellt werden kann.

Workflows können in WMS unterschiedlich technisch und begrifflich modelliert sein und werden von einer entsprechenden Ausführungsumgebung ausgeführt. Die Prozesse eines Workflows können in einer solchen Umgebung

beispielsweise mithilfe von Aufgaben, Bedingungen, Zuständen und Inhalten als Fälle (engl. Cases) modelliert werden [Van Der Aalst u. a. 2004].

- *Aufgaben* beschreiben in einem solchen Workflow eine logische Einheit innerhalb des Gesamtprozesses, wie beispielsweise ein Dokument abzustempeln. Mehrere Aufgaben hintereinander ergeben so den gesamten Workflow.
- *Bedingungen* formulieren u.a., welche Aufgaben bereits erledigt wurden, und sie legen so auch fest, welche anderen Aufgaben noch zu erledigen sind. Sie definieren somit die Reihenfolge und Abschlusskriterien innerhalb eines Prozesses.
- *Inhalte* sind geschäftsprozessrelevante Daten innerhalb des Workflows, wie etwa Dokumente, Dateien, Datenbanken oder Archive. Die Inhalte der Aufgaben werden jedoch nicht vom WMS verwaltet.

Für Workflow-Management-Systeme ist es weiterhin von Bedeutung, zu welchem Grad die einzelnen Aufgaben automatisiert sind. Die Aufgaben selbst müssen nicht vollständig digital vorliegen, jedoch muss die Koordination der Aufgaben im WMS abgebildet werden, damit diese dort korrekt repräsentiert sind. Je nach Grad der Automatisierung wird nach manuellen, semi-automatischen und voll-automatischen Aufgaben unterscheiden. Zur konkreten Modellierung von Geschäftsprozessen und Workflows werden eigenständige Modellierungssprachen verwendet [Van Der Aalst u. a. 2004].

2.3. Modellierung von Geschäftsprozessen und Workflows

Sowohl zur Modellierung von Geschäftsprozessen als auch zur Modellierung von Workflows können unterschiedliche (mehr oder weniger formale) Sprachen verwendet werden, die je nach ihrer Ausrichtung unterschiedliche Nutzergruppen, technisch oder nicht-technisch, ansprechen. Hybride Modellie-

rungssprachen, die beide Nutzergruppen ansprechen sollen, müssen für die teils widersprüchlichen Anforderungen der beiden Nutzergruppen geeignet sein, was für zusätzliche Komplexität sorgen kann.

Modellierungssprachen lassen sich grundsätzlich in die drei nachfolgenden Kategorien unterteilen [Lin und Krogstie 2010]:

Informal Informale Modellierungssprachen sind für Nutzer leicht verständlich, aber für Maschinen nicht interpretierbar. Sie bestehen häufig aus natürlicher Sprache, um einen Geschäftsprozess für Menschen leicht erkenntlich zu definieren. Um Komplexität zu minimieren, wird hierbei auf jegliche Information zur automatisierten Ausführung verzichtet. Hierdurch wird weiterhin ermöglicht, dass auch Nicht-Techniker an der Modellierung teilhaben können, da keinerlei Spezialwissen erforderlich ist.

Formal Formale Modellierungssprachen zeichnen sich dadurch aus, dass jedem Element der Sprache eine formale und eindeutige Semantik zugewiesen ist. Dies ermöglicht, dass nach der Modellierung eines Prozesses, dieser direkt auf einem geeigneten System gestartet werden kann und als Workflow ausführbar wird. Damit dies aber gewährleistet werden kann, müssen formale Sprachen auch die notwendigen semantischen Details des Prozesses mit einbeziehen und sind somit komplexer als ihre natürlichsprachlichen Pendants.

Semi-Formal Semi-formale Sprachen bestehen aus einer Mischung der beiden anderen Kategorien. Ziel ist es, dass Nicht-Techniker damit umgehen können und darüber hinaus auch noch eine gewisse Form der Automation unterstützt wird. Semi-formale Sprachen ermöglichen es so beispielsweise zuerst mit einer informellen Skizze eines Prozesses zu beginnen, um diese dann fortlaufend zu erweitern, bis hin zu einem vollständig automatisierten Prozess. Nachteilig kann jedoch auch sein, dass eine semi-formale Modellierungssprache nicht dieselbe Mächtigkeit besitzt, um komplexe Workflows abzubilden. Für Nicht-Techniker

wiederum, kann sie aufgrund einiger weniger bestehender formaler Strukturregeln trotzdem zu komplex sein.

Einige der Sprachen zur Modellierung von Geschäftsprozessen und Workflows werden im Folgenden kurz aufgezeigt. Eine der ersten grafischen Modellierungssprachen sind die „Flowcharts“ (dt. Programmablaufplan, Flussdiagramm). Mit ihnen kann grafisch leicht eine Abfolge von Schritten durch geometrische Formen z.B. für Terminale, Prozesse, Entscheidungen, Ein- und Ausgaben, dargestellt werden [Gilbreth und Gilbreth 1922].

Die „Petri-Netze“ stellen eine weitere grafische Modellierungssprache dar. Während Flussdiagramme nur eine grafische Repräsentation haben, besitzen Petri-Netze zusätzlich auch eine mathematische Definition der Semantik. In Petri-Netzen können Prozesse nur mithilfe von „Stellen“ und „Transitionen“ modelliert werden. Durch „Marken“ können den Stellen auch Attribute zugeschrieben werden [Reisig 2012].

Die „Business Process Model and Notation“ (BPMN) ist eine viel genutzte semi-formale Modellierungssprache. Sie versucht also, sowohl informelle Modellierungen als auch automatisierte Ausführung von modellierten Prozessen zu ermöglichen. BPMN setzt ebenso auf grafische Elemente, die denen der Flowcharts stark ähneln. Mithilfe der „Business Process Execution Language“ (BPEL) lassen sich mit BPMN modellierte Workflows dann auch ausführen [White 2004].

Nachdem die Modellierung einer bestimmten prozessorientierten Anwendung abgeschlossen und im Falle von informalen Modellierungssprachen konvertiert wurde, können entsprechende Workflows von dazugehörigen Systemumgebungen ausgeführt werden. Dies geschieht entweder als Teil einer bestehenden Software oder durch die zur Sprache gehörigen WMS. Um die WMS zu unterstützen, beispielsweise bei der Verbindung mit anderen Systemen und Datenbanken, werden zusätzliche Komponenten benötigt. Sobald ein Workflow über mehrere Organisationen oder Einheiten hinausgeht, ist ein deutlicher Mehraufwand bei der Modellierung zur (automatisierten) Durch-

führung erforderlich. Die Eigenheiten, Herausforderungen und Gründe dafür werden im nachfolgenden Kapitel aufgezeigt.

2.4. Verteilte Ausführung von Geschäftsprozessen

Geschäftsprozesse, deren Management und auch Prozesse im Allgemeinen haben in den letzten Jahren maßgeblich die Entwicklung heutiger Softwareinformationssysteme vorangetrieben. Getrieben durch die Entwurfsmetapher des „Separation of Concerns“ (dt. Trennung der Verantwortlichkeit) und von Service-orientierten Architekturen, enthalten Enterprise Systeme nun eine Ansammlung von multiplen einzelnen Informationssystemen und Subsystemen, die es innerhalb einer oder mehrerer Organisationen zu koordinieren gilt.

Die Vernetzung vieler unabhängiger Organisationen durch gemeinsame Lieferketten oder Geschäftsprozesse verlangt auch hier übergeordnete Koordinationsmaßnahmen. Zusammenschlüsse zu virtuellen Organisationen erlauben zwar die gesamtheitliche Modellierung in Geschäftsprozess- und Workflow-Sprachen, stellen aber die Ausführung dieser vor das Problem, dass sie von mehreren Organisationen aktuelle Daten zu den dort ablaufenden Prozessschritten benötigen. Gerade die Anbindung vieler unterschiedlicher Organisationen stellt aufgrund der meist unterschiedlichen Softwaresysteme eine Integrationshürde dar [Klen u. a. 1999].

Die Koordination solcher virtueller Organisationen ist für jeden verteilten Prozess also von größter Wichtigkeit, um einen reibungslosen Prozessablauf zu garantieren. Es muss jederzeit möglich sein den Prozessstatus zu überwachen, obwohl die einzelnen Teilorganisationen eher als Blackbox zu betrachten sind, da ihre innerbetrieblichen Prozesse durchaus vor anderen Organisationen geheim verbleiben sollen. Weiterhin muss auch dafür gesorgt werden, dass bei aufkommenden Konflikten diese zeitnah und effizient gelöst werden, wofür wiederum genaue und aktuelle Informationen über den Prozessstatus erforderlich sind [Klen u. a. 1999].

Unabhängig von der Architektur des Systems ist aus Softwaresicht die *Zustandsänderung* die fundamentale Aktion innerhalb eines jeden Prozesses. Jede Veränderung im Prozess löst intern eine Zustandsänderung aus, wodurch wiederum andere Komponenten dadurch aktiviert werden. Eine Reihe von Zustandsänderungen sorgt also dafür, dass verschiedene Komponenten innerhalb des Prozesses nacheinander ihre Aufgabe erfüllen. In verteilten Prozessen kann also auch davon ausgegangen werden, dass eine Komponente B erst ihren Teil übernimmt, sobald die vorherige Komponente A ihre Zustandsänderung persistiert hat. Um für eine korrekte Ausführung zu sorgen, müssen dabei folgende drei Eigenschaften erfüllt sein [Masternak und Pobiega 2020]:

Zuverlässigkeit Zustandsänderungen und Nachrichten zwischen Komponenten innerhalb des Prozesses müssen zuverlässig übermittelt werden und dürfen nicht verloren gehen.

Deduplikation Ein wiederholtes Zustellen von alten Nachrichten sollte einerseits vermieden werden, andererseits sollte keine wiederholte Zustandsänderung erfolgen. Das erneute Empfangen einer alten Nachricht hat folglich keinen Effekt.

Kausalität Die Kausalität aller Nachrichten muss gegeben sein. Eine Zustandsänderung darf nicht vor der Zustandsinitialisierung übermittelt werden, damit die semantische Reihenfolge aller Ereignisse eingehalten wird und valide bleibt.

Um sowohl die Prozessdurchführung, Prozessüberwachung als auch die Konfliktlösung zu gewährleisten, wird meist eine zentrale Instanz innerhalb der (virtuellen) Organisation verwendet, die möglichst uneingeschränkten Zugriff auf alle Abläufe und Teile des Prozesses hat. Dieser Koordinator kann somit fürs Monitoring, Konflikterkennung, reaktive Entscheidungen, Simulation und Beurteilung von Alternativen sowie zur Analyse der virtuellen Organisation selbst eingesetzt werden [Klen u. a. 1999].

Um für Ausfallsicherheit zu sorgen, werden in heutigen Computersystemen oft redundante Komponenten verwendet, die genau dann ihre Arbeit aufnehmen, wenn die Hauptkomponente ausfällt. Um sowohl zwischen mehreren Hauptkomponenten als auch zwischen den Repliken für konsistente Zustände zu sorgen, können in verteilten Systemen, ohne einen alleinigen zentralen Koordinator, Konsensalgorithmen eingesetzt werden. Geschäftsprozesse und auch andere Prozesse profitieren dabei auch durch den Einsatz von Konsensalgorithmen, indem diese unter bestimmten Annahmen für einheitliche Zustände sorgen. Das nachfolgende Kapitel gibt deshalb einen Überblick über die damit verbundene Konsensbildung im Allgemeinen, Fehlertoleranz und bestimmte Ausprägungen und Eigenheiten einiger Algorithmen in diesem Zusammenhang.

2.5. Verteilter Konsens

Das Konsensproblem ist eines der fundamentalen Probleme in verteilten Systemen mit dem Ziel, ein zuverlässiges Gesamtsystem mit gültigem Systemzustand auch unter Einflussnahme von fehlerhaften und schadhaften Teilnehmern zu wahren und sich letztendlich auf einen gemeinsamen Wert zu einigen [Lamport u. a. 1982]. In der heutigen IT-Landschaft, in der fast ausschließlich die Infrastruktur in Cloud-Rechenzentren ausgelagert wird, werden beispielsweise, um die Ausfallsicherheit in Rechenzentren zu erhöhen, mittlerweile statt eines einzigen Servers meist mehrere dezentrale Server an verschiedenen Standorten verwendet. Dabei findet zwischen derartigen Servern jeweils ständig sowohl eine Replikation als auch eine Synchronisation der verschiedenen Systemzustände statt.

Um dabei aber dennoch unabhängig voneinander, z.B. bei oder nach Systemausfällen, einen fortlaufend konsistenten Systemzustand zu erreichen und beizubehalten, werden so z.B. bei verteiltem Rechnen, verteilten Datenbanken aber auch in Multiagentensystemen Konsensalgorithmen eingesetzt. Für Geschäftsprozesse und alle anderen Prozessen ist es dabei von besonderer

Wichtigkeit, dass alle beteiligten Komponenten zu jedem Zeitpunkt einen Konsens über den Gesamtzustand des Systems haben.

Damit so alle beteiligten Agenten oder Prozesse in einem verteilten System gemeinsam ein Problem lösen können, müssen sie sich zunächst auf ein gemeinsames Vorgehen einigen. In einer perfekten Welt wären alle Teilnehmer ständig erreichbar und würden sich gemäß der aufgestellten Erwartungen und den vereinbarten Regeln verhalten. Da eine solche perfekte Welt aber nicht existiert, müssen Konsensalgorithmen resilient oder fehlertolerant gegenüber Ausfällen (Crash-Fault) und arbiträren Fehlverhalten (Byzantine-Fault) sein.

Leicht formalisiert dargestellt schlagen bei einem Konsensproblem alle korrekten Prozesse p einen Wert v vor und müssen darauf eine einstimmige und irreversible Entscheidung über einen Wert treffen, der im Bezug zu den vorgeschlagenen Werten steht [Fischer 1983].

Ein Konsensalgorithmus für praxistaugliche Systeme erfüllt daher meist die folgenden Eigenschaften [Chandra und Toueg 1996; Lynch 1996]:

Integrität (Integrity) Wenn alle korrekten Prozesse denselben Wert v vorschlagen, müssen sie sich auch für v entscheiden.

Gültigkeit (Validity) Wenn ein korrekter Prozess sich für einen Wert v entscheidet, hat mindestens ein Prozess v vorgeschlagen.

Terminierung (Termination) Alle korrekten Prozesse entscheiden sich irgendwann.

Übereinstimmung (Agreement) Alle korrekten Prozesse entscheiden sich für denselben Wert.

Häufig werden in der Literatur die Eigenschaften Integrität und Gültigkeit zusammengefasst und als eine Eigenschaft betrachtet. Konsensalgorithmen, die die Eigenschaften Integrität/Gültigkeit, Terminierung und Übereinstimmung erfüllen, werden zugesprochen das Konsensproblem zu lösen.

Wenn zwei unterschiedliche Prozesse bereits fehlerfrei miteinander agieren und kommunizieren, dann sollte es bei Hinzunahme eines Konsensalgorithmus nicht dazu kommen, dass diese sich auf einen unterschiedlichen Wert einigen. Sollte es allerdings doch passieren, heißt das, dass der Konsens nicht die **Safety**-Eigenschaft von verteilten Systemen erfüllt. Ebenso muss sichergestellt werden, dass der Konsens zu irgendeiner Zeit stattfinden muss. Da Prozesse in einem verteilten System kontinuierlich Nachrichten austauschen und gegenseitig ihren Zustand verändern, muss zudem dafür gesorgt werden, dass keine Endlosschleifen oder Deadlocks eintreten und alle Prozesse immer einen finalen Zustand erreichen. Wenn also Prozesse untereinander einen Konsens erreichen können, wird von der **Liveness**-Eigenschaft gesprochen.

Zusammen genommen sorgen also die Safety- und Liveness-Eigenschaft dafür, dass keine ungültigen Werte entstehen und dass respektive ein gültiger Wert irgendwann auch eindeutig beschlossen wird.

2.5.1. Fehlertoleranz im Allgemeinen

Im Allgemeinen beschreibt Fehlertoleranz die Eigenschaft eines Systems sich auch während des Auftretens von Fehlern in einer seiner Komponenten korrekt zu verhalten. Ziel ist es dabei soweit möglich, die jeweils gleiche Funktionalität unter möglichst gleichen Bedingungen, wie z.B. Durchsatz oder Berechnungszeiten, sicherzustellen. Das heißt, dass das Gesamtsystem als Ganzes weiter funktioniert [González u. a. 1997].

Eine weit verbreitete Möglichkeit für Fehlertoleranz zu sorgen ist Redundanz. Dabei kann unterschieden werden zwischen verschiedenen Formen der Redundanz, wie Informationsredundanz, Hardwareredundanz, Softwareredundanz und Zeitredundanz.

Zeitredundanz ermöglicht einem System primär die Erkennung von vorübergehenden Fehlern. Da bei einmaliger fehlerhafter Ausführung einer Ope-

ration nicht sicher festgestellt werden kann, ob es sich um ein temporäres oder ein permanentes Problem handelt, hilft es deshalb, eine Operation im Fehlerfall mehrfach auszuführen. Hierbei muss sichergestellt werden, dass die zu verifizierende Operation die gleichen Ausgangsbedingungen hat wie bei der ersten Ausführung [Johnson 1984].

Softwareredundanz ist das Hinzufügen von zusätzlicher Software, um Ergebnisse zu verifizieren, Berechnungen erneut durchzuführen oder einer kompletten Replikation [Johnson 1984]. Die komplette Replikation von Software wird häufig in lebenswichtigen Systemen eingesetzt, wie z.B. in der Luftfahrtbranche. Hier wird von zwei (oder mehr) verschiedenen Teams unterschiedliche Software entwickelt, die dieselbe Aufgabe lösen soll.

Hardwareredundanz ist die komplette Replikation der Hardware, wie es in vielen Rechenzentren gemacht wird. Dazu kann entweder passiv, aktiv oder hybrid repliziert werden. In passiver Replikation werden Komponenten beispielsweise in Triplets zusammengefasst und auf deren Ergebnisse ein einfaches Mehrheitsvotum angewendet, was wiederum Einzelfehler maskiert. Um Fehler zu erkennen und sich davon zu erholen, werden in aktiver Replikation, z.B. mit der „Pair-and-Spare“ (Paar und Ersatz) Konfiguration, auf zwei Komponenten Berechnungen durchgeführt, deren Ergebnisse verglichen und im Fehlerfall eine dritte Komponente aus dem Standby geholt, um zu erkennen, welche Komponente fehlerhaft war, aber auch gleichzeitig den Wert der anderen zu bestätigen [Johnson 1984].

Informationsredundanz ermöglicht das Erkennen oder teilweise Rekonstruieren von Nachrichten durch Fehler-Erkennungs-Codes, die als Zusatzinformationen zur eigentlichen Nachricht angehängt werden. Hierzu können Paritätsbits, Hamming-Codes oder Hash-Werte benutzt werden [Johnson 1984].

Die komplette Replikation von Informationen (Datenredundanz), z.B. in einer *RAID-1* Festplattenkonfiguration, in der die Daten zweier Datenträger gespiegelt werden, sorgt dafür, dass Teile komplett ausfallen können, ohne dass Informationen verloren gehen. Um eine Replikation zwischen verschiedenen Komponenten zu ermöglichen, werden Konsensalgorithmen ein-

gesetzt, wenn auf eine zentrale Komponente zur Replikationskontrolle verzichtet werden soll, oder wenn alle Teile des Systems autonom voneinander agieren sollen.

2.6. Modellierung fehlertoleranter Systeme

Um Systeme tolerant gegenüber Fehlern jeglicher Art zu machen, werden erforderliche Ressourcen redundant angelegt, sodass bei Ausfällen Teile weiter verfügbar sind. Auch wenn diese nicht zwangsläufig entscheidungsfähig bleiben, gewährleistet es weiterhin in gewissem Maße deren Verfügbarkeit. Um fehlertolerante Systeme mit einem verteilten Konsens zu modellieren, kann beispielsweise auf Zustandsautomaten oder auf Replikation gesetzt werden.

2.6.1. Modellierung als Zustandsautomat

Der wohl am weitesten verbreitete Ansatz zur Modellierung von verteiltem Konsens beruht auf der Zuhilfenahme von replizierten Zustandsautomaten (engl. *Replicated State Machine, RSM*). Bei diesen wird jeweils ein deterministischer Zustandsautomat auf mehrere Prozesse repliziert, sodass dieser als einzelner Zustandsautomat fungiert, obwohl einige der verteilten Prozesse ausfallen [Schneider 1990].

Ein Zustandsautomat wird von einer Menge von Eingaben angetrieben, die je nach ihrer Gültigkeit, entweder zu einem neuen Zustand führen oder gar nicht erst auf den Ausgangszustand angewendet werden. Das System stellt Aktionen bereit, mit denen der Zustand ausgelesen und verändert werden kann. Eine Menge von Aktionen, die eine konsistente Transformation des Zustandes darstellen, können zu einer *Transaktion* zusammengefasst werden. Transaktionen erhalten dabei immer einen konsistenten Systemzustand, das heißt, dass sie sich an die jeweiligen Regeln halten müssen und nur kon-

sistente Zustände zu anderen konsistenten Zuständen transformieren können. Im Allgemeinen sind Transaktionen von atomarer Natur, entweder werden alle Aktionen ausgeführt und die Transaktion wird als *Commit* im Ganzen ausgeführt, oder keine der Aktionen wird ausgeführt und die Transaktion wird abgebrochen. Ein etwaiger „Zwischenzustand“ existiert nicht [Gray u. a. 1981].

Die Logik zur Überführung von einem in den nächsten Zustand wird durch die Zustands-Transitions-Funktion δ angegeben, welche einen Zustand S mit einer Eingabe Σ auf einen neuen Zustand S' abbildet.

Formal geschrieben als: $\delta : S \times \Sigma \rightarrow S'$.

Die RSM, die für Konsensalgorithmen verwendet werden, verwenden typischerweise intern ein repliziertes Logbuch (Replicated Log). Dabei speichert jeder Server einzeln in seinem Log die Abfolge von Aktionen, die sein Zustandsautomat nacheinander ausführt. Jedes Log enthält dabei dieselben Befehle in derselben Reihenfolge, sodass jeder Zustandsautomat am Ende dieselben Aktionen ausführt. Da die Aktionen und Zustände deterministisch sind, sind auch die erreichten Zustände und Ausgabewerte identisch [Ongaro 2014].

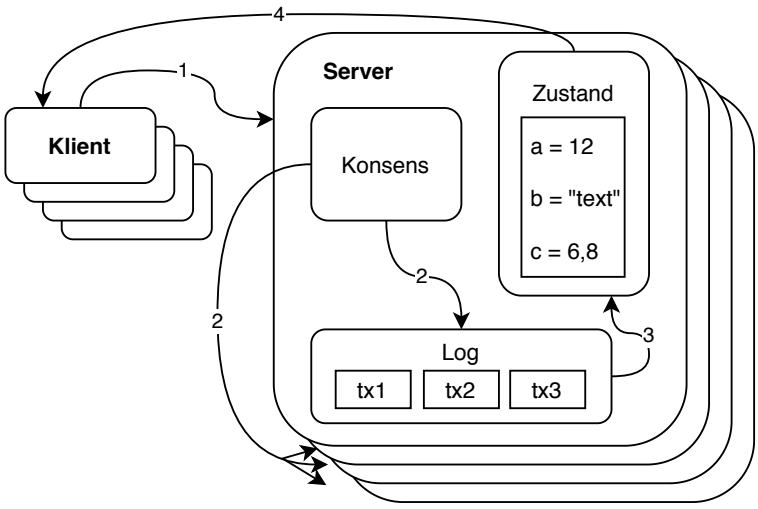

Abbildung 2.2.: Replizierter Zustandsautomat, nach [Ongaro 2014]

Damit das Log auf allen beteiligten Servern konsistent ist und bleibt, wird ein Konsensalgorithmus eingesetzt. Das zuständige Konsens Modul innerhalb der Server Software empfängt die Nachrichten von seinen Klienten und speichert sie in seinem Log. Zusätzlich kommuniziert es mit den Konsens-Modulen der anderen Server, um sicherzustellen, dass alle Logs schlussendlich dieselben Anfragen in derselben Reihenfolge enthalten, auch wenn einige Server ausfallen. So erreicht das System, dass es wie ein einziger hochverfügbarer Server erscheint [Ongaro 2014]. Abbildung 2.2 zeigt eine Beispielarchitektur für ein solches RSM System.

Damit das System ausreichend vor diversen Ausfällen geschützt ist, muss der verwendete Konsensalgorithmus sicherstellen, dass sowohl bei ausfallenden (crash fault) als auch bei mutwillig erzeugten oder unbekannten Fehlern (byzantine fault) ein Konsens hergestellt wird.

2.6.2. Modellierung als Primary-Backup-Replikation

Das Ziel der Primary-Backup-Replikation (*PBR*), ebenso wie bei RSM, ist dem Klienten zu suggerieren, dass es nur ein einziges System gibt und die Ausfälle im Hintergrund versteckt bleiben. Innerhalb des Systems wird ein Server als Primary deklariert, alle anderen fungieren als Backup und sind komplette Replikas. Weiterhin senden bei PBR die Klienten ihre Anfragen nur an den Primary des Systems und auch nur dieser antwortet auf Anfragen. Sollte der Primary ausfallen, übernimmt sofort einer der Replikas [Alsberg und Day 1976; Budhiraja u. a. 1993]. Abbildung 2.3 zeigt beispielhaft ein PBR System.

Die Verfahren, wie mit Fehlern umgegangen wird, unterscheiden sich zwischen der Modellierung als RSM und der als PBR. Bei RSMs werden die Effekte von Fehlern durch Abstimmen fast vollständig versteckt. Hingegen können Ausfälle in PBRs durchaus dazu führen, dass Anfragen verloren gehen, womit dann entsprechend umgegangen werden muss. Vorteilhaft ist

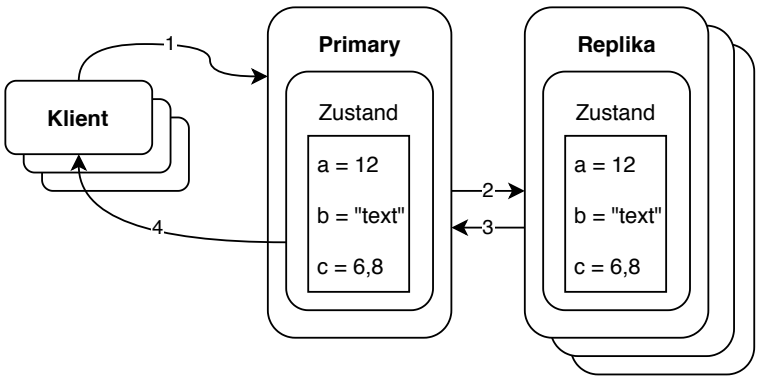

Abbildung 2.3.: Primary Backup Replication

hingegen der weitaus geringere Nachrichtenaustausch zwischen den Servern [Budhiraja u. a. 1993].

Die Systemmodellierung als PBR spielt für fehlertolerante Systeme heutzutage eine wichtige Rolle. Viele Cloud- und Rechenzentren sind so aufgebaut. Für die Modellierung von Blockchain-Systemen ist diese Form hingegen eher kontraproduktiv, da ein besonderer Wert auf Dezentralität und Autonomie gelegt wird und sich generell gegenseitig misstraut wird.

2.7. Netzwerksynchronität

Netzwerksynchronisierung spielt eine wesentliche Rolle in jedem großen Kommunikationsnetzwerk. Viele der frühen Probleme von Netzwerksynchronisierung beschäftigen sich beispielsweise mit der Synchronisierung von physisch getrennten Uhren (Knoten) über geeignete Netzwerke [Lamport 1979]. Generell können in verteilten Systemen Abläufe in synchrone und asynchrone Prozesse unterteilt werden, was bei Konsens Algorithmen nicht anders ist [Lindsey u. a. 1985].

Die Vorteile von asynchronen Systemen sind die leichtere Implementierung dieser und die deutlich höhere Ausfallsicherheit, da die Knoten nicht voneinander abhängen. Die Nachteile wiederum sind der höhere Wartungsauf-

wand und der erhöhte Nachrichtenaustausch, um Synchronität sicherzustellen [Lindsey u. a. 1985].

In Modellen von synchronen Netzwerken ist die obere Grenze Δ für Nachrichtenlaufzeiten zwischen zwei Prozessen, sowie die obere Grenze Φ der relativen Geschwindigkeiten der Prozesse bekannt. In asynchronen Systemen sind keine der beiden Grenzen bekannt, was sich besonders negativ auf Konsensalgorithmen auswirkt [Dwork u. a. 1988].

Weiterhin sind partiell-synchrone Netzwerke eine Mischform der beiden. Partielle Synchronität ist dabei entweder so festgelegt, dass zu einem bestimmten Zeitpunkt eine oder beide der oberen Grenzen für Δ und Φ existieren, die aber nicht von vornherein bekannt sind. Oder die oberen Grenzen sind von vornherein bekannt, können aber erst zu einer unbekannten Zeit in der Zukunft garantiert werden [Dwork u. a. 1988].

2.7.1. Unmöglichkeit der vollen Asynchronität

Die später im Kapitel vorgestellten Algorithmen für Crash-Fault Toleranz bzw. Byzantine-Fault Toleranz haben gemein, dass sie die Safety oder Liveness-Eigenschaft, die für dezentralen Konsens benötigt werden, nur mit Einschränkungen erfüllen können.

Das Problem dabei liegt in der Asynchronität. In synchronen Systemen kann beispielsweise das *3-Phase-Commit*-Protokoll dazu benutzt werden, um sowohl Safety als auch Liveness zu garantieren. Wenn die oberen Grenzen von Nachrichtenlaufzeit Δ und Prozessdauer Φ bekannt sind, kann mit Sicherheit bestimmt werden, wenn ein einzelner Teilnehmer ausfällt. Dasselbe gilt nicht für asynchrone Systeme. Es ist unmöglich zu bestimmen, ob ein Teilnehmer ausgefallen ist und dadurch in einen Wiederherstellungsprozess übergangen ist und wieder normal am Prozess teilnehmen wird, oder ob nur die Kommunikation ausgefallen ist, weshalb er weiterläuft und potentiell widersprüchliche Zustände entstehen.

Der Beweis der Unmöglichkeit aus dem FLP-Theorem besagt, dass es für eine Menge an Prozessen in einem asynchronen System unmöglich ist, sich auf einen binären Wert zu einigen, sogar wenn nur ein einzelner Prozess unangekündigt ausfällt [Fischer u. a. 1985].

Hierbei wird die Annahme gemacht, dass sich sowohl Prozesse asynchron verhalten, die Nachrichtenlaufzeit nie bekannt ist und Nachrichten in beliebiger Reihenfolge beim Empfänger ankommen, also das System vollständig asynchron ist. Wenn aber bestimmte Einschränkungen gemacht werden, können hingegen Safety und Liveness garantiert werden [Dolev u. a. 1987]. Von partieller Synchronität auszugehen, macht es daher für praxisnahe Systeme deutlich leichter diese Eigenschaften zu erfüllen [Dwork u. a. 1988].

2.8. Konsensalgorithmen

Prozesse, die auf mehreren Systemen stattfinden, benötigen zwingend einen gleichen Systemzustand. Ohne einen gemeinsamen Zustand wäre es vielen Prozessen gar nicht möglich korrekt abzulaufen. Wenn mehrere Akteure in einem Prozess den Zustand unabhängig voneinander und parallel zueinander verändern können, wird ein Konsens über diese Änderungen benötigt. Hierzu können in verteilten Systemen Konsensalgorithmen verwendet werden.

Ein Konsensalgorithmus sorgt in einem verteilten System dafür, dass unter einer Menge von Teilnehmern und unter Ausschluss von Ausfällen und mutwilligem Fehlverhalten, diese sich irgendwann auf einen gemeinsamen Wert festlegen. Dazu gibt es verschiedene Algorithmen, die bestimmte Eigenschaften und Verhalten haben, um das Konsensproblem in ihrem Anwendungsfall zu lösen. Dafür wird auch jeweils definiert, ab wann eine „Mehrheit“ an Teilnehmern erreicht wird, bis diese einen abgestimmten Wert anerkennt. Häufig werden die 50% Mehrheit, die $\geq \frac{2}{3}$ Mehrheit, oder die $> \frac{2}{3}$ bzw. $+\frac{2}{3}$ Mehrheit für Konsensalgorithmen verwendet. Nachfolgend werden zu den zwei Ka-

tegorien von Algorithmen einige Ausgewählte für generelle (Crash-) Fault Tolerance und Byzantine Fault Tolerance vorgestellt.

2.8.1. Crash Fault Tolerance

Unter der Kategorie von Crash Fault toleranten (CFT) Algorithmen fallen solche, die das Konsensproblem unter Einfluss von Ausfällen lösen. Ausfälle umfassen dabei sowohl Hardware- und Softwareausfälle in den Komponenten, als auch der Ausfall des Übertragungssystems, also wenn Nachrichten nicht mehr zwischen einzelnen Teilnehmern zugestellt werden können. Vor und nach dem Eintritt eines Ausfalls verhält sich die Komponente immer korrekt und sendet zu jedem Zeitpunkt auch korrekte Nachrichten.

Für gängige Algorithmen wie *Paxos* [Lamport 1998] und *Raft* [Ongaro 2014] gilt, dass sie „$2t + 1$“ Teilnehmer benötigen um t Ausfälle tolerieren zu können. Sie benötigen also eine Mehrheit von mehr als 50% aller Teilnehmer. [Keidar und Rajsbaum 2001].

Paxos

Paxos ist mittlerweile eine ganze Familie von Konsensalgorithmen. In der einfachsten Form ist Paxos für die Einigung auf einen einzelnen Wert zwischen Teilnehmern (Replicas) konstruiert worden. Die Annahmen für Paxos sind, dass Replicas zu jederzeit ausfallen können und dass Nachrichten im Netzwerk verlorengehen. Der Speicher jedes Replicas wird durch das Ausfallen nicht kompromittiert [Lamport 1998].

Tendenziell kann Paxos mit 2-Phase- oder 3-Phase-Commit Protokollen verglichen werden [Gray und Lamport 2006], jedoch blockiert Paxos nicht bei Ausfällen bzw. toleriert Netzwerkpartitionen und erfüllt somit sowohl die Safety- als auch die Liveness-Eigenschaft (mit Einschränkungen, siehe Kapitel 2.7.1).

In Paxos werden Konsensteilnehmer nach *Leader* und *Acceptor* unterschieden. Zusätzlich gibt es noch die Rollen der *Proposer* und *Learner*, diese werden jedoch in der Praxis häufig mit Leader und Acceptor kombiniert, sodass nur zwei Rollen von Relevanz sind. Ein Teilnehmer kann zu jeder Zeit entscheiden ein Leader zu werden. Ein Leader schlägt dann einen Wert vor und fordert die Übereinstimmung von diesem bei den anderen Acceptoren ein. Beim Erreichen der Mehrheit verkündet der Leader den gewählten Wert [Lamport 1998]. Der Ablauf des einfachen Paxos-Algorithmus wird nachfolgend dargestellt und ist in Abbildung 2.4 mit einem Leader (0) und drei Acceptoren (1-3) aufgezeigt.

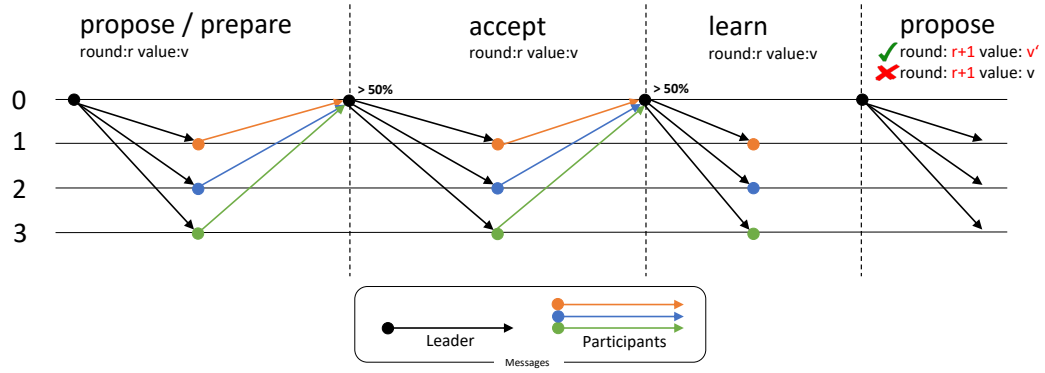

Abbildung 2.4.: Ablauf des einfachen Paxos ohne Fehlerfälle

Die Phase 1 dient der Leader Bestimmung. Der potentielle Leader schickt eine Nachricht mit einer Rundennummer an alle Teilnehmer. Darauf antworten die Teilnehmer entweder mit einer Nachricht mit ihrer aktuellen Rundennummer und dem darin vorgeschlagenen Wert, wenn sie noch keine gleiche oder höhere Rundennummer erhalten haben, oder sie ignorieren diese ganz. Zwischen Phase 1 und 2 kann der Leader anhand der Antworten bestimmen, ob ein Konsens zur aktuellen Runde mit Wert w bereits stattgefunden oder ob bisher kein Konsens gefunden wurde. In Phase 2 kann der Leader dann entweder einen neuen Wert vorschlagen, wenn noch nichts vorgeschlagen wurde, oder er schlägt den Majoritätswert vor, den er in Phase 1 erfahren hat. Acceptoren antworten auf die Nachricht aus Phase 2 mit einer Bestätigung oder ignorieren diese, wenn sie bereits in einer höheren Runde sind. In

Phase 3 bestätigt der Leader den festgelegten Wert aus Phase 2 und schließt die Runde ab [Gray und Lamport 2006; Lamport u. a. 2001].

Da im Algorithmus nicht festgelegt ist, welcher Teilnehmer zu welcher Zeit Leader ist, können zur gleichen Zeit mehrere Vorschläge im Netzwerk kursieren. Um noch einen Konsens zu erreichen, ist es wichtig, dass Acceptoren sich auf einen Wert festlegen und den anderen ablehnen (Phase 1+2). Weil Paxos nur eine $> 50\%$ Mehrheit benötigt und sich Acceptoren immer korrekt verhalten, wird irgendwann ein Konsens erreicht, auch wenn mehrere Runden von Abstimmungen benötigt werden [Chandra u. a. 2007].

Für ein System, in dem sich kontinuierlich auf neue Werte geeinigt werden muss, erzeugt der einfache Paxos einen unnötigen Nachrichten Overhead durch die erste Phase, in der ein Leader jedesmal neu erfahren muss, ob er sich als Leader durchsetzen würde, bevor er mit Phase 2 starten kann. Wenn der Leader also sehr zuverlässig ist oder sowieso der einzige der Vorschläge tätigt, wird Phase 1 unnötig. In **Multi-Paxos** wird folglich erlaubt, dass ein Leader die Phase 1 überspringt, wenn er in der vorigen Phase bereits Leader war. Um einen aktuellen Leader abzulösen, kann jederzeit ein anderer Teilnehmer eine Phase 1 Nachricht mit einer höheren Rundennummer schicken, um selbst Leader zu werden [Chandra u. a. 2007].

Raft

Obwohl die Funktionalität und die erfüllten Konsenseigenschaften des Paxos-Algorithmus mathematisch bewiesen sind, ist es basierend auf seiner Beschreibung nur sehr schwer möglich, ein reales System umzusetzen und der implementierte Algorithmus weicht meist in Teilen vom Beschriebenen ab [Chandra u. a. 2007]. Ebenfalls sind die vorgeschlagenen Multi-Paxos-Versionen nicht einheitlich. Sie weichen sowohl voneinander als auch von der ursprünglichen Version von Paxos ab [Kirsch und Amir 2008; Mazieres 2007; Van Renesse und Altinbuken 2015]. Das führt dazu, dass in realen Systemen Algorithmen verwendet werden, die lediglich auf Paxos basieren.

Basierend auf vorherigen Überlegungen zu CFT Algorithmen wurde Raft mit dem Fokus der praktischen Umsetzbarkeit entwickelt. Der Raft Konsens teilt sich ebenfalls in mehrere Schritte und Rollen. Teilnehmer haben entweder die Rolle *Leader*, *Follower* oder *Candidate*. Während der fehlerfreien Ausführung gibt es nur exakt einen Leader und beliebig viele Follower. Der Leader ist als einziger aktiv im System und beantwortet Anfragen von Klienten, während Follower nur passiv auf Aktionen vom Leader reagieren. Sobald der Leader ausfällt, können potentiell alle Follower zu Candidates werden und über den nächsten Leader abstimmen. Hierbei gilt wieder die Annahme, dass sich alle Teilnehmer korrekt verhalten. [Ongaro und Ousterhout 2013, 2014].

2.8.2. Byzantine-Fault-Tolerance

Byzantinische Fehlertoleranz (BFT) steht für die Eigenschaft, sowohl Ausfälle und Nachrichtenverluste als auch arbiträres Verhalten, darunter auch mutwillig negatives Verhalten, tolerieren zu können.

Häufig wird für die Beschreibung des zugrundeliegenden Problems die Analogie der „Byzantinischen Generäle“ verwendet . Eine Anzahl von n Generälen der byzantinischen Armee belagern eine Stadt, jedoch befinden sich unter ihnen m Verräter (byzantinische Fehler). Das Ziel der Generäle ist es, gemeinsam einen einstimmigen Plan bezüglich Angriff oder Rückzug (binäre Entscheidung) zu formulieren, bei dem immer ein Rückzug stattfindet, wenn kein Konsens erzielt wird. Die Generäle können nur mittels Nachrichten miteinander kommunizieren, aber der Übertragungsweg könnte kompromittiert werden, ohne dass es bemerkt wird. Zusätzlich können Verräter widersprüchliche Nachrichten an verschiedene Generäle senden, um den Konsens maximal zu stören [Lamport u. a. 1982].

Mit Ausfall-Fehlern kann wesentlich leichter umgegangen werden, da Systeme einfach wegfallen und nicht andere Systeme „belügen“ können. Dass dieselben Teilnehmerzusammensetzungen von $2f + 1$ bei CFT zur Fehlertoleranz in einem BFT Szenario nicht passen, zeigt folgendes Beispiel sehr ein-

fach. Um bei drei Teilnehmern eine >50% Mehrheit zu erreichen, genügen bereits 2 Stimmen von den 3 Teilnehmern. Wenn nun von zwei fehlerfreien und einem byzantinischem Teilnehmer ausgegangen wird, kann dieser mit dem einen Teilnehmer für Wert A abstimmen und mit dem anderen für den Wert B. Beide gutartigen Teilnehmer erhalten eine 50% Mehrheit für Wert A bzw. Wert B und würden damit ihren normalen Prozess fortführen, aber in ihren Zuständen divergieren.

Weiterhin beweisen Lamport u. a., dass mindestens $n = 3f + 1$ Teilnehmer nötig sind, um f byzantinische Teilnehmer zu tolerieren. Demnach ist es auch unmöglich ein Netzwerk byzantinisch fehlertolerant zu machen, welches drei oder weniger Teilnehmer hat.

Die vorgeschlagene Lösung beruht auf der Annahme eines synchronen Netzwerkmodells. Da die Nachrichtenlaufzeiten Δ im Netzwerk bekannt sind, kann ein byzantinischer Teilnehmer die Nachrichten nur um maximal die Zeitspanne Δ verzögern [Pease u. a. 1980]. Für praxisrelevante Systeme ist diese Annahme unpraktisch, da sie die Komplexität erhöht und das System anfällig für Denial-of-Service (DoS) Attacken wird [Baudet u. a. 2019].

Practical Byzantine Fault Tolerance

Die inhärente Notwendigkeit für BFT tolerante Konsensalgorithmen in asynchronen Netzwerken, gerade auch durch die steigende Popularität des Internets zu dieser Zeit, sorgten Mitte der 90er für eine Reihe an Entwicklungen von Konsensalgorithmen, wie [Canetti und Rabin 1993], [Reiter 1996] oder [Garay und Moses 1998]. Sich auf synchrone Netzwerke zu verlassen, ist besonders für Internet-basierte Systeme schwierig, da mittels DoS-Attacken schnell die Nachrichtenlaufzeit Δ schnell überschritten werden kann [Lamport 1984].

Der von Castro und Liskov vorgeschlagene Algorithmus *PBFT* zur Lösung des Konsensproblems in Systemen mit BFT Eigenschaften verspricht, gegenüber den anderen Vorschlägen dieser Zeit, sowohl praktisch umsetzbar zu

sein als auch in asynchronen Systemen die Eigenschaften Safety und Liveness (mit Einschränkungen) zu erfüllen [Castro und Liskov 1999]. PBFT nutzt dazu eine Systemmodellierung mit Zustandsautomaten und Nachrichten mit starker Kryptografie, ohne wiederum weitere Einschränkungen auf die Netzwerktopologie oder Dauer der Nachrichtenübermittlung zu setzen, solange die Anzahl an byzantinischen Teilnehmern kleiner als $\frac{1}{3}$ ist (BFT Kriterium $3f + 1$) und die gewählten kryptografischen Algorithmen sicher sind und nicht durch den Angreifer gebrochen werden können. Speziell werden Nachrichten durch asymmetrische Kryptografie, Message Authentication Codes (MAC) und Message-Digests (Hash-Funktionen) eingesetzt, um das Fälschen oder Verändern von Nachrichten durch Angreifer zu verhindern.

Der Algorithmus unterteilt dabei die Systemkomponenten in zwei verschiedene Rollen, Primary und Backup (analog zur *Primary-Backup-Replication*). Die Kurzform des Algorithmus sieht folgendermaßen aus: Ein Klient schickt einen Request an den zuständigen Primary. Dieser verteilt den Request an alle Backups, die diesen wiederum ausführen und dem Klienten das jeweilige Ergebnis mitteilen. Sobald $f + 1$ gleiche Antworten beim Klienten eintreffen, akzeptiert dieser sie als gültig.

Die Durchführung des Konsens vom Primary mit den Replicas ist, ähnlich zu Paxos oder den 2P/3P-Commit Protokollen ([Lampson und Sturgis 1979]), in die drei Phasen *Pre-Prepare*, *Prepare* und *Commit* aufgeteilt. Abbildung 2.5 zeigt einen einfachen Ablauf des PBFT Algorithmus ohne Fehlerfälle mit einem Klienten C, dem Primary P und drei Backups 1 bis 3. Hierbei sei angemerkt, dass aus Darstellungsgründen Klienten scheinbar gleichzeitig neue Phasen beginnen, was im Ablauf des Algorithmus nicht der Fall ist. Dies dient ausschließlich der Übersichtlichkeit. Sobald eine $+\frac{2}{3}$ Mehrheit erreicht wird, tritt jeder Klient asynchron in die nächste Phase über, ohne auf andere zu warten.

Wenn der Primary von dem Klienten eine Anfrage (Transaktion) bekommt, startet er den Konsensalgorithmus und beginnt die drei Phasen, indem er die Nachricht an alle Backups weiterleitet. Anfragen werden in sogenannten

Views gruppiert, welche analog zu den Runden in Paxos verstanden werden können. Die Nummer des aktuellen Views v entscheidet welcher Knoten p aus der Menge aller Replicas $|R|$ der aktuelle Primary ist, nach $p = v \ mod \ |R|$

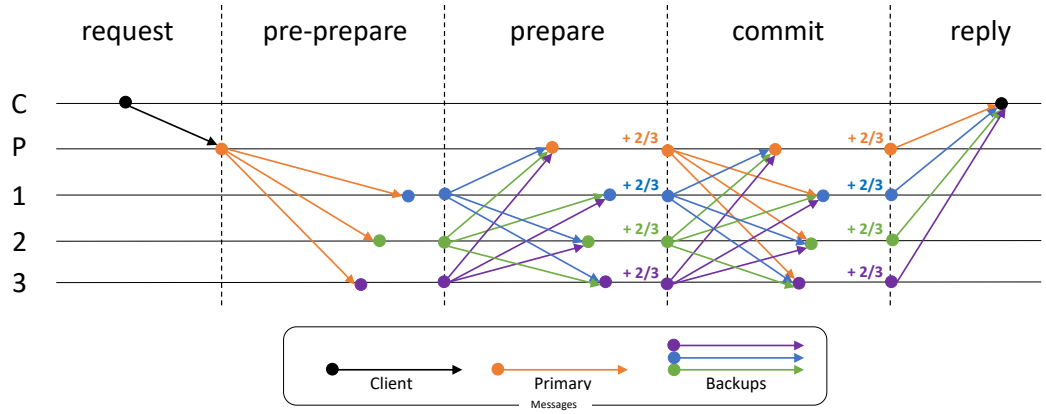

Abbildung 2.5.: Ablauf PBFT ohne Fehlerfälle

Die Pre-Prepare- und Prepare-Phasen des Algorithmus sorgen dafür, dass alle fehlerfreien Teilnehmer sich auf dieselbe Reihenfolge von Anfragen innerhalb eines Views einigen. Auf eine Pre-Prepare Nachricht des Primary folgt eine Prepare-Nachricht mit derselben Anfrage an alle weiteren Replicas. Sobald ein Teilnehmer $2f$ Prepare-Nachrichten erhält, die dieselbe Anfrage enthalten, geht er in die Commit-Phase über und versendet Commit-Nachrichten mit wiederum derselben Anfrage. Sobald ein Teilnehmer $2f$ Commit-Nachrichten erhält, führt er die Anfrage auf seinem lokalen Zustand aus und bestätigt die Ausführung dem Klienten.

Die jeweiligen Schritte erfolgen asynchron zwischen den Teilnehmern und diese wechseln die Phasen autonom immer bei erfolgreichen Erhalt von mindestens $2f$ Nachrichten der jeweiligen Phase. Um nicht unendlich lange auf Nachrichten zu warten, denn in einem asynchronen System ist die Nachrichtenlaufzeit Δ unbekannt, wird nach einer vordefinierten Zeit der aktuelle Zyklus abgebrochen und von vorne begonnen. Sollte der Primary nicht erreichbar sein, kann die Nummer des aktuellen Views erhöht werden und durch

eine View-Change Nachricht die Nutzung des nächsten Primary bestimmt werden.

Zusammenfassung

Für die Konzeption und Entwicklung von verteilten Prozessen müssen diese sowohl aus geschäftlicher als auch aus technischer Sicht betrachtet werden. Mithilfe von Geschäftsprozessen und Workflows lassen sich zwischen- und innerbetriebliche Prozesse beschreiben, modellieren und ausführen. Diese werden meist von zentraler Stelle aus koordiniert und überwacht.

Um für einen fehlerfreien Ablauf mit mehreren Teilnehmern zu garantieren, müssen Konsensalgorithmen eingesetzt werden, die unter verschiedenen Ausfall- oder Angriffsszenarien für systemweite gültige Zustände sorgen. Bei den vorgestellten Konsensalgorithmen zur Ausfallsicherheit und zur byzantinischen Fehlertoleranz nehmen einzelne zentrale Komponenten eine wichtige Rolle im System ein.

Heutzutage entsteht durch Cloud-Diensteanbieter, wie Amazon oder Google, eine starke Zentralisierung und Abhängigkeit der Infrastruktur zu diesen. Im Falle von konkurrierenden Marktteilnehmern innerhalb eines Geschäftsprozesses werden notwendigerweise Dritte oder Intermediäre als Instanz, die für Vertrauen sorgen, kostenpflichtig hinzugezogen. In beiden Fällen entsteht ein Bedarf für einen sicheren und vertrauensvollen Mechanismus, der für Konsens garantiert und unabhängig von Dritten ist, um zu einer Dezentralisierung der Prozesse zu gelangen.

3. Grundlagen der Blockchain-Technologie

Damit Geschäftsprozesse in einem verteilten System mit multiplen Akteuren auf einem gemeinsamen Systemzustand ablaufen können, ist ein Konsens über diesen zwingend notwendig. Nur so kann der korrekte Ablauf des Prozesses zu jederzeit gewährleistet werden. Bisher wurden aus diesem Grund vielfach zentralisierte Systeme eingesetzt. Hierdurch wurde die Konsensfindung des Systemzustandes auf eine dritte Instanz übertragen, deren Aufgabe auch gleichzeitig das Verhindern von Störungen durch böswillige Teilnehmer ist. Nur so konnte vermeintlich bisher erreicht werden, dass auch Teilnehmer mit gegensätzlichen Interessen innerhalb eines Systems miteinander agieren können. Die Blockchain-Technologie hingegen verspricht, eine Konsensfindung in einem verteilten System mit sich gegenseitig nicht vertrauenden Teilnehmer zu ermöglichen, ohne die Notwendigkeit einer vertrauensvollen dritten Instanz.

Eine Blockchain im Allgemeinen ist eine ständig wachsende, verkettete Liste von Datensätzen, deren nachträgliche Veränderbarkeit mittels Kryptografie verhindert wird und bei der die Gültigkeit ihrer Daten durch einen globalen dezentralen Konsens sichergestellt wird.

Technisch gesehen handelt es sich bei einer Blockchain um einen redundant replizierten Zustandsautomaten. Atomare Zustandsänderungen werden als Transaktionen innerhalb von Blöcken zusammengefasst und in eine eindeutige Reihenfolge gebracht. Die Übereinstimmung über die Reihenfolge er-

folgt über einen dezentralen und probabilistisch byzantinisch fehlertoleranten Konsens in einem Peer-to-Peer Netzwerk.

Im allgemeinen Sprachgebrauch wird der Begriff Blockchain häufig synonym für Bitcoin verwendet. Einerseits liegt dies an der starken Medienpräsenz durch den außerordentlich starken Kursanstieg des Wertes von 900$ im Januar 2017 auf 19.700$ Ende Dezember 2017 [Higgins 2017], was für Wertanlangen und starke Spekulationen an den jeweiligen Börsen sorgte. Andererseits liegt es auch daran, dass Bitcoin die erste Blockchain überhaupt war, die 2008 in einem Whitepaper von einem pseudonymen Autor veröffentlich wurde [Nakamoto 2008].

Für den Begriff Blockchain wird innerhalb dieser Arbeit folgende Definition verwendet:

> *Eine Blockchain ist ein verteilter, (vollständig) redundant replizierter Zustandsautomat, in dem eine Einigung über eine finite Ordnung von Zustandsänderungen mittels eines dezentralen Konsens innerhalb eines unorganisierten Netzwerkes erreicht und mittels kryptografischer Prüfsummen abgesichert wird.*

Bitcoin, oder Blockchains generell, lösen zwei schwierige Probleme in Computersystemen. Das Konsensproblem wird in einem Szenario gelöst, in dem sich keiner der Teilnehmer wirklich gegenseitig vertraut. Da es sich bei Bitcoin im weitesten Sinn um eine Währung handelt, ist der Anreiz einer Manipulation besonders groß. Der von Nakamoto entwickelte Konsensmechanismus *Proof of Work* (PoW) verlegt das Vertrauen der Teilnehmer auf die zur Lösung eines Rätsels aufgewendete Rechenzeit, um sowohl die Ordnung der Transaktionen zu bestimmen als auch dafür zu sorgen, dass kein Wert doppelt (Double-Spend) ausgegeben wurde.

Die Schlüsseleigenschaften, die einen Einsatz einer Blockchain ermöglicht, sind **Unveränderbarkeit**, **Transparenz**, **Dezentralität** und **Sicherheit**.

Unveränderbarkeit Alle Datensätze in der Blockchain werden durch Prüfsummen (Hashwerte) gegen Manipulation gesichert. Wird ein Datensatz

modifiziert, würde auch der Hash geändert werden müssen. Durch Verkettung der Hashes untereinander kaskadiert eine Veränderung eines Datensatzes in alle folgenden Hashes und invalidiert diese. Eine Neuberechnung aller Hashwerte ist ab einer gewissen Anzahl an Datensätzen nichtmehr praktisch durchführbar.

Transparenz Die Datensätze in der Blockchain sind für alle vollständig einsehbar. Um Transaktionen zu validieren, also deren Gültigkeit zu prüfen, muss jeder Teilnehmer die relevanten Daten einsehen können. Hierdurch wird verhindert, dass Werte doppelt ausgegeben werden können und ein Konsens erreicht wird.

Dezentralität Blockchains verwenden ein dezentrales Peer-to-Peer Netzwerk. Alle Teilnehmer im Netz sind gleichberechtigt und es gibt keine zentrale Autorität. Ausfallende Teilnehmer beeinflussen nicht das Gesamtnetz.

Sicherheit Die erhöhte Sicherheit einer Blockchain entsteht durch mehrere Faktoren. Zum einen sorgt die dezentrale Natur des Netzwerkes ohne eine zentrale Entität und vollständige Replikation der Daten für eine erhöhte Ausfallsicherheit. Wenn Hardware ausfällt oder Teilnehmer durch DoS-Attacken unerreichbar sind, läuft die Blockchain als Ganzes weiter. Zum anderen macht der Einsatz von starker Kryptografie die Daten gegenüber Angreifern sicher und nicht manipulierbar.

Um den Begriff der Blockchain weiter zu fassen, wurde der Begriff *Distributed Ledger* oder *Distributed Ledger Technology* (DLT) geprägt. DLTs beschreiben hierbei umfassend alle Technologien, die einige, aber nicht alle, Merkmale einer Blockchain innehaben. Beispielsweise gibt es DLTs, die keine lineare Verkettung von Blöcken vorsieht oder andere byzantinisch fehlertolerante Algorithmen einsetzen. Auch müssen DLTs nicht zwingend in unstrukturierten oder vertrauenslosen Netzwerken Einsatz finden. Die oben genannte Definition kann also weitestgehend für DLTs verwendet werden.

Nachfolgend werden die typischen Bausteine, Komponenten und Prozesse einer Blockchain vorgestellt. Diese Konzepte werden einerseits allgemein

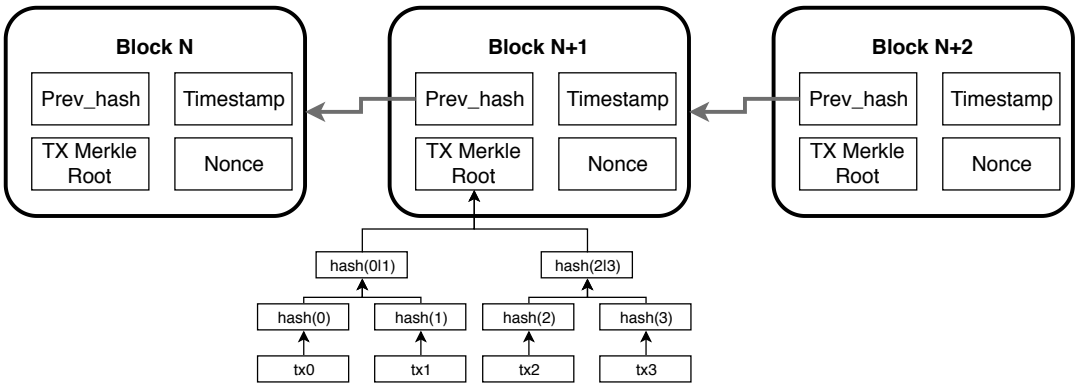

Abbildung 3.1.: Vereinfachte Darstellung der Blockchain-Datenstruktur

betrachtet, andererseits auch im Bezug auf bereits bestehende Blockchain-Systeme, wie Bitcoin oder Ethereum.

3.1. Konzept und Datenstruktur

Eine Blockchain ist eine dezentrale Datenstruktur, deren interne Konsistenz durch einen verteilten Konsens über einen gemeinsamen Anwendungszustand im Netzwerk hergestellt wird. Die Daten selbst sind dabei auf allen Knoten komplett repliziert (mit einigen Ausnahmen) und werden ständig synchron gehalten [Antonopoulos 2014].

Um den Anwendungszustand der Blockchain zu verändern, muss eine *Transaktion* erstellt und ins Netzwerk übermittelt werden. Jede Transaktion wird mit anderen in einem Block gebündelt, welcher mittels Hashwerten in Merkle-Bäumen und einem Zeiger zum vorherigen Block verkettet wird. Die Verkettung der Blöcke garantiert hierbei die Reihenfolge der Blöcke, aber auch die „Unveränderbarkeit“ dieser. Abbildung 3.1 zeigt beispielhaft die Datenstruktur der Blockchain mit drei Blöcken sowie deren Verbindung mittels Hashwerten (in Abb. als `Prev_hash`) zum vorherigen Block sowie den Root-Hash des Merkle-Baums (`TX Merkle Root`) mit den Transaktionen TX0 bis TX3.

3.1.1. Transaktion

Die elementare Datenstruktur der Blockchain sind die Transaktionen und Blöcke. Eine Transaktion ist die kleinste Dateneinheit, die verarbeitet wird und in der Lage ist, den Anwendungszustand der Blockchain zu verändern. Netzwerkteilnehmer erstellen diese und verbreiten sie im Netzwerk an andere Teilnehmer, meist durch ein Peer-To-Peer Netzwerk. Abhängig von der genutzten Blockchain-Technologie und dem gewählten Anwendungsfall können die Transaktionen beliebige Nutzdaten enthalten.

In *Bitcoin* und ähnlichen Cryptocurrencies enthalten Transaktionen die notwendigen Daten zur Übertragung der Währung. Vereinfacht dargestellt gehören dazu unter anderem die Sender- und Empfängeradressen und die Menge der zu sendenden Währung (eine detaillierte Darstellung folgt in Kapitel 3.2.1) [Nakamoto 2008].

In *Ethereum* hingegen können die Nutzdaten der Transaktion auch Funktionsaufrufe enthalten. Diese werden verwendet, um die sog. *Smart Contracts* oder den *Chain Code* auszuführen [Wood 2018]. Eine detaillierte Darstellung von Smart Contracts folgt in Abschnitt 3.2.2.

3.1.2. Block

Nachdem eine Transaktion veröffentlicht und im Netzwerk verteilt wurde, wird sie in einem Block gebündelt. Dabei wird nur der Root Hash des Merkle-Baumes, in dem die Transaktionen enthalten sind, im eigentlichen Block gespeichert. Zusätzlich zum Root Hash werden noch weitere Informationen im sog. Block Header abgelegt. Dazu gehören neben Zeitstempel und Root Hash auch der Block Hash des vorhergehenden Blockes. Je nach verwendeter Technologie gehören zu den Block Header Informationen noch zusätzliche Werte wie die Versionsnummer, Block Höhe, Target Difficulty, Gas Limit und weitere. Sämtliche Informationen aus dem Block Header werden wiederum ge-

hasht und bilden den Block Hash, welcher im darauf folgenden Block wiederum referenziert wird.

Durch die Verwendung des vorherigen Block Hashwertes werden alle Blöcke effektiv in einer großen Kette miteinander bis zum Ursprungsblock, dem Genesis-Block, verbunden. Diese Verkettung ermöglicht die Nachvollziehbarkeit über alle Transaktionen und dadurch auch den kompletten Anwendungszustand und jede Zustandsveränderung der Blockchain.

Die Zeitspanne zwischen der Erstellung eines Blockes und seines Nachfolgers wird als Blockzeit bezeichnet (engl. block time). Aus der Blockzeit und der maximalen Anzahl an Transaktionen pro Block ergibt sich der Transaktionsdurchsatz pro Sekunde.

3.1.3. Merkle-Baum

Ein Merkle-Baum oder auch Hash-Tree ist ein (meist) binärer Baum aus Hashwerten und Datensätzen. Dabei werden bei Merkle-Bäumen, im Gegensatz zu normalen binären Bäumen, Datensätze nur in den untersten Kindknoten abgelegt [Merkle 1987]. Ob die Datensätze tatsächlich in der Datenstruktur direkt oder separat abgelegt werden, hängt von der konkreten Implementation ab.

Der Elternknoten einer Transaktion bzw. eines Datensatzes setzt sich aus deren Hashwert zusammen. Der Elternknoten eines Hashwertes hingegen beinhaltet den kombinierten Hashwert seiner zweier Kindknoten. Wie in Abbildung 3.2 dargestellt, ist der Elternknoten von `tx0` der Hash `h0`. Dieser wird berechnet aus einer Hash-Funktion (z.B. SHA1) mit dem Input `tx0`. Dessen Elternknoten `h01` wiederum berechnet sich aus der Hash-Funktion mit den Inputs `h0` und `h1`. Diese Form der Verkettung setzt sich fort bis zum Wurzelknoten, dem sog. *Root Hash* oder *Top Hash*.

Die Verwendung von Merkle-Bäumen zur Verkettung von Transaktionen ermöglicht die effiziente Überprüfung der Daten auf Konsistenz und Nachwei-

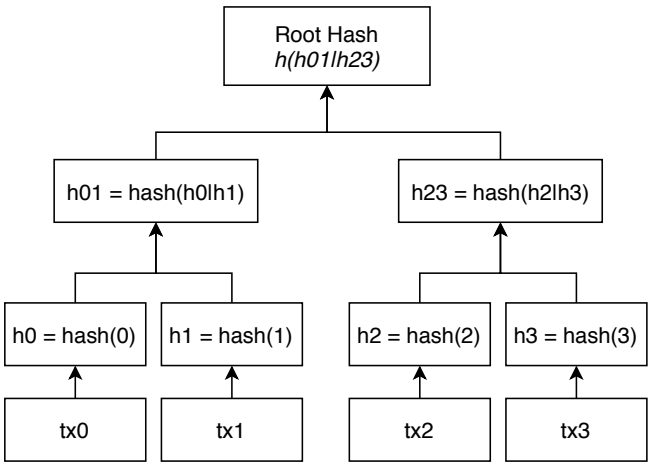

Abbildung 3.2.: Merkle-Baum mit vier Datensätzen

se über die Inklusion von Transaktionen. Da viele Blockchains, wie Bitcoin, in öffentlichen und unkontrollierten Netzwerken eingesetzt werden, kann nicht allen Nachbarknoten bedenkenlos vertraut werden. Um Abgleiche des Anwendungszustandes innerhalb des Netzwerkes zwischen Nachbarn zu beschleunigen, können Blöcke von den Nutzdaten bereinigt werden und nur mittels der Hashwerte verglichen werden. Hierdurch werden die zu übertragenden Daten drastisch reduziert und ermöglichen Abgleiche mit mehreren Nachbarn in kurzer Zeit.

Der Beweis, dass eine Transaktion in einem Block enthalten ist, wird als *Merkle-Proof* bezeichnet. Basierend auf einer Kette von Hashwerten kann nachgewiesen werden, dass eine bestimmte Transaktion Teil des Merkle-Baumes ist, ohne den kompletten Block zu übertragen. Dies ist besonders für sog. „Light Nodes“ von Vorteil, die nicht die komplette Blockchain speichern. Abbildung 3.3 zeigt visuell den Nachweis für die Inklusion der Transaktion $\texttt{tx}_\texttt{e}$.

Um zu beweisen, dass eine Transaktion enthalten ist, werden nur die Hashwerte vom Root Hash h_R bis zur jeweiligen Transaktion benötigt. Für die Transaktion $\texttt{tx}_\texttt{e}$ sieht der Nachweis demnach wie folgt aus:

$$Proof_{tx_e} = h_R, (h_x, h_y), (h_r, h_s), (h_e, h_f) \tag{3.1}$$

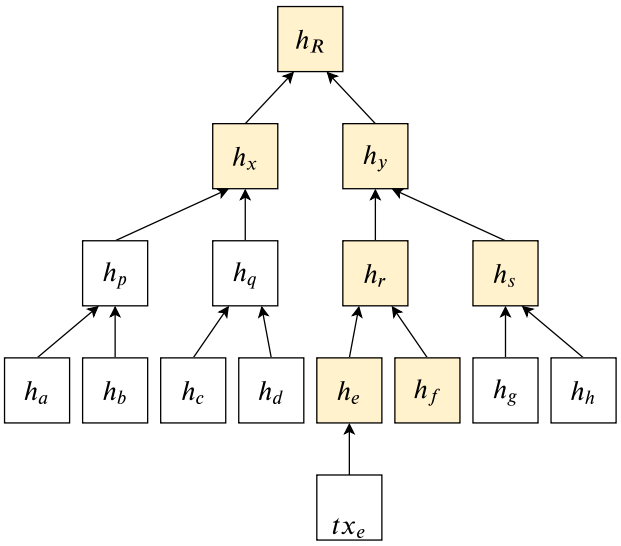

Abbildung 3.3.: Merkle-Proof für tx_e

Der Nachweis, dass eine Transaktion wiederum nicht enthalten ist, erfordert hingegen deutlich mehr Hashes. Aus den Hashwerten aller Blattknoten lassen sich die folgenden Hashes bis zum Root Hash konstruieren und somit zeigen, dass eine Transaktion nicht enthalten ist.

Das Durchsuchen eines Merkle-Baumes nach einem bestimmten Wert ist ebenfalls zeitintensiv, da er in der vorgestellten Form keinerlei Optimierungen bietet (wie z.B. bei Suchbäumen) und so der gesamte Baum traversiert werden muss.

3.2. Arten von Blockchains

Im nachfolgenden Abschnitt werden zwei der bekanntesten Blockchains und deren zugrundeliegenden Ideen und Konzepte kurz vorgestellt. *Bitcoin* gilt als der Begründer der Blockchain-Technologie. Es basiert auf einem Modell, dass einen Wertetransfer als „Unspent-Transactions“ (dt. nicht ausgegebene Transaktionen) darstellt. Basierend auf den Einschränkungen von Bitcoin und der Notwendigkeit für eine echte Turing-vollständige Ausführungsum-

gebung innerhalb von Blockchains, entstand *Ethereum*. Die Umsetzung von Smart Contracts und die Abbildung ihrer Zustände innerhalb von Ethereum machte es erforderlich auf ein Account-basiertes Transaktions-Modell zurückzugreifen.

3.2.1. Beispiel: Bitcoin

Mit Bitcoin erschien 2008 die erste Blockchain-Technologie [Nakamoto 2008]. Während im Bitcoin-Whitepaper von S. Nakamoto (Pseudonym des Autors) das Wort Blockchain jedoch nicht einführt wird, beschreibt es die genaue Konstruktion einer Kette von Blöcken mit einer durch Kryptografie verketteten elektronischen Währung, basierend auf digitalen, pseudonymen Identitäten.

Die zunehmende Zentralisierung der Banken und das durch die Weltwirtschaftskrise ([Erkens u. a. 2012]) ausgelöste Misstrauen diesen gegenüber, sorgte dafür, dass mit Bitcoin erstmalig eine Möglichkeit aufgezeigt wurde, eine dezentrale Währung ohne vertrauenswürdige Intermediäre zu entwerfen.

> *What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party.*
>
> S. Nakamoto [Nakamoto 2008]

Bitcoin setzt dabei auf ein Peer-to-Peer Netzwerk, in dem mit kryptografischen Verfahren, das Vertrauen in die Reihenfolge und die Unveränderbarkeit der Transaktionen sowie die Pseudonymität der Teilnehmer sichergestellt wird. Dabei wurden mehrere aus der Informatik bekannte Verfahren kombiniert, um eine vertrauensvolle, dezentrale Währung zu kreieren. Die Komponenten, die Bitcoin dabei kombiniert, basieren auf verketteten Timestamps bzw. verifizierbaren Logs, digitalen Währungen, Proof of Work

zur Spam Vermeidung und digitalen Identitäten basierend auf öffentlichen Schlüsseln der asymmetrischen Verschlüsselung.

Der Technologie von Bitcoin gehen digitale Währungen voraus [Panurach 1996]. In Ecash (engl. elektronisches Geld) hinterlegen Nutzer bei einer Bank eine Fiat-Währung und erhalten dafür im Austausch eine digitale Währung. Diese kann von der Bank in einen persönlichen Mint-Account (Münzpräge) überführt werden und verlässt dadurch die direkte Kontrolle dieser. Nutzer können mit ihrer Mint interagieren, um die Währung mit anderen zu tauschen. Dafür wird von der Mint Ecash abgezogen und diese mittels asymmetrischer Kryptografie verschlüsselt und dem Empfänger überstellt. Dieser entschlüsselt die Daten und überträgt den Ecash an seine Mint und wiederum weiter zurück in seinen Bank-Account.

Um zu verhindern, dass Dritte oder die Bank selbst von den Identitäten erfahren, werden Münzen mit asymmetrischer Kryptografie an Konten gebunden. Die Korrektheit der Münze kann leicht nachvollzogen werden und verhindert gleichzeitig die Offenlegung der Identität und das wiederholte ausgeben (Double-Spending). Denn sobald eine Münze doppelt verwendet wird, erschließt sich daraus auch die Identität des Besitzers [Chaum 1985; Chaum u. a. 1988].

Weil digitale Daten leicht kopiert werden können, besteht die Gefahr eines Double-Spendings, also der mehrfachen Benutzung derselben Münzen. Das Double-Spending Problem wird in Ecash dadurch gelöst, dass jede transferierte Währung mit der ausstellenden Bank verifiziert werden muss [Chaum u. a. 1988].

Die starke Asynchronität in Netzwerken durch undefinierbare, variable Nachrichtenlaufzeiten und die Schwierigkeit für synchrone Uhren zu sorgen, erfordert einen Mechanismus, um eine distinkte Reihenfolge von Aktionen festzulegen [Levine u. a. 1989]. Mithilfe von verketteten Logs kann hingegen belegt werden, in welcher Reihenfolge und zu welchem Zeitpunkt Dinge in Relation zueinander passiert sind, konkret für Blockchains bedeutet das, welcher Block auf welchen anderen Block folgt [Haber und Stornetta 1990; Mer-

kle 1987]. Dies löst in Bitcoin folglich auch das Double-Spending Problem von digitalen Währungen. Die Verkettung der Blöcke und die fest definierte Reihenfolge der Transaktionen innerhalb dieser, sorgt für eine eindeutige Reihenfolge der Transaktionen, sodass zweifelsfrei belegt werden kann, ob eine Münze bereits ausgegeben wurde. Folglich kann für jedes Währungs-Paar $(₿_A, ₿_B)$ festgestellt werden, dass Transaktion TX_A vor Transaktion TX_B stattfand und ebenso, dass die Währung $₿_A$ bereits verwendet wurde und somit $₿_B$ nichtmehr zur Verfügung steht.

Um zu verhindern, dass beliebige Teilnehmer das Netzwerk mit neuen Blöcken fluten, wird ein belohnungsbasiertes Anti-Spam System eingesetzt. Dieses erfordert die Berechnung einer Kostenfunktion, wie z.B. der Quadratwurzelberechnung modulo einer Primzahl [Dwork und Naor 1992] oder partieller Hashwert-Kollisionen [Back 2002]. Diesen Kostenfunktionen gemein ist, dass sie schwer zu berechnen, aber leicht zu verifizieren sind.

Konkret für Bitcoin sieht die Kostenfunktion vor, dass der Hash des Blockes (Vorgängerblock und Merkle-Root) und einer Zahl *Nonce* einen bestimmten Output-Wert liefert, der unterhalb einer bestimmten, aber variablen Grenze (*difficulty*) liegt.

Der Einsatz von PoW ermöglicht es Bitcoin hier, in einem Netzwerk unbekannter Größe und unbekannten, nicht vertrauenswürdigen Entitäten, einen Konsens über eine finite Ordnung von Transaktionen zu erreichen. Dies war in vorhergehenden „dezentralen Währungen“ nur möglich, indem auf vertrauensvolle, zentrale Instanzen zurückgegriffen wurde.

Die konkrete Wahl der Umsetzung der vorgestellten Verfahren, die Bitcoin verwendet, werden nachfolgend detailliert dargestellt. Der Proof-of-Work Konsensalgorithmus, den Bitcoin zur Herstellung von Vertrauen verwendet, wird umfassender in Kapitel 3.3.1 behandelt und sei hier nur am Rande im nötigen Umfang erwähnt.

Transaktionen: Unspent Transaction Output

Wie in Abschnitt 3.1.1 dargestellt, bestehen Transaktionen aus arbiträren Daten, die erst auf der Anwendungsschicht interpretiert werden, in diesem Fall die Modellierung von Zahlungstransaktionen. Bitcoin verwendet in Transaktionen für die Modellierung der Währung keine Accounts, Kontobücher oder -stände, wie klassische Banken dies vermeintlich tun, sondern benutzt die sogenannten *Unspent Transaction Outputs* (UTXO) (sinngemäß: unverbrauchte Transaktions-Ausgänge) [Antonopoulos 2014]. Um in Bitcoin seinen eigenen Kontostand zu erfahren, müssen alle Transaktionen, deren Output an einen selbst gerichtet ist, gefunden und addiert werden.

In jeder Transaktion gibt es neben der Versionsnummer und dem Wert nur zwei weitere Felder: Eingänge und Ausgänge. Die Eingänge referenzieren dabei Ausgänge aus vorherigen Transaktionen, wie in Abbildung 3.4 dargestellt. Ein Eingang wird durch den Hashwert des Blockes und dessen Position (Index) darin referenziert. Die Ausgänge sind dabei neue Münzen, die später gleichermaßen referenziert werden können.

Die *Coinbase* Transaktion ist als zusätzliche besondere Form von Transaktionen in Bitcoin erlaubt. Unterscheiden tut sich diese von den anderen Transaktionen dadurch, dass sie keinerlei Inputs enthält, sondern nur Outputs. Sie erzeugt quasi Währung aus dem Nichts und kommt in jedem Block nur einmal vor, nämlich als Belohnung für das Lösen der Proof-of-Work Kostenfunktion.

Um Eigentum der Währung zu modellieren, wird nicht etwa der öffentliche Schlüssel eines Teilnehmers direkt in jeden Ein- und Ausgang eingetragen, sondern seine „Adresse“. Eine Adresse ist ein 160 Bit langer Hashwert aus dem öffentlichen Teil eines Schlüssels.

Mit den sogenannten *Response-* und *Public-Key-Scripts* werden die Konditionen für die Verwendung der Outputs festgelegt. Innerhalb des Response-Scripts in den Eingängen werden diese Konditionen erfüllt, um diese zu verwenden. Diese Konditionen sind meist die Erzeugung einer Signatur passend

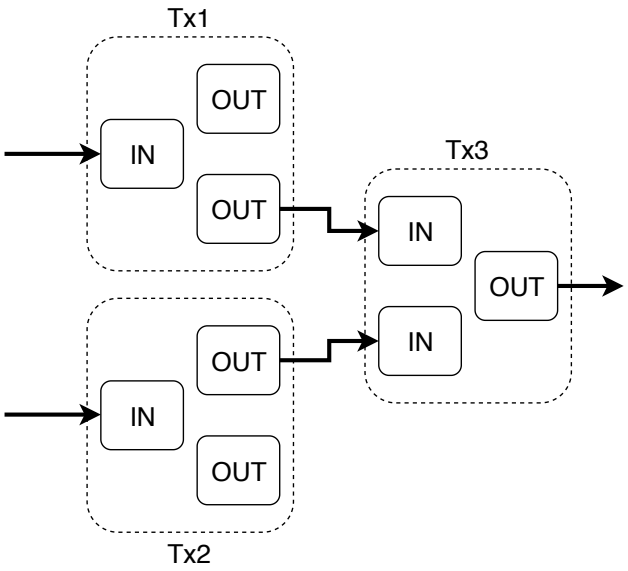

Abbildung 3.4.: UTXO Transaktionsmodell

zum öffentlichen Schlüssel der angegebenen Adresse durch die Verwendung des privaten Schlüssels. Es können mithilfe der minimalistischen Skriptsprache auch weitere Konditionen programmiert werden, jedoch ist diese nicht Turing-vollständig und in der Anzahl der Bytes beschränkt, was die Möglichkeiten für andere Anwendungsszenarien deutlich beschränkt.

3.2.2. Beispiel: Ethereum

Die Beschränkungen der Bitcoin Skriptsprache war mitunter einer der Auslöser für die Entwicklung von Ethereum [Buterin 2014]. Zusätzlich wurden die Konzepte von „Altcoins“[1], On-Chain Metaprotokollen und die Möglichkeit für Entwickler konsensbasierte dezentrale Applikationen innerhalb der Transaktionen zu kreieren, mit einbezogen.

[1] Altcoin: Zu Bitcoin meist technisch identische, aber inkompatible alternative Währungen unter neuem Namen

> *... a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications*
>
> V. Buterin [Buterin 2014]

Konkret werden im Ethereum Whitepaper ([Buterin 2014]) die folgenden Einschränkungen der Bitcoin Skriptsprache kritisiert, um echte Smart Contracts zu realisieren:

Turing-unvollständig Es werden keine Schleifen unterstützt, um nicht-terminierende Skripte während der Validierung zu vermeiden.

Werte-Blindheit Die Bitcoin Skriptsprache kann nicht zwischen den Werten der Währung der Outputs differenzieren.

Zustandslos UTXOs besitzen keinen echten Zustand oder ermöglichen dessen Modellierung in jeglicher Weise. UTXOs können entweder ausgegeben oder nicht-ausgegeben sein.

Blockchain-Blindheit Transaktionen kennen keinerlei weitere Informationen der Blockchain, wie etwa Blockhöhe, Nonce oder den Vorgänger-Hash, um z.B. irgendeine Form von Pseudozufall im Skript zu simulieren.

Accounts

Während Bitcoin das UTXO Modell zur Repräsentation der Währung verwendet und es faktisch keine Kontostände oder anderweitige Zustände gibt, wird in Ethereum ein Account-Modell verwendet. Dadurch können Transaktionen, die in Ethereum als *Messages* (dt. Nachrichten) bezeichnet werden, direkt mit Werten umgehen, ohne erst mehrere Outputs als neue Inputs zusammenzuführen.

Ein Account in Ethereum wird über eine einzigartige 20-Byte lange Adresse identifiziert. Innerhalb des Accounts werden als Zustand seine Währungsanzahl in Ether, eine Nonce, um Mehrfachausführungen von Transaktionen

zu verhindern, der *Contract Code* und die *Storage Root* festgehalten [Buterin 2014].

Zudem werden zwischen zwei Arten von Accounts unterschieden. Die *externen Accounts* sind Accounts, die durch Private Keys kontrolliert werden, also von nicht Blockchain-internen Mechanismen. Die *Contract Accounts* hingegen modellieren Smart Contracts. Innerhalb der Contact Accounts gibt es den Contract Code, der die Programmlogik darstellt und der ausgeführt wird, wenn Transaktionen an diesen gesendet werden. Ein externer Account hat hingegen keinen Code [Buterin 2014].

Die *Nonce* innerhalb des Accounts schützt diesen vor Mehrfachausführung von Transaktionen. Da es kein UTXO-Transaktionsmodell gibt, kann bei Transaktionen nicht festgestellt werden, ob ein Output (hier Kontostand) bereits verwendet wurde. Stattdessen kann eine Transaktion wie „*Überweise 200ETH von Account* `0xAA` *nach Account* `0xBB`“ beliebig oft wiederholt werden, bis der Account leer ist. Deshalb wird in jeder Transaktion die Nonce angegeben, die aktuell im Account besteht. Zur Validierung der Transaktion werden die Noncen verglichen und nach erfolgreicher Ausführung abschließend die Account-Nonce inkrementiert. Eine erneute Ausführung würde nun an der ungleichen Nonce fehlschlagen und nicht ausgeführt werden [Buterin 2014].

Smart Contracts

Die Grundidee eines Smart Contracts geht auf Szabo zurück [Szabo 1997]. Ein Smart Contract soll als digitaler Vertrag fungieren, der automatisch Vertragsbedingungen überprüft und gegebenenfalls ausführt. Durch sie soll es möglich werden, unabhängig vom Einfluss Dritter, den bisher benötigten vertrauenswürdigen Intermediären, Rechtsgeschäfte durchzuführen. Auch sollen Smart Contracts häufige Teile eines Vertrages, wie etwa Zahlungsdurchführung oder Überweisungen, automatisiert durchführen.

Als etwaiges Anwendungsbeispiel nennt er den Kauf eines Produktes an einem Getränkeautomaten [Szabo 1997]:

> *Eine Person wirft eine Münze in den Automaten ein und bekommt dafür Restgeld und ein Getränk. Das Restgeld berechnet sich aus dem Preis der Ware und der eingeworfenen Menge und dem Wert der Münzen. Physische Sicherheitsmechanismen sichern die eingeworfenen Münzen, um einen Vertragsbruch zu verhindern, etwa durch nachträgliches Entwenden der Münzen.*

Als Implementation wurde vorgeschlagen, auf virtuelle Maschinen, unveränderliche Transaktionsverläufe und asymmetrische Kryptografie zu setzen, jedoch existierte zur Zeit von Szabos Veröffentlichung kein solches System [Szabo 1997].

Mit der Blockchain-Technologie ergibt sich nun hingegen die Möglichkeit, Smart Contracts nach der Idee von Szabo umzusetzen, denn vor allem bieten sie ein vollständig dezentrales System mit einem unveränderbaren Transaktionsverlauf in Form der verketteten Blöcke. Im Blockchainkontext ist ein Smart Contract ein Programm, das innerhalb der Transaktionen gespeichert und dessen Ausführung durch den Konsensalgorithmus erzwungen wird [Luu u. a. 2016].

Um die Realisierung von möglichst vielen Programmen und Geschäftsprozessen zu gewährleisten, muss die für Smart Contracts eingesetzte Programmiersprache Turing-vollständig sein und deren Ausführungsumgebung dies unterstützen [Szabo 1997].

Smart Contracts sind so gesehen eine Form von autonomen Agenten, deren Ausführungslogik in der Blockchain gespeichert ist. Der Programmcode eines Smart Contracts wird zu Bytecode kompiliert und innerhalb einer Transaktion gespeichert. Die sogenannte *Contract Address* kann eindeutig durch die Adresse des Erstellers und dessen Transaktions-Nonce berechnet werden, ist aber auch Teil der Erstellungstransaktion. Über diese Adresse wird der Smart Contract für alle adressierbar, indem Teilnehmer Transaktionen an eben diese

Adresse schicken. Innerhalb der Transaktion sind als Nutzdaten die relevanten Aktionen codiert und führen zu seiner Ausführung [Buterin 2014].

Smart Contracts haben in Ethereum einen eigenen Zustand, *Private Storage* genannt, der ebenfalls innerhalb der Blockchain gespeichert wird. Dieser Zustand bleibt erhalten und kann konsekutiv durch Interaktionen mit dem Smart Contract verändert werden. Zusätzlich können Smart Contracts auch die Ether innerhalb der eigenen Accounts verwalten. Durch die Kombination von internem Zustand und dem Zugriff auf die Ethereum-interne Währung und verschiedene Meta-Informationen können Smart Contracts beispielsweise eigene Währungen, Glücksspiele, Crowdfunding und ähnliches erstellen, aber auch mit anderen Smart Contracts interagieren [Wood 2018].

Was Smart Contracts jedoch von echten autonomen Agenten abgrenzt, ist deren Unfähigkeit proaktiv zu agieren, sie sind immer nur reaktiv. Ein Smart Contract wird nicht von selbst aktiv, sondern muss immer durch eine Transaktion aktiviert werden. Es sind somit auch keine automatischen *Wenn-Dann* möglich, wie z.B.: „*Überweise immer montags um 12:00 200ETH an Konto* `0xAABBCCDDEEFF`, *wenn der Kontostand ausreichend hoch ist*“.

Weitere Einschränkungen von Smart Contracts betreffen die Reihenfolge der Transaktionsausführung und „Validierungs-Determinismus“. Wie auch in Bitcoin, ist in Ethereum der Miner für die Reihenfolge der Transaktionen im Block verantwortlich. Dies führt dazu, dass auch die Aktionen, die auf einem Smart Contract ausgeführt werden, in der Reihenfolge erfolgt, wie der Miner es festgelegt hat. Wenn nun zwei Nutzer gleichzeitig (bzw. innerhalb desselben Blockes) eine Aktion ausführen und vom selben Initialzustand ausgehen, trifft dies folglich nur bei der ersten Ausführung zu, die zweite Ausführung hat entweder einen anderen Ausgangszustand und ein anderes Ergebnis oder schlägt ganz fehl [Christidis und Devetsikiotis 2016].

Weiterhin müssen Inputs und Outputs von Transaktionen immer deterministisch sein, folglich können keine volatilen Daten von außerhalb einbezogen werden. Transaktionen müssen zu jeder Zeit, d.h. auch weit in der Zukunft, denselben Zustand erzeugen. Wenn nun eine Transaktion das Wetter von

„*heute*“ abfragt und für eine Berechnung verwendet, ist das Ergebnis dieser Berechnung möglicherweise zu einem zukünftigen Zeitpunkt anders, weil auch das Wetter anders ist. Deshalb werden für die Abfrage und Inklusion von externen Daten häufig Orakel-Services genutzt. Diese rufen die externen Daten einmalig ab und speichern das Ergebnis dieses Aufrufs in einer Form, die für Smart Contracts zugänglich ist und vor allem auch zukünftig zugänglich bleibt [Buterin 2014].

Der letzte Kritikpunkt an Bitcoin-Skripten war deren Turing-Unvollständigkeit. Jedoch bedeutet eine Turing-Vollständigkeit auch gleichzeitig die Möglichkeit von Endlosschleifen. Eine Endlosschleife innerhalb eines Smart Contracts würde bedeuten, dass alle Teilnehmer, die eine Transaktion zur Validierung auf diesem Smart Contract ausführen würden, für ewig blockiert werden. Um diesem Szenario vorzubeugen gibt es in Ethereum Ausführungskosten für alle Smart Contracts. Das „*Gas*“ (von engl. Gasoline, dt. Benzin) bestimmt, wie viele Operationen ausgeführt werden können. Die kompletten Gaskosten ergeben sich aus einer konstanten Anfangsgebühr von 21.000 Gas und den jeweiligen zur Laufzeit ausgeführten Operationen. Jede Operation verbraucht unterschiedlich viel Gas. Einfache Arithmetik Operationen verbrauchen 3 bis 8 Gas, wohingegen Operationen zur permanenten Speicherung von Werten 20.000 Gas verbrauchen. Eine Funktion eines Smart Contracts wird also solange ausgeführt, bis sein Gas verbraucht ist. Sollte zu wenig Gas verwendet worden sein, wird die Ausführung abgebrochen und alle Zustandsänderungen rückgängig gemacht. Eine partielle Ausführung ist nicht vorgesehen [Wood 2018].

Wenn ein Nutzer mit einem Smart Contract interagieren will, sendet er eine Transaktion an diesen und gibt zugleich an, wieviel Gas er maximal verwenden möchte und was sein Gas-Preis ist. Der Gas-Preis gibt an, wieviel Einheiten der Ethereum-Währung ETH er maximal bereit ist, pro Einheit Gas zu zahlen. Jeder Miner kann nun für sich entscheiden, ob der vorgeschlagene Gas-Preis für ihn profitabel genug ist, um die Transaktion in einen Block einzufügen oder nicht. Dies führt zu ständig fluktuierenden Gas/ETH Preisen, weil nicht alle Miner denselben Preis verwenden. Die Anzahl an aus-

geführten Operationen steht in direkter Relation zu monetären Werten und kann deshalb gerade bei komplexen Geschäftsprozessen durchaus kostspielig sein.

Um seine Geschäftsprozesse innerhalb von Smart Contracts auf der Blockchain abzubilden, muss mit mit vielen Einschränkungen gerechnet und die Prozesse gegebenenfalls auch stark vereinfacht werden. Speicherintensive Vorgänge, wie beispielsweise das Sortieren einer Liste, kosten zu viel Gas, um sie „Onchain“ ausführen zu lassen. Ebenso ist die Möglichkeit von Kryptografie eingeschränkt. Einerseits würden Ver- und Entschlüsselung zu viel Gas verbrauchen, andererseits gibt es keine Möglichkeit, die notwendigen Schlüssel geheim innerhalb des Smart Contracts abzulegen. Der Speicherbereich eines Smart Contracts wird zwar als „Private Storage“ bezeichnet, jedoch ist dieser von allen externen Parteien zugänglich. Ein hier gespeicherter privater Schlüssel ist für alle einsehbar und die Verschlüsselung mit diesen somit nutzlos.

Für die Speicherung von Zuständen der Smart Contracts und auch der Kontostände der Accounts werden in Ethereum *Merkle-Patricia-Tries* verwendet.

Merkle-Patricia-Trie

Ein Merkle-Patricia-Trie (von Re**trie**ve) ist die Kombination eines Merkle-Trees mit einem Patricia-Trie. Während im Merkle-Tree Daten in Blättern abgelegt werden und alle Elternknoten jeweils die Hashwerte aller ihrer Kindknoten darstellen, werden Patricia-Tries zur effizienten Speicherung und Suche von Daten verwendet.

Ein Patricia-Trie, oder Prefix- oder Radix-Trie, ist eine speicheroptimierte Datenstruktur, bei der sich die Schlüssel von Werten mit gemeinsamen Präfixen auch gemeinsame Pfade teilen. Anders als im Merkle-Tree werden hier alleinstehende Kindknoten direkt mit ihrem Elternknoten zusammengefasst [Morrison 1968]. Abbildung 3.5 zeigt beispielhaft einen Patricia-Trie mit fünf Schlüsseln. Je nachdem wie der Radix gewählt wird, haben in Patricia-Tries

Knoten unterschiedliche Anzahlen an Kindknoten, beispielsweise bei Bitweise unterschiedenen Schlüsseln jeweils zwei und bei einem String-basierten Trie die jeweilige Größe des Alphabets. In der Beispielabbildung ist der Radix 16, was aus der 4-bit-Datenstruktur hervorgeht.

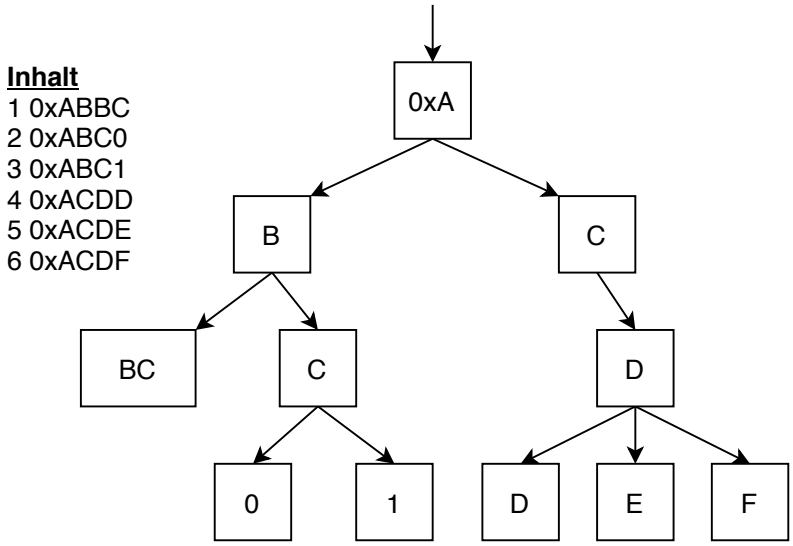

Abbildung 3.5.: Beispiel eines Patricia-Tries

Der Merkle-Patricia-Trie (MPT) kombiniert nun die Integritätseigenschaften des Merkle-Trees durch Hashwerte mit der Kompaktheit und den besseren Sucheigenschaften des Patricia-Tries. Abbildung 3.6 stellt einen vereinfachten MPT mit vier Accounts der Ethereum Implementation dar. Hier werden drei verschiedene Arten von Knoten eingesetzt:

- Ein *Extension-Node* (grün) ist ein komprimierter Pfad von gemeinsamen Präfixen. Das Prefix-Feld gibt an, ob die Anzahl an Prefix 4-bit Werten (shared nibbles) gerade oder ungerade ist.

- Ein *Leaf-Node* (violett) ist ein Kindknoten mit einzigartigem Suffix, also ohne weitere Kinder. Das Prefix-Feld wird analog zu dem im Extension Node verwendet. Das Value-Feld enthält die Account Balance, Nonce und den Hash des Storage Tries.

- Ein *Branch-Node* (blau) stellt eine Verzweigung der Präfixe mit Radix 16 dar.

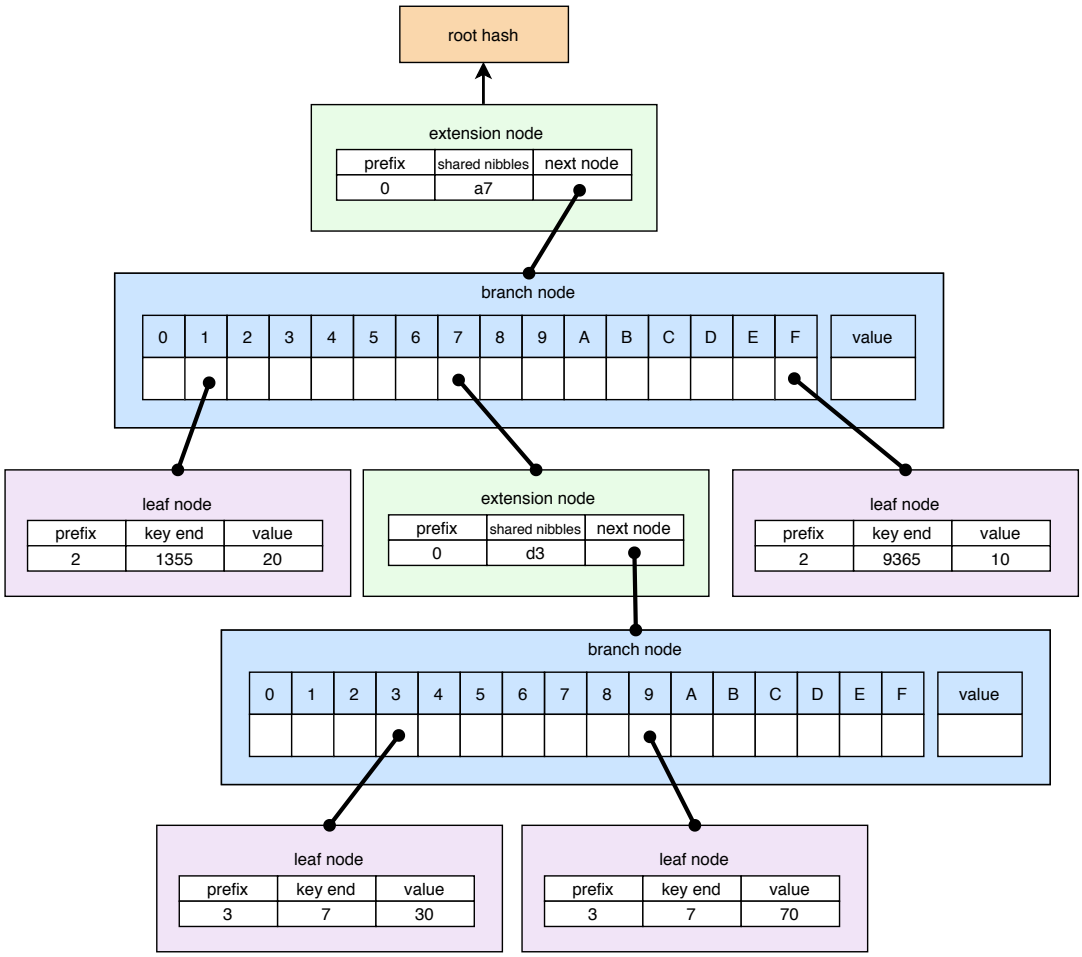

Abbildung 3.6.: Merkle-Patricia-Trie

Im dargestellten MPT befinden sich vier Accounts. Beispielsweise setzt sich die gespeicherte Adresse `A711355` mit Value `20` zusammen aus den Nibble-Werten des ersten Extension Nodes `A7`, des Branch-Nodes `1` und des Schlüsselendes des Leaf Nodes `1355`.

Zur Speicherung von Account Informationen, Transaktionen und Smart Contract Zuständen werden in Ethereum vier unterschiedliche Arten von MPTs pro Block verwendet. Der *State Trie*, bestehend aus Adresse und Account-

zustand, speichert den globalen Zustand der Blockchain. Der *Transaction Trie* speichert die Transaktionen, die in diesem Block inkludiert wurden. Der *Transaction Receipt Trie* speichert die Effekte der Transaktionen, also deren ausgelöste Zustandsveränderungen. State-, Transaction- und Receipt-Trie sind in jedem Block nur einmal vorhanden und deren Hashwerte werden direkt in den Blockhash mit einbezogen. Der vierte MPT Typ ist der *Storage Trie*, der pro Account verwendet wird, somit also mehrfach in einem Block vorkommt. Hier wird unter anderem der Smart Contract Code abgelegt. Der Roothash des Storage Tries wird innerhalb des State Tries gespeichert, im `value`-Feld des jeweiligen Leaf Nodes. Der gesamte Storage Trie wird jedoch nicht innerhalb des Blockes gespeichert, da er sich aus den bisherigen Transaktionen bei Bedarf erstellen lässt [Wood 2018].

3.3. Konsensalgorithmen in Blockchains

Das Schlüsselelement hinter jeder Blockchain ist dessen Konsensalgorithmus. Durch ihn wird sichergestellt, dass die Mehrheit der Teilnehmer denselben validen Zustand langfristig teilt. Der erste in einer Blockchain eingesetzte Konsens, Proof-of-Work oder auch *Nakamoto-Konsens* genannt, basiert auf Hashwert Kollisionen, wie sie in *HashCash* ([Back 2002]) eingesetzt werden [Nakamoto 2008]. In PoW ist es leicht die Validität zu verifizieren, aber vergleichsweise schwer einen gültigen Nachweis zu erbringen, worauf letztendlich das inhärente Vertrauen zwischen den Teilnehmern basiert. Andere Konsensalgorithmen setzten auf andere Sicherheiten, wie zentrale Autoritäten oder BFT-Mechanismen, um für Konsens zu sorgen.

Je nach verwendetem Algorithmus variiert die Zahl der zustimmenden Teilnehmer für die notwendige Mehrheit für den Konsens. In Proof-of-Work Konsensalgorithmen wird häufig 51% als erforderliche Mehrheit angegeben. Algorithmen, die auf PBFT basieren, besitzen einen Mehrheitsbedarf von $+\frac{2}{3}$, da ihr Ablauf sehr stark dem von PBFT gleicht und dieser ebenfalls eine $+\frac{2}{3}$ Mehrheit erfordert, um byzantinisch fehlertolerant zu sein.

Während sich klassische Konsensalgorithmen leicht nach ihren erfüllten Eigenschaften und Kriterien in die Gruppen CFT und BFT unterteilen lassen, gibt es bei Blockchain-Konsensalgorithmen keine weit anerkannte Gruppierung der Algorithmen. Viele der konzipierten Algorithmen erfüllen Eigenschaften, um crash-fault-tolerant zu sein, aber nicht zwingend alle, um auch byzantine-fault-tolerant zu sein. Einige Algorithmen unterscheiden verschiedene Rollen der Teilnehmer nach ihren Qualifikationen, Netzwerkvertrauen oder Aufgaben. Auch variieren die Beschränkungen bei der Netzwerkstruktur und möglicher Teilnehmerzahlen. Letztlich werden unterschiedliche Garantien bezüglich der Konsenseigenschaften *Liveness* und *Safety* gemacht, zusätzlich zu den unterschiedlichen Prioritäten *Konsistenz*, *Verfügbarkeit* und *Partitionstoleranz* des *CAP-Theorems*.

Grundsätzlich lässt sich feststellen, dass alle Blockchains das Ziel haben einen gemeinsamen Zustand über alle Teilnehmer und ein hochverfügbares Gesamtsystem herzustellen. Aus dem CAP-Theorem geht hervor, dass nur zwei der drei gewünschten Eigenschaften gleichzeitig erfüllt werden können [Brewer 2000]. Folglich könnten Blockchain-Konsensalgorithmen nach ihren bevorzugten Kriterien aus Konsistenz, Verfügbarkeit und Partitionstoleranz gruppiert werden. Hierdurch lassen sich die beiden Gruppen von „Eventual Consistency“ (dt. spätere/letztendliche Konsistenz) und „Strong Consistency“ (dt. starke Konsistenz) unterscheiden.

Weitere mögliche Unterscheidungsmerkmale ergeben sich aus der Art, wie Blöcke angehängt werden. Dieses Kriterium würde die Algorithmen in die Gruppen der „*append-based*“ und „*propose-based*“ einteilen. In append-based Algorithmen, wie dem klassischen Proof-of-Work, werden Blöcke durch Miner eigenständig validiert und an die lokale Historie angefügt. Erst hiernach werden sie im Netzwerk verteilt und durch andere Teilnehmer validiert und ebenfalls angefügt. In propose-based, also vorschlagsbasierten Algorithmen, wird ein Blockkandidat erstellt und durchs Netz propagiert. Erst danach entscheiden die Teilnehmer, basierend auf dem Blockkandidaten, ob der Block gültig ist und angehängt wird, wie es beispielsweise in *Byzantine Fault Tolerant Proof of Stake* der Fall ist. Auch kann zwischen *nachrichtenintensiven*

und *nachweisintensiven* Algorithmen differenziert werden, was zu einer ähnlichen Gruppierung führt. Unter nachweisintensiven Algorithmen (oder *Proof of X*) werden alle Algorithmen verstanden, die aus einem berechenbaren und nachweispflichtigen Aufwand hervorgehen, wie etwa dem Hashen von Werten oder Belegen von Festplattenspeicher. Nachrichtenintensive Algorithmen sind wiederum solche, die durch das Versenden von Nachrichten den nächsten Block bestimmen, meist durch Abstimmungen.

Nachfolgend werden die grundlegenden Arten von Konsensalgorithmen, die häufig verwendet werden, vorgestellt:

- Proof-of-Work (PoW), das wohl am weitesten verbreitete Verfahren. Es wird vor allem in Public Permissionless Blockchains, wie Bitcoin oder Ethereum, eingesetzt.
- Proof-of-Stake (PoS), als Nachfolger von PoW, um die Effizienz und die Ressourcenverschwendung zu minimieren.
- Byzantine Fault Tolerant Proof-of-Stake (BFT-PoS), als Konsensalgorithmus, der von klassichen BFT-Algorithmen abgeleitet, besonders in Konsortien[2] oder Private Permissioned Systemen zum Einsatz kommt.

Abschließend werden einige weitere Verfahren unter dem Sammelbegriff *Proof-of-X* kurz vorgestellt. Diese unterscheiden sich nicht wesentlich von den davor vorgestellten Verfahren, benutzen aber einige andere Mechanismen, wie beispielsweise Matrixmultiplikation anstelle von Hashwert-Berechnung.

3.3.1. Proof-of-Work

In Bitcoin und ähnlichen Technologien wird der verwendete Algorithmus als *Proof-of-Work* bezeichnet. Das Ziel von PoW ist es, Vertrauen durch eine Brute-Force Aufgabe zu erzeugen [Buterin 2014; Nakamoto 2008; Wood

[2] Konsortium: Zusammenschluss von Unternehmen, zur gemeinsamen Durchführung von Geschäftsprozessen.

2018]. Analog zu den Verfahren zur Bekämpfung von Spam-Emails ([Dwork und Naor 1992], [Back 2002]) sind PoW-Algorithmen Anreizsysteme, die auf korrektes Verhalten setzen, da bei Fehlverhalten unnötig Rechenleistung zur Erfüllung von „Vertrauensbeweisen“ verschwendet würde.

Die gestellte Aufgabe besteht darin, einen Wert (*Nonce*) zu finden, sodass der erzeugte Hashwert aus Nonce, dem Root-Hash des aktuellen Blockes ($hash_{root}$) und dem Hash des vorherigen Blockes ($hash_{B-1}$) unterhalb eines bestimmten Zieles (Target oder Difficulty) bleibt. Ungleichung 3.2 stellt die Relation zwischen Hashes und Target dar. Das Ziel wird meist angegeben als „Anzahl an führenden Nullen“, da die erzeugten Hashwerte in Hexadezimalschreibweise beispielsweise so aussehen: `0x000000000FF...AB`.

$$hash(hash_{B-1}, hash_{root}, nonce) < target \tag{3.2}$$

Das Finden der Nonce, häufig als Krypto-Rätsel bezeichnet, ist insofern aufwendig, als dass es praktisch unmöglich ist, rückwärts aus einem Output-Wert der Hash-Funktion und den übrigen Input-Werten eine Nonce zu generieren, die die Ungleichung erfüllt. Um eine Nonce zu finden, müssen also systematisch, in Form eines Brute-Forces, viele Noncen durchprobiert werden bis ein passender Hash gefunden wird. Als Anreiz an diesem Vorgang teilzunehmen, bekommt ein erfolgreicher Miner den sogenannten *Blockreward* (Coinbase-Transaktion) in Form der integrierten Währung und zusätzlich alle Transaktionsgebühren der im Block inkludierten Transaktionen.

Um weiterhin zu verhindern, dass durch das Hinzufügen von zusätzlicher Rechenleistung schneller passende Noncen und Hashwerte gefunden werden, wird der Schwellenwert der Difficulty periodisch angepasst. Nach jedem 2016. Block wird die Difficulty in Bitcoin entweder rauf oder runtergesetzt, um im Mittel bei einer Blockzeit von 10 Minuten zu bleiben. Wird durch zusätzliche Rechenleistung die durchschnittliche Blockzeit unter 10 Minuten

gebracht, steigt die Schwierigkeit an. Liegt durch den Wegfall von Rechenzeit die Blockzeit darüber, wird die Schwierigkeit nach unten korrigiert.

Nun kann es jedoch passieren, dass zwei Knoten zur selben Zeit eine Nonce finden, die das Ziel erfüllt. Grundsätzlich gilt, dass ein Knoten denjenigen gültigen Block lokal aufnimmt, den er zuerst erhält. Dies würde allerdings im ungünstigsten Fall dazu führen, dass das Netzwerk sich in zwei Hälften partitioniert, die disjunkt voneinander weiterlaufen. Es entsteht ein *Fork*. Um dies auf lange Sicht zu verhindern, gibt es die sogenannte *Longest-Chain-Rule*.

Regel der längsten Kette

Die Entstehung eines Forks ist in PoW Algorithmen nicht vermeidbar. Allerdings ist es zunächst nicht weiter problematisch, da PoW keine starken Konsistenzkriterien hat, sondern „eventually consistent“ ist. Um jedoch zu vermeiden, dass Netzwerkpartitionen immer weiter voneinander in Forks divergieren, gibt es die Regel der längsten Kette (Longest-Chain-Rule).

Ganz im Sinne des Vertrauens in die Rechenleistung, besagt auch die Longest Chain Rule, dass der Fork mit der längsten Kette der vertrauensvollste ist, da am meisten Rechenleistung in ihm steckt. Netzwerkteilnehmer sind also immer angehalten, sich auf einen Fork festzulegen oder auf den zu wechseln, der die meisten Blöcke beinhaltet [Nakamoto 2008]. In Abbildung 3.7 wird dies beispielhaft dargestellt. Auf den Block `7` folgen die Blöcke `8A` und `8B`, die zeitgleich von zwei unterschiedlichen Minern erstellt wurden, sodass sich Teile des Netzwerkes auf einen der beiden festlegen. Folglich erstellen Teilnehmer separat voneinander die Blöcke `9A` und `9B`, die auf die jeweiligen vorherigen Blöcke folgen. Erst mit dem Block `10` schafft es ein Netzwerkteil eine längere Kette zu erzeugen. Da im anderen Teil des Netzes kein Nachfolger auf `9A` gefunden wurde, wechseln alle Teilnehmer auf die neue längere Kette. Die Blöcke `8A` und `9A` des Forks sind nun *Orphans* (Waisen) und werden verworfen.

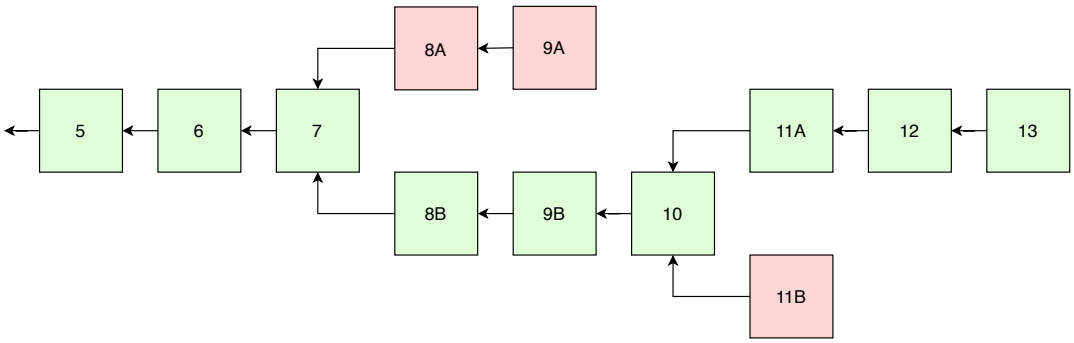

Abbildung 3.7.: Blockchain mit zwei Forks

Die Longest Chain Rule und die Funktionsweise von PoW Algorithmen allgemein ist der Hauptgrund für die schwache Konsistenz des Netzwerkes. Hingegen ist es dadurch hochverfügbar und partitionstolerant. Jedoch ist dies auch einer der Gründe, warum sich PoW Algorithmen nicht für bestimmte Vorgänge, wie beispielsweise *Instant-Payments* eignen. Das Verwaisen von Blöcken in unbenutzten Forks hat mehrere negative Konsequenzen für die Transaktionen darin.

Zuerst verliert der Miner, der die verwaisten Blöcke erstellt hat, die Belohnung in Form des Blockrewards und der enthaltenen Transaktionsgebühren. Dies stellt für die meisten Teilnehmer des Netzes erst einmal kein sonderliches Problem dar. Hingegen ist es für Netzwerkteilnehmer deutlich schlechter, wenn ihre Transaktionen in einem verwaisten Fork auftauchen, aber nichtmehr in der aktuellen Kette enthalten sind. Wenn also eine Transaktion in der Blockchain als Folge eines Offchain-Prozesses getätigt wurde, muss dieser Prozess zwangsweise auch rückabgewickelt werden, was nicht zwingend in jedem Fall möglich ist.

Wenn beispielsweise für eine erbrachte Dienstleistung Bitcoins zwischen Käufer und Verkäufer transferiert wurden und diese in einem verwaisten Fork landen, müsste diese Transaktion erneut ins Netzwerk gestellt und in einen Block inkludiert werden. In der Praxis wird, beispielsweise bei Bitcoin, dazu geraten, dass sechs oder mehr Blockzyklen gewartet wird, bevor eine Transaktion als final angesehen werden kann. Dies stellt sicher, dass die Trans-

aktion in der Hauptkette ist und auch bleibt. Ausgehend von sechs Blockzyklen in Bitcoin resultiert eine verzögerte Anerkennung der Transaktion um eine Stunde, was dieses System äußerst ungeeignet für Instant-Payments macht.

Die Auswirkungen eines Forks lassen sich auch gezielt für ein Double-Spend ausnutzen. In einer 51%-Attacke oder Long-Range-Attacke können so mutwillig Forks erstellt werden, um ein Double-Spend zu ermöglichen. Beide Attacken funktionieren ähnlich und nutzen die Longest-Chain-Rule und die Notwendigkeit einer einfachen 50% Mehrheit in PoW Algorithmen aus. Die 51%-Attacke beschreibt einen Umstand, in dem ein bösartiger Teilnehmer eine Rechenleistung von mehr als 50% der Gesamtkapazität des Netzwerkes innehat. Dies ermöglicht es ihm, im Schnitt schneller Blöcke zu erstellen als andere. Dadurch kann er gezielt Forks begünstigen, um Transaktionen in Blöcke zu platzieren, die seine Interessen favorisieren.

Die Long-Range-Attacke geht noch einen Schritt weiter. Ihr zugrunde liegt die Anforderung, dass ein Teilnehmer eine signifikante Menge an Rechenleistung besitzt. Zur Durchführung eines Double-Spends mittels der Long-Range-Attacke beginnt der Angreifer an einem Block vor der zu invalidierenden Transaktion. Nun wird parallel zur Hauptkette im Geheimen absichtlich ein Fork erstellt, indem er neue Blöcke bildet, inklusive der Berechnung gültiger Nonce und Hashwerte. Sobald der erstellte Fork länger ist als die aktuelle Kette, wird dieser Fork ins Netzwerk publiziert, sodass alle Knoten diese Kette als neue längste Kette übernehmen. Mit genügend Rechenkapazität kann ein Angreifer an einem beliebigen Zeitpunkt bzw. einer beliebigen Blockhöhe anfangen neue Forks zu erstellen, um die Hauptkette abzulösen – theoretisch auch direkt beim Genesis-Block.

Nachteile von Proof-of-Work

Obwohl Proof-of-Work die meisten Vorteile bietet, wenn es um vollständige Dezentralisierung und komplett vertrauenslose Netze geht, leidet es gerade auch deswegen unter diesen Designentscheidungen.

Der Vertrauensbeweis für gültige Blöcke in Form des Minings, konkret das Brute-Forcing der Nonce, benötigt viel Rechenleistung. Je mehr Rechenleistung ein Miner aufwendet, desto höher ist seine Wahrscheinlichkeit den Blockreward zu erhalten. Nun gibt es für die Miner zwei Möglichkeiten ihre Gewinne zu maximieren, entweder sie kaufen mehr und spezialisiertere Hardware oder sie schließen sich Mining-Pools an [Hileman und Rauchs 2017].

Die Wahl des Hashingalgorithmus, beispielsweise SHA-256 in Bitcoin, führte Anfang 2013 zur Entwicklung von spezieller ASIC Miner Hardware (Application Specific Integrated Circut, dt. anwendungsspezifische integrierte Schaltungen). Dies ist spezielle Hardware, die nur noch in der Lage ist, Hashwerte zu berechnen, dabei aber 50-mal schneller als Heim-PCs sind, die bis dato eingesetzt wurden. Die weitreichende Kritik an diesen Geräten und an PoW allgemein ist die Verschwendung von Ressourcen. Wie für Heim PCs, Server und Mobiltelefone werden für die Herstellung der ASICs die Metalle der „Seltenen Erden“ (z.B. Cerium, Neodym, Lanthan u.a.) benötigt, deren Gewinnung mit großen Umweltproblemen verbunden ist.

Zusätzlich zu den benötigten Mineralen, verbrauchen die Rechner natürlich Strom. Der jährliche Stromverbrauch von Bitcoin allein wird auf ca 70 TWh geschätzt, was dem Jahresverbrauch von Österreich entspricht. Um den globalen Energieverbauch von Proof-of-Work festzustellen, müssten noch etwa 200 weitere Kryptowährungen mit einberechnet werden. Im Vergleich mit *VISA* Transaktionen, diese setzen allerdings auf zentrales Vertrauen, schneidet Bitcoin deutlich schlechter ab. Anhand des geschätzten Jahresenergieverbrauches von Bitcoins und den getätigten Transaktionen lässt sich berechnen,

dass die Bestätigung einer Bitcoin Transaktion ca. 740 kWh verbraucht, die einer VISA-Transaktion hingegen nur 0,00151 kWH [Digiconomist 2020].

Der Zusammenschluss zu Mining-Pools ermöglicht es Minern mit weniger eigener Rechenleistung sich zu einer Gruppe zusammenzuschließen, die dadurch wieder eine höhere Wahrscheinlichkeit hat eine Nonce zu Brute-Forcen. Die Gewinne werden zwar nach dem Anteil der Rechenleistung aufgeteilt, jedoch ist die Frequenz von Gewinnausschüttungen deutlich höher, als wenn jeder der Miner allein agiert. Im Laufe der Zeit schlossen sich nun vermehrt einzelne Miner solchen Pools an, was dazu führte, dass 2016 vier große Mining Pools bereits über 80% der Rechenleistung zu Bitcoin beisteuerten [Hileman und Rauchs 2017].

Hinzukommt noch die räumliche Aufteilung der Mining Pools. Das Betreiben von profitablen Mining Pools erfordert viel Hardware, die wiederum viel Strom benötigt, was wiederum dazu führt, dass viele der Pools in Ländern mit günstigem Strom angesiedelt sind. Konkret heißt das für Bitcoin, dass 58% der Rechenleistung in China getätigt wird. Generell führt dies zu einer ungewollten Zentralisierung des Netzwerkes, da nun vier Mining Pools darüber entscheiden, welche Transaktionen in welchem Block aufgenommen werden, die zusätzlich zu großen Teilen von einer autokratischen bzw. diktatorischen Regierung beeinflusst werden könnten. Außerdem entscheiden diese großen Pools maßgeblich über die Weiterentwicklung des Protokolls. Wenn nämlich Protokolländerungen, z.B. die Erhöhung der Blockgröße, zu ihrem Nachteil gemacht würden, könnten sie sich entschließen, weiter nach dem alten Protokoll zu arbeiten, was zu einem (Hard-) Fork führen würde. Nun müssten sich die anderen Netzwerkteilnehmer entscheiden, ob sie die neue Protokollversion präferieren, oder auf einem Fork verbleiben, der durch deutlich mehr Rechenleistung abgesichert ist [Hileman und Rauchs 2017].

3.3.2. Proof-of-Stake

Der häufigst genannte Kritikpunkt an PoW ist das massive und nutzlose Verschwenden von Energie und Rechenleistung für die Hashwertfindung. Um dem entgegenzuwirken, wurden Projekte vorgeschlagen, bei denen die Hashwertberechnung zu einem nützlichen Ziel (*Useful Work*) beitragen, wie die Findung von Primzahlen (*Primecoin* [King 2013]) oder die Lösung von Matrix-basierten Rechnungen (*Proof of eXercise* [Shoker 2017]). Problematisch ist hierbei, eine Aufgabe zu finden, die schwer zu lösen und nach Möglichkeit in der Schwierigkeit anpassbar, aber dennoch leicht zu verifizieren ist [King 2013].

Ein gegenläufiger Ansatz ist hingegen das Reduzieren der Rechenleistung und des Energiebedarfs durch gänzliches Verzichten auf „Arbeit“ und die resultierende Verringerung der erforderlichen Hardware dafür. In *Proof-of-Stake* (PoS) wird, wie der Name beschreibt, auf das Einsetzen von Münzen oder Stakes (dt. Anteile) gesetzt, wodurch ein *Mining* teilweise oder ganz entfällt [King und Nadal 2012].

Prinzipiell gilt, dass in PoS ein *Minter* (Münzpräger) Anteile seines bisherigen Besitzes einsetzen (staken) muss, um einen Block zu erstellen und um zusätzlich das Vertrauen in seinen neuen Block zu erzeugen. Je nach dem gewählten PoS muss die eingesetzte Währung bestimmte Kriterien erfüllen, um einsetzbar oder „vertrauensvoller“ zu sein. In *Peercoin* wird dazu das Coinage (Münzalter) verwendet. Eine Münze ist hier umso vertrauensvoller, desto länger sie unbenutzt in einem Wallet liegt. So wird sichergestellt, dass auch Minter mit wenig Besitz am PoS teilnehmen können, indem sie einfach etwas länger warten. Wie auch beim Mining wird in PoS der Minter in Form von neuer Währung für seinen Einsatz belohnt [King und Nadal 2012].

Damit Minter überhaupt eine Initialwährung zum Einsatz bekommen, gibt es mehrere Möglichkeiten, wie man diese erzeugen kann. In privaten Netzwerken können Initialwährungen vor dem Start zentral definiert werden. Für öffentliche Netzwerke haben sich einerseits Initial Coin Offerings dafür

etabliert, oder es wird andererseits im laufenden Betrieb von einem PoW-Konsens auf ein PoS-Konsens gewechselt. Bei einem Wechsel werden aus den Minern nun Minter, während ihre Währung bestehen bleibt.

Angriffe in PoS Algorithmen

In frühen Varianten von PoS Algorithmen entstanden zwei Probleme durch die gewählte Form des „Einsetzen“ von Stake. Das reine Besitzen von Währung war bereits ausreichend, um sich fürs Minting zu qualifizieren. Ein Minter musste seine Währung nicht hinterlegen oder einzahlen, somit konnte auch keine Bestrafung eingeführt werden, wenn sich dieser nicht protokollkonform verhielt. Wie auch in PoW, ist es möglich, dass Forks entstehen, wenn sich zwei oder mehr Minter gleichzeitig qualifizieren, einen Block zu erstellen. Während sich in PoW nun die Miner für einen Ast des Forks entscheiden und darauf aufbauend weiter Hashes berechnen, müssen Minter dies nicht tun. Wenn ein Miner sich entscheidet, auf beiden Ästen des Forks Hashes zu berechnen, würde das seine Rechenleistung pro Ast halbieren (ausgehend von einer 50:50 Aufteilung). Da ein Minter keine signifikante Rechenleistung benötigt, kann er seinen Block an beide Äste des Forks anhängen und so sicherstellen, dass sein Block auf jeden Fall in der längsten Kette der Blockchain auftaucht. Ultimativ würde dies dazu führen, dass fortan beide Forks für immer bestehen würden, da alle Minter sich diesem Vorgehen kontinuierlich anschließen würden. Dieses Problem wird „Nothing-at-Stake“ (Nichts zu verlieren) genannt und ist heutzutage nur noch von theoretischer Bedeutung, da in keinem aktuellen PoS Algorithmus mehr das bloße Besitzen von Währung ausreicht [Rose u. a. 2019].

Das zweite Problem ist die Long-Range Attacke, die analog zu der Long-Range Attacke in PoW Algorithmen funktioniert. Während in PoW ein Angreifer deutlich mehr Rechenleistung als der Rest des Netzwerkes benötigt, um eine Kette zu produzieren, die länger ist als die aktuelle, ist dies in PoS mit vernachlässigbarem Rechenaufwand möglich. Hierdurch geht von Mintern mit hohen Anteilen von Währung eine deutlich größere Gefahr aus, da

diese tendenziell zu jederzeit beschließen könnten, mit ihrem Stake einen Fork zu erstellen und die aktuelle Kette durch eine längere zu überholen [Rose u. a. 2019].

Delegated Proof-of-Stake

Weitere Anpassungen an PoS Algorithmen wurden vorgenommen, um potentiellen Skalierungsproblemen vorzubeugen, die entstehen, wenn jeder der teilnehmenden Knoten zu jederzeit auch Minter sein kann und häufig zwischen Forks gewechselt wird. Zusätzlich verhindern PoS Algorithmen ohne spezielle Vorkehrungen mit zunehmender Zeit, dass auch Teilnehmer mit kleinem Stake aktiv an der Blockbildung teilnehmen, obgleich die Reichen mit großen Stake-Anteilen ständig begünstigt sind und bleiben [Larimer 2017, 2018].

Im *Delegated Proof of Stake* (DPoS) wird die Menge an Kandidaten durch einen deterministischen oder pseudozufälligen Prozess limitiert. Obwohl die Menge limitiert ist, sind die übrigen Teilnehmer mit Stake nicht komplett ausgeschlossen. In *Bitshares* [Schuh und Larimer 2017] und *Steemit* [Steemit Inc. 2017] wird dazu eine Menge an Teilnehmern vorgeschlagen, die als *Witness* (dt. Zeuge) fungieren. In einer weiteren Runde wird über diese abgestimmt, sodass die N Witnesses mit den meisten Stimmen für die nächste Periode (hier ein Tag) zuständig ist Blöcke zu produzieren. Blöcke werden reihum alle 2 Sekunden produziert. Nachdem alle Witnesses einen Block produziert haben, wird die Reihenfolge geändert und sie produzieren wieder reihum Blöcke.

Im Ouroboros Protokoll werden je Periode (ca. 5 Tage) neue Stakeholder durch die Stakeholder der vorherigen Periode gewählt und in Stake Pools zusammengefasst. Innerhalb der Perioden gibt es mehrere Slots zu 20 Sekunden, in denen je ein Teilnehmer einen Block erstellt und seinen Nachfolger bestimmt, der wiederum den nächsten Block erstellt [Kiayias u. a. 2017].

Auch das Stellar Protokoll läuft ähnlich zu den beiden vorgestellten ab, indem Submengen der Knoten ausgewählt werden und Gruppen bilden. Jeder Teilnehmer beschliesst für sich selbst, welchen anderen Teilnehmern er vertraut und so ergeben sich Gruppierungen von gegenseitigem Vertrauen. Diese formen ein „Quorum“, welches ausreichend groß ist, um eine Übereinstimmung zu finden. Um Konsens zu erreichen, versucht eine Menge der Teilnehmer (Quorum Slice) einen Knoten zu überzeugen. Zuerst werden Werte in einer Runde vorgeschlagen und anschließend wird in der zweiten Runde darüber abgestimmt diese Werte anzunehmen oder abzulehnen [Mazieres 2015].

3.3.3. Byzantine-Fault-Tolerant PoS

Byzantinisch fehlertolerante Proof-of-Stake (BFT-PoS) Algorithmen versuchen das Konsensproblem zu lösen, indem sie das byzantinische Generäle Problem mittels BFT-Replikation lösen [Vukolić 2015]. Viele der BFT-PoS Algorithmen basieren dabei mehr oder weniger stark auf dem Algorithmus *PBFT* ([Castro und Liskov 1999]) oder anderen *2-* bzw. *3-Phasen-Commit* ([Lampson und Sturgis 1979]) Protokollen. Der grundlegende Unterschied zu den bisher vorgestellten PoW und PoS Algorithmen besteht darin, dass die Teilnehmer hier immer gemeinsam über einen Block in mehreren Runden abstimmen und anschließend signieren.

In BFT-PoS werden alle Teilnehmer nach zwei Eigenschaften unterschieden, beziehungsweise in zwei Gruppen eingeteilt. Die *Validatoren* sind die Gruppe der Knoten, die aktiv am Konsens beteiligt sind, also neue Blöcke erstellen und im Netzwerk propagieren. *Nicht-Validatoren* hingegen nehmen nur passiv am Konsens teil. Während jeder der Knoten Transaktionen und Blöcke validieren muss (entgegen der Bezeichnung der Knoten), sind nur die Validatoren für das Signieren des Blockes zuständig. Validatoren bürgen mit ihrer Signatur für die Gültigkeit der Transaktionen und des Blockes im Ganzen. Sie

haben somit auch eine übergeordnetere Rolle im Netz und stellen eine Menge von Teilnehmern mit höherem Vertrauen dar [Kwon 2014].

Die Prozesse, die in BFT-PoS Algorithmen ablaufen, sind in drei Phasen aufgeteilt. In der ersten Phase schlägt einer der Validatoren (Proposer) einen Block zur Abstimmung vor. In den darauffolgenden Phasen wird, analog zu PBFT mittels Pre-Prepare und Prepare Nachrichten, jeweils darüber abgestimmt, diesen Block anzunehmen oder abzulehnen. Hierbei verhalten sich die Validatoren asynchron und wechseln eigenständig in die nächste Phase, sobald sie eine $+\frac{2}{3}$ Mehrheit erhalten haben. Abbildung 3.8 zeigt beispielhaft den Ablauf eines BFT-PoS Konsens mit 4 Validatoren anhand des *Tendermint* Algorithmus, bei dem jedoch die Phasen *Pre-Vote* und *Pre-Commit* heißen, sich aber nicht in deren Funktion von den Phasen aus PBFT unterscheiden [Kwon 2014].

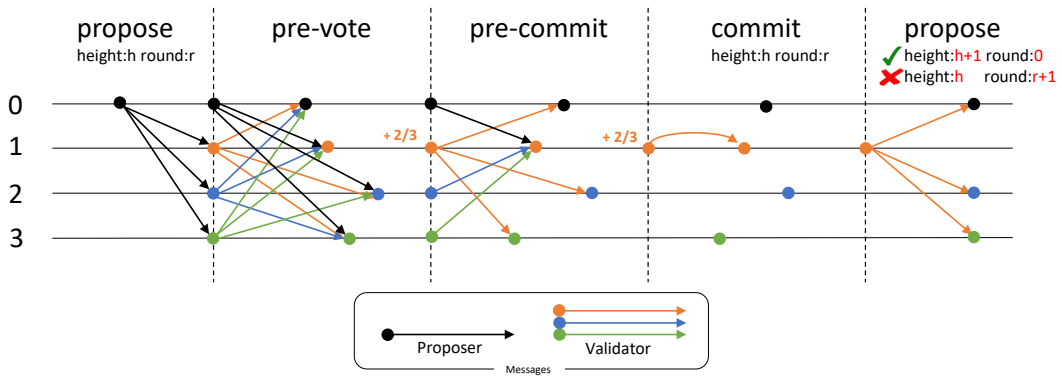

Abbildung 3.8.: Tendermint BFT-basierter Konsensalgorithmus

Sollte in einer Runde keine $+\frac{2}{3}$ Mehrheit zustande kommen, entweder durch Timeouts oder Ablehnung, wird die aktuelle Runde abgebrochen, der Rundenzähler erhöht und der Prozess von vorne begonnen. Sobald ein Validator lokal einen Block speichert (commit), wartet er auf den nächsten Vorschlag für die Höhe $H + 1$ und Runde 0. Während in PoW Algorithmen Knoten zu jederzeit entscheiden können, ob sie für die nächste Höhe minen wollen oder nicht, müssen in BFT-PoS mindestens $+\frac{2}{3}$ der Validatorknoten am Konsens teilnehmen, damit ein Block zur nächsten Höhe entsteht. Sobald weniger als

$\frac{2}{3}$ verfügbar, sind kann keine Mehrheit an Stimmen erreicht werden, selbst wenn die Anzahl exakt für eine $+\frac{2}{3}$ Mehrheit ausreicht, müssen auch alle Validatoren positiv für den Block abstimmen. Andererseits kann durch die beschränkte Menge der Validatoren sofort festgestellt werden ob eine ausreichende Mehrheit erreicht wurde, was zu sofortiger Block-Finalität führt [Buchman 2016].

Während in reinem PoS ein Stake eingesetzt und dies in irgendeiner Weise mit im Block verankert wird, werden in BFT-PoS Signaturen verwendet. Zusätzlich werden in einer Liste alle Validatoren und ihre öffentlichen Schlüssel vermerkt. Um diese Liste gegen Manipulation zu schützen, wird diese im Genesis-Block verankert. Hierdurch wird allen Teilnehmern, egal ob Validator oder Nicht-Validator, die Menge der Validatoren bekannt gemacht. So ist es jedem Validator möglich, die Nachrichten des Konsensprotokolls auf gültige Signaturen zu prüfen und präzise zu entscheiden, wann eine $+\frac{2}{3}$ Mehrheit erreicht wird, um in die nächste Phase zu wechseln. Weiterhin ermöglicht das Signieren von Nachrichten, dass der Nachrichtendurchsatz in einem P2P Netzwerk verkleinert werden kann. Indem Nachrichten mehrfach signiert und per *Gossip* weiter verbreitet werden, muss nicht jeder Knoten jedem Knoten einzeln Nachrichten senden. Zusätzlich führen Multisignaturen auch dazu, dass beispielsweise nur eine einzige Nachricht ausreicht, um einen neu hinzugekommenen Teilnehmer von der Mehrheit zu überzeugen [Buchman 2016].

Anhand eines deterministischen Prozesses wird festgelegt, welcher Validator zu welcher Zeit zum Proposer wird, zum Beispiel aus einer Kombination von Blockhöhe und Rundenzahl in einem Round-Robin Verfahren. Dieser Umstand sorgt dafür, dass es in BFT-PoS Algorithmen keine Notwendigkeit für die Longest-Chain Regel gibt, denn zu jeder Höhe und Runde kann es nur einen Proposer geben, solange sich alle Knoten an das Protokoll halten. Gleichzeitig bedeutet dies auch, dass BFT-PoS Blockchains nicht „eventual consistent“ sind, denn es muss nach Abstimmung über einen Block keine bestimme Karenzzeit mehr gewartet werden, bis die darin enthaltenen Transaktionen als final angesehen werden können. BFT-PoS Algorithmen sor-

gen also für eine starke Konsistenz (immediate consistency) [De Angelis u. a. 2018].

Während PoW Algorithmen im CAP-Theorem die Eigenschaften $A + P$ erfüllen, also Verfügbarkeit (Availability) und Partitionstoleranz, gehören BFT-PoS in die Kategorie von $C + P$ Systemen. Sie präferieren also Konsistenz (Consistency) und Partitionstoleranz vor Verfügbarkeit. Abbildung 3.9 verdeutlicht diese Einteilung nochmal am CAP-Theorem. Grundsätzlich lässt sich für Konsensalgorithmen von Blockchains festhalten dass, sobald durch einen Algorithmus Forks entstehen, keine starke Konsistenz herrschen kann [De Angelis u. a. 2018].

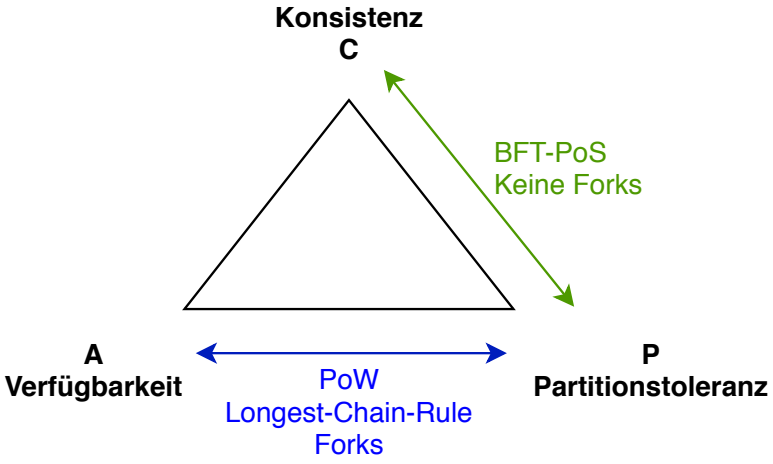

Abbildung 3.9.: Einordnung in das CAP-Theorem

Durch die Eigenschaften von BFT-PoS Algorithmen eignen sich diese besonders gut für Geschäftsprozesse, die eine starke Konsistenz erfordern und das immer unmittelbar. Während PoW Blockchains, wie Bitcoin, von einer hohen Dezentralität profitieren, leiden sie unter den Auswirkungen der schwachen Konsistenz (eventual consistency). Long-Range, Nothing-At-Stake oder 51%-Attacken sind in BFT-PoS nicht möglich, da es zu jeder Blockhöhe nur einen Blockkandidaten gibt und es nicht möglich ist, ohne weiteres neue Rechenleistung (in der Form von zusätzlichen Validatoren) hinzuzufügen.

3.3.4. Proof-of-X

Um sich vom klassischen Proof-of-Work abzuheben, haben viele Startups und Projekte eigene Konsensverfahren entwickelt, die in verschiedenen Formen nachweisbasiert Ressourcen benötigten, sei es etwa in Form von Berechnungen, Speicher, Wartezeit oder sonstigen Formen. Einige dieser vorgeschlagenen Algorithmen sind kaum von praktischer Relevanz und haben nie wirklich breiten Einsatz gefunden. Nachfolgend werden einige ausgewählte Konsensverfahren kurz vorgestellt:

Proof of Burn basiert auf der absichtlichen Invalidierung von Coins. Hierbei werden Coins an eine verifizierbar ungültige Adresse transferiert, von der aus sie nicht mehr ausgelöst werden können [Karantias u. a. 2020].

Proof of Elapsed Time verwendet ein Lotteriesystem, was beteiligten Miner zufällige Wartezeiten zuteilt, nach deren Ablauf sie für die Blockbildung zuständig sind. Zur Ausführung wird Hardware von Intel mit *Trusted Execution Environments* benötigt [Chen u. a. 2017].

Proof of Excercise setzt wie PoW auf das Berechnen von Kryptorätseln mit der Ausnahme, dass hier die Rätsel auf der Berechnung von „sinnvollen“ Werten beruhen, wie beispielsweise Matrixmultiplikation [Shoker 2017].

Proof of Importance verwendet ein Vertrauensmodell auf Basis eines Transaktionsgraphen. Basierend auf der Graphentheorie, wird jedem Account ein Wichtigkeitswert zugeordnet, der wiederum die Blockbildung beeinflusst [NEM Foundation 2018].

3.4. Distributed Ledger

Der Begriff Distributed Ledger Technology (DLT) beschreibt sinngemäß eine Technologie für das dezentrale Verwalten von Kontobüchern. Während

klassische Blockchains die feste Struktur von einfach verketteten Blöcken haben, ist der DLT Begriff weiter gefasst. Um eine Technologie als DLT zu kategorisieren, erfordert es nur das Verwenden eines dezentralen Konsens zur Abstimmung über den Zustand und das Replizieren der Daten in einer verteilten Datenbank zwischen allen Teilnehmern. Somit sind klassische Blockchains, die Daten in Blöcken verketten, DLTs, aber nicht alle DLTs sind Blockchains.

Nachfolgend werden zwei DLTs vorgestellt, die das Blockchain-Konzept um weitere Mechanismen ergänzen oder bekannte austauschen, um deren Eignung für bestimmte Anwendungsfälle zu erhöhen.

3.4.1. Beispiel: IOTA

IOTA ist aus den speziellen Anforderung von IoT Geräten entstanden [Kusmierz 2017; Popov 2016]. Diese haben deutlich weniger Speicher und Rechenleistung als Desktop-PCs oder Server. Das Verwenden einer klassischen Blockchain für IoT wird erschwert durch das unvermeidliche ständige Wachsen der Block-Kette und das Aufbringen der notwendigen Rechenleistung für einen Proof-of-Work Konsensalgorithmus. Im ersten Quartal 2020 lag die Größe der Bitcoin-Blockchain bei 260 GB [Blockchain Luxembourg S.A. 2020], was für simple IoT Geräte astronomisch hoch ist. Ein weiterer Nachteil sind die Transaktionskosten, die durch Trinkgelder an die Miner entstehen. Angenommen, IoT-Devices tauschen selbstständig Transaktionen mit geringem Wert aus, sog. Mikrotransaktionen, dann werden diese unökonomisch, sobald das Trinkgeld den Wert der Transaktion übersteigt. Trinkgelder wiederum lassen sich nicht leicht aus Cryptocurrencies wie Bitcoin entfernen, da sie als Anreiz für die Miner fungieren, den Konsens im Netzwerk aufrechtzuerhalten.

In IOTA werden deshalb die folgenden Konzepte verändert, um Skalierbarkeit, Leichtgewichtigkeit und Mining-Bedarf zu verbessern. Anstelle einer einzelnen Kette von Blöcken wird in IOTA ein gerichteter azyklischer Graph

(genannt Tangle) von Transaktionen gebildet, wie in Abbildung 3.10 dargestellt. Um Arbeitslast im Netzwerk auf alle Teilnehmer zu verteilen, müssen Knoten, die eine Transaktion anhängen wollen, ein leichtgewichtiges Proof-of-Work (vgl. [Back 2002]) lösen. Einerseits vermeidet dies Spam und andererseits validieren Knoten so zusätzlich zwei oder mehr vorhergehende Transaktionen.

Die Genesis-Transaktion und die direkt darauf folgenden Transaktionen sind davon nicht betroffen (vgl. Abbildung 3.10). Der Genesis hat nur sich selbst als Vorgänger, und Transaktionen der Höhe Eins nur den Genesis als Vorgänger[3]. Durch diese Einschränkung hat jede Transaktion mindestens zwei validierte Vorgänger-Transaktionen. Daraus ergibt sich eine Kette von direkt oder indirekten validierten Transaktionen. Je größer die Menge aus direkten und je länger die Kette aus indirekten Bestätigungen, desto höher ist das Vertrauen in die Gültigkeit einer Transaktion.

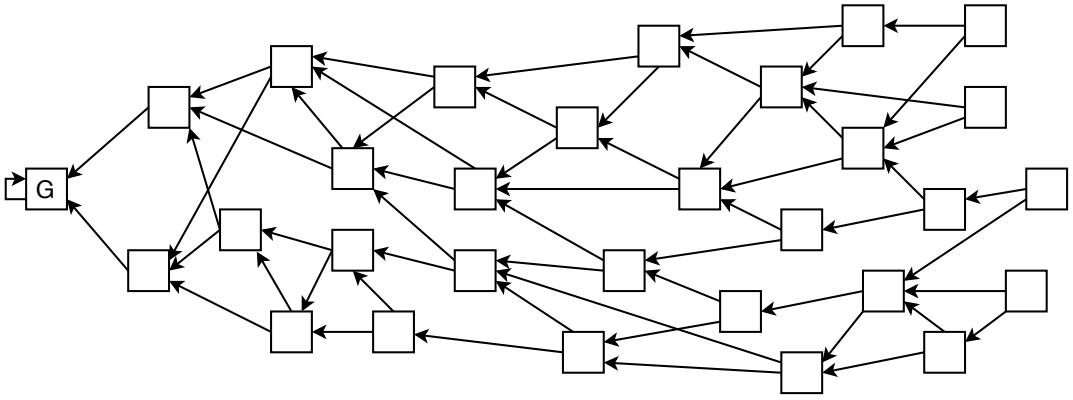

Abbildung 3.10.: IOTA „Tangle“ - Directed Acyclic Graph

Die Bestätigung von zwei vorhergehenden Transaktionen eliminiert den Bedarf an globalen Minern, da kein Proof-of-Work mehr für den Konsens verwendet wird und jeder Teilnehmer im Netzwerk gleichberechtigt ist. Transaktionen können zusätzlich deutlich schneller im Netzwerk als gültig anerkannt werden, da sie nicht an festgelegte Blockzeiten gebunden sind. Das

[3]Streng genommen ist dies nun kein gerichteter azyklischer Graph mehr, da der Genesis sich zyklisch selbst referenziert.

Netzwerk skaliert durch den Wegfall der Blockzeit in diesem Fall deutlich besser, da auch eine plötzliche Flut an Transaktionen unabhängig voneinander behandelt werden kann.

Letztlich wird die Leichtgewichtigkeit für IoT-Geräte dadurch hergestellt, dass diese nicht den gesamten Graphen speichern müssen, sondern nur Subgraphen mit relevanten Daten. Ein zusätzlicher Snapshot-Mechanismus sorgt für ein periodisches Verkleinern der Daten. Hierbei werden nur Kontostände persistiert und alle Transaktionen und somit die ganze Historie gelöscht, um den Tangle komplett von vorne zu beginnen.

Die Wahl eines Snapshot-Mechanismus und die nicht variable Proof-of-Work Challenge sind die Hauptkritikpunkte an IOTA [Johnson 2017; Kauflin 2018; Rodarmor 2018]. Der Snapshot-Mechanismus wird global durch einen zentralisierten Koordinator gesteuert. Während dieser einen Snapshot durchführt und erneut verteilt, wird das ganze Netzwerk blockiert. Zusätzlich stellt dieser einen zentralen Point-of-Failure dar. Weiterhin wird in den meisten PoW-Verfahren eine variable Schwierigkeit verwendet. Sollte das Netzwerk schneller Blöcke produzieren, steigt die Schwierigkeit an, um im Mittel eine feste Blockzeit zu erhalten. Da die PoW-Schwierigkeit in IOTA gleichbleibend niedrig ist, kann ein Angreifer mit vergleichsweise wenig Aufwand das Netzwerk mit Transaktionen fluten und zu seinen Gunsten beeinflussen.

3.4.2. Beispiel: Hashgraph

Zur etwa selben Zeit wie IOTA entwickelt wurde, entstand 2016 eine weitere Variante eines konsensbasierten Ledgers, der nicht auf einer Kette von Blöcken, sondern auf einem azyklischer Graphen beruht. Im Hashgraph werden durch einen asynchronen BFT-Konsensalgorithmus *Events* zeitlich zueinander in Verbindung gebracht [Baird 2016]. Events in Hashgraph sind prinzipiell Blöcken in Blockchains gleichzusetzen, da sie zur Bündelung von Transaktionen und kryptografischen Absicherung des Inhaltes verwendet werden. Werden die Events und ihre Vorgänger, ausgehend von ihren versendenden

Knoten, auf einer Zeitline dargestellt, ergibt sich der namensgebende „Hashgraph“, wie in Abbildung 3.11 dargestellt.

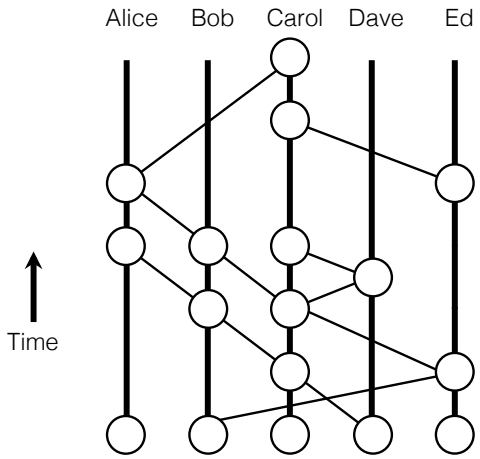

Abbildung 3.11.: Beispiel eines Graphens mit Events in Hashgraph

Während bei IOTA eine Minimalversion eines PoW-Konsensalgorithmus verwendet wird, verwendet Hashgraph einen BFT-basierten Konsensalgorithmus. Um auf einen Proposer/Leader zu verzichten, wie es in BFT sonst üblich ist, wird ein virtuelles Voting vorgenommen. Zusätzlich zum normalen Gossip-Protokoll, wo jeder Teilnehmer einkommende Nachrichten an alle seine Nachbarn weiterschickt, wird hier ein weiterer Schritt unternommen. Eine zusätzliche Nachricht sagt, wer welche Nachricht an wen gesendet hat. Dieses Verfahren wird in Hashgraph „gossip about gossip“ genannt. Das virtuelle Voting wird nun unterstützt, indem ein Teilnehmer A errechnen kann, für welche Events ein anderer Teilnehmer B abstimmen würde, ohne das eine direkte Vote-Nachricht von B zu A geschickt wird. Wie auch in anderen BFT Algorithmen wird hier rundenbasiert über die Reihenfolge der Events abgestimmt.

3.5. Netzwerkformen und Zugangsrechte

Blockchain-Systeme lassen sich, wie auch im Cloud-Computing-Bereich, nach ihrer Zugangsform, und den vorhandenen Rechten in diesen, in bestimmte Klassen gruppieren. Dabei kann zwischen *Public* (öffentlich) und *Private* (nicht öffentlich) Netzwerken zusammen mit *Permissioned* (zulassungsbedingt) und *Permissionless* (zulassungsfrei) unterschieden werden [Pilkington 2016]. Je nach Wahl der Kombination der beiden Eigenschaften ergeben sich zu einem bestimmten Anwendungsfall passende Nutzungsvorteile [Klebsch u. a. 2019; TeleTrusT 2017]. Häufig werden die Netzwerkform und die Berechtigung jedoch austauschbar verwendet. Es ergibt aber Sinn, diese differenziert zu betrachten, um auf die speziellen Ausprägungen der jeweiligen Kombinationen hinzuweisen. Aufgrund der besseren Differenzierung werden in dieser Arbeit die Begriffe *public* und *private* sowie *permissoned* und *permissionless* getrennt verwendet.

Als *Public* Netzwerke werden im Blockchain-Umfeld solche Netze bezeichnet, die keinerlei Zugangsrechte zum Lesen der Daten erfordern, also öffentlich einsehbar sind. Dies sorgt für erhöhte Transparenz, da jeder potentielle Teilnehmer aktiv auf die Daten zugreifen, diese nachvollziehen und nachverfolgen kann.

Um zu kennzeichnen, dass Teilnehmer auch schreibend auf ein Blockchain-Netzwerk einwirken können, wird der Begriff *permissionless* attributiert. In public-permissionless Systemen können also alle Teilnehmer sowohl lesen als auch schreiben. Dadurch, dass jeder Teilnehmer auch den Zustand der Blockchain verändern kann, gibt es weitere Schutzmechanismen, um etwa Spam zu vermeiden, wie etwa Proof-of-Work und Transaktionskosten. Zusätzlich wird mit public-permissionless auch häufig ausgedrückt, dass Teilnehmer aktiv am Konsens teilnehmen können, z.B. durch das Lösen der Hash-Challenge in PoW-Algorithmen.

Gegensätzlich zu *public-permissionless* stehen die *public-permissioned* Systeme. Wie der Name schon suggeriert, erfordern diese Zugangsberechtigungen zum

Schreiben. Allerdings muss auch hier differenziert werden zwischen der generellen Möglichkeit, mittels Transaktionen den Zustand zu verändern, oder aktiv am Konsens teilzuhaben durch das Vorschlagen oder Anhängen von neuen Blöcken. Eine Konsens-Beschränkung in permissioned Systemen wird meist in Proof-of-Stake oder Byzantine-Fault-Tolerant Konsensalgorithmem eingesetzt, was die Anzahl der Teilnehmer deutlich reduziert und dadurch die Blockzeit erhöht, jedoch Vertrauen in diese zentralisierten Teilnehmer erforderlich macht.

In *Private* Netzwerken wird hingegen der Zugang zum Netzwerk komplett beschränkt. Das heißt, dass ohne Einladung oder Berechtigung weder schreibend noch lesend auf das Netzwerk zugegriffen werden kann. Diese Form wird häufig auch als Konsortial- oder Federated-Blockchain bezeichnet, da sie häufig im inner- oder zwischenbetrieblichen zum Einsatz kommt. Wie auch bei Public Netzwerken kann man hier zwischen *private-permissionless* und *private-permissioned* unterscheiden, wobei ersterem keine praktische Relevanz beigemessen werden kann.

Private-permissioned Netzwerke werden zum Beispiel in Konsortien eingesetzt, um einer Menge aus mehreren Teilnehmern generellen Zugriff auf die Daten zu gewähren. Während jeder Teilnehmer Transaktionen lesen und einstellen darf, wird nur von einem Subteil dieser die Blockbildung übernommen. Als Konsensalgorithmus wird in den häufigsten Fällen kein PoW benutzt, da mittels anderer Algorithmen die Blockzeiten und die notwendigen Rechenkapazitäten wesentlich geringer sind. Zusätzlich wird für den Konsens keine spezielle Hardware benötigt, normale Server aus Rechenzentren sind hier ausreichend.

Für Kryptowährungen, wie Bitcoin, ist eine public-permissionless Blockchain die notwendige Wahl. Sie erlaubt es allen Teilnehmern unbenachteiligt am Netzwerk teilzunehmen, Transaktionen zu lesen und eigene Transaktionen einzustellen. Dies schafft die notwendige Transparenz in die modellierte Währung. Zusätzlich wird der Konsens potentiell von allen Teilnehmern gleicher-

maßen ausgeführt, was für die notwendige Dezentralität und den Wegfall von jeglichen Intermediären sorgt.

Für den betrieblichen Einsatz, unabhängig ob inner- oder zwischenbetrieblich, ist ein vollständiges offenes Model meist nicht geeignet. Während es in Supply-Chain Anwendungsfällen durchaus denkbar ist, ein public-permissioned System zu betreiben, damit Endnutzer die Verarbeitungsschritte nachvollziehen können, aber selbst keine hinzufügen, ist dies für andere Geschäftsprozesse, z.B. aufgrund von Geschäftsgeheimnissen, nicht wünschenswert ein öffentliches Netzwerk zur Verfügung zu stellen.

3.6. Anwendungsfälle für Blockchains

Die potentiellen Anwendungsfälle für Blockchains umfassen bei weitem nicht nur Kryptowährungen. Viele Prozesse können von der Dezentralität, Transparenz und Immutabilität profitieren, die der Einsatz einer Blockchain-Technologie mit sich bringt. Dennoch werden viele Prozesse, die heutzutage irgendwo ablaufen, durch „vertrauenswürdige“ Intermediäre oder zentrale Institutionen abgewickelt. Ein Einblick hinter die Kulissen ist hier aus vielen Gründen technisch gesehen, aber auch wirtschaftlich, nicht möglich oder erwünscht. Zusätzlich haben Intermediäre verständlicherweise wenig Interesse, ihr eigenes Geschäftsmodell durch eine dezentrale Lösung selbst zu zerstören.

3.6.1. Digitale Währung

Der Anwendungsfall einer Währung ist wohl der erste, der mit Blockchain in Verbindung gebracht wird. Vor allem dank der rapiden Popularisierung von Bitcoin, *Altcoins* und später auch Ethereum. Die Kombination aus dezentralem PoW-basiertem Konsensalgorithmus mit verketteten Logs ist eine perfekte Lösung für das Double Spending Problem, weshalb häufig auf zen-

trale Banken gesetzt wird, um zu gewährleisten, dass alle Transaktionen in einer bestimmten distinkten Reihenfolge ablaufen.

Das Vertrauen, das Nutzer Bitcoin oder anderen Kryptowährungen entgegenbringen, basiert auf demselben Vertrauen, das Nutzer in Hash-Funktionen oder in andere kryptografische Verfahren legen: Sie sind mit der heutigen verfügbaren Rechenkapazität nicht in annehmbarer Zeit berechenbar bzw. brechbar. Solange keine effizienteren Methoden als reines Brute-Force für die Hash-Berechnung gefunden werden, kann man davon ausgehen, dass der PoW Konsens weiterhin die Sicherheit gewährleistet, die von einer Währung erwartet wird.

3.6.2. Supply Chain Management

Unter Supply Chain Management wird das Verwalten von Aufwärts- und Abwärtsbeziehungen zwischen Lieferanten und Konsumenten verstanden, um einen höheren Kundenwert bei niedrigeren Kosten in der Lieferkette im Ganzen zu erreichen [Christopher 2005].

Die zunehmende Globalisierung erschwert jedoch das Verwalten und die Kontrolle über die Prozesse einer Lieferkette, mit ständig wachsender Komplexität und diversen regionalen Gesetzen. Einige Hersteller haben nicht einmal Kenntnis von ihren direkten Lieferanten, geschweige denn von deren Lieferanten [Bonanni 2011]. Abbildung 3.12 zeigt eine vereinfachte Lieferkette, die von Rohstoffen über Lieferanten zu einem Produkt werden und dann über Groß- und Einzelhandel an die Endkunden gelangen.

Viele Lieferketten erfordern heute eine vollständige Rückverfolgbarkeit und Transparenz aller Prozessschritte, wie z.B. in der Landwirtschaft ([Costa u. a. 2013]) oder der Medizin ([Benedetti u. a. 2014]). Auch die Fairtrade-, Bio- oder Organic-Bewegungen könnten von transparenten und nachvollziehbaren Prozessen profitieren, während heute Kunden ohne Einblick in die Pro-

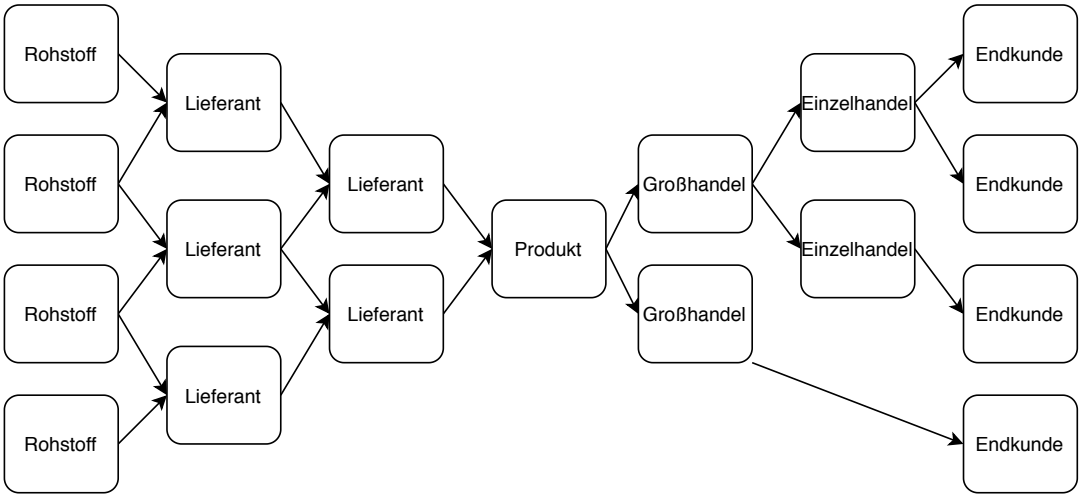

Abbildung 3.12.: Beispiel einer Lieferkette

zesse blind auf die Fairtrade und Öko-Logos auf den Packungen vertrauen müssen [Abeyratne und Monfared 2016].

Lieferketten beinhalten verschiedene Entitäten, Personen, Verträge, physische Ressourcen und Transaktionen, die es ermöglichen, ein Produkt vom Lieferanten an den Kunden zu bringen. In einer großen Lieferkette ist es deshalb schwierig, ein Gesamtbild über alle Prozesse an den Endkunden zu vermitteln [Haq u. a. 2010], da einige dieser Informationen typischerweise in mehreren Stellen zentral bei den jeweiligen Lieferanten verwaltet werden.

Die Modellierung einer Lieferkette mittels einer Blockchain-Technologie bringt deshalb einige Vorteile mit sich. Hauptsächlich wird durch die dezentrale Natur Transparenz und Nachvollziehbarkeit geschaffen. Anstelle mehrerer voneinander getrennter Systeme fungiert eine Blockchain hier als globales System für alle Parteien einer Lieferkette. Solange dem Endkunden Zugang auf das System gewährt wird, stehen dort alle Informationen zur Verfügung.

Das Double-Spending Problem in Lieferketten wird ebenfalls durch Nachvollziehbarkeit eliminiert. Durch vollständige Dokumentation der Prozesse kann aus 1 kg zertifizierten Kaffeebohnen nicht über Nacht 2 kg gemahlenes

zertifiziertes Kaffeepulver werden. Innerhalb der Validierung können durch fest definierte Regeln die Prozesse so formuliert werden, dass sie nicht von bestimmten Mustern abweichen dürfen, um als valide akzeptiert zu werden. Zusätzlich kann mit der Wahl von einer private-permissioned Blockchain bestimmten Lieferanten nur die Aktionen gestattet werden, die zu ihren Tätigkeiten innerhalb der Lieferkette passen.

3.6.3. E-Marketplace

Unter E-Marketplaces wird der virtuelle Handelsraum eines Marktplatzbetreibers verstanden, dem Anbieter und Nachfrager von Produkten digital beitreten können [Kollmann 2019]. Meist treten Betreiber eines Marktplatzes als Plattformbetreiber (z.B. eBay) auf, können aber auch selbst als Teilnehmer (z.B. Amazon) partizipieren. Betreiber der Marktplätze fungieren vielfach als notwendiger Intermediär, der Zahlungen zwischen Verkäufer und Käufer abwickelt, als Streitschlichter in Problemfällen und als Anbieter der nötigen Infrastruktur. Dafür erhält er eine Vergütung in Form von Gewinnbeteiligung oder eines monatlichen Beitrags durch die Händler.

Vielfach sind diese Plattformbetreiber notwendige Intermediäre, um eine vertrauensvolle Instanz in den Prozess zwischen zwei sich gegenseitig nicht vertrauenden Parteien einzubringen. In einigen Prozessen, wie im Energiehandel, gibt es noch weitere Intermediäre, die zusätzliche Gebühren auferlegen. Konkret sind in diesem Fall neben dem Börsenbetreiber noch zusätzlich Clearinghäuser, Broker, Indexagenturen und weitere in den Prozess eingebunden [Merz 2019].

Im Energiehandel erfüllen viele dieser Intermediäre eine spezifische für den Prozess essenzielle Rolle. Zum Beispiel sorgt die Börse als Plattformbetreiber dafür, dass Angebote nicht mehrfach gekauft oder verkauft werden. Sie sorgen hier also zentral für die Lösung des Double-Spending Problems, indem sie entscheiden, welcher Kauf zuerst getätigt wurde, wenn zwei Käufer (exakt) gleichzeitig bieten.

Viele der notwendigen Intermediäre können durch eine Blockchain-Lösung ersetzt werden. Die Transparenz der Transaktionen auf der Blockchain und der dezentrale Konsensalgorithmus sorgen für die Sicherheit der Prozesse. Käufe und Verkäufe sind sequentiell und fälschungssicher in der Historie für alle sichtbar belegt. Kauf-Limits, User-Bewertungen und weitere funktionale und nicht-funktionale Anforderungen können ebenso in den Prozess auf der Blockchain integriert werden.

3.6.4. Digitale Identität

Im Kern unserer aktuellen serviceorientierten Ökonomie sind internetbasierte Transaktionen. In vielen Fällen werden diese automatisch ausgelöst und bearbeitet. Anders als im Einzelhandel können Transaktionen im Internet hingegen nicht zwischen zwei sich unbekannten Teilnehmern durchgeführt werden, es muss immer bekannt sein, wer eine bestimmte Leistung erhält. Dafür ist es notwendig, irgendeine Form der Identität zu erbringen, wobei es nicht zwingend der Name eines Nutzers sein muss [Windley 2005].

In der analogen Welt gibt es diverse Möglichkeiten sich zu identifizieren, wie zum Beispiel: Ausweis, Führerschein, Reisepass oder andere national anerkannte Dokumente. Diesen ist gemein, dass in vielen Nationen die Hürden für die Fälschung dieser Dokumente nicht besonders hoch sind [Rivera u. a. 2017]. Zusätzlich stellt auch der Verlust dieser Dokumente ein Risiko für die Identifikation dar, da sich nun nicht authorisierte Personen damit identifizieren, authentisieren oder authentifizieren können.

Die vorteilhaften Eigenschaften, die man sich von der Blockchain für die digitale Identität verspricht, sind, wie in anderen Anwendungsfällen auch, die Sicherheit, Datenintegrität und Anonymität unter dem Wegfall der Notwendigkeit von dritten Parteien. Über eine gemeinsame Blockchain-basierte Plattform könnten mehrere Webseiten Konten für ihre Dienste verwalten. Den Nutzern wird so erspart, sich mehrfach mit verschiedenen Kennungen und Passwörtern zu registrieren. Stattdessen würden sie eine dezentrale Iden-

tität auf dieser Blockchain anlegen, die mittels Public/Private-Keys verwaltet wird und zur Authentifizierung würde eine Challenge gegen den Public Key gestellt, die nur mittels des Private Keys gelöst werden kann. Allgemein gesehen, stellt die Blockchain nicht die Funktion der Authentifizierung zur Verfügung, sondern sie dient lediglich als dezentraler und unveränderbarer Datenspeicher.

Grundsätzlich sollten nie die kompletten Daten in der Blockchain gespeichert werden. Da hier durchaus sicherheitskritische und personenbezogene Daten vorliegen, sollten diese nur referenziert werden, denn ein späteres Entfernen ist aufgrund der Immutabilitätseigenschaft der Blockchain nicht möglich. Um eine Datenreferenz anzulegen, wird ein Hashwert über die Daten gebildet und nur dieser in der Blockchain-Historie gespeichert, die echten Daten werden anderweitig sicher hinterlegt. Über Challenges und Hashwert Nachweise kann ein Nutzer jederzeit beweisen, dass er im Besitz dieser Daten ist.

Ein darüber hinausgehender Anwendungsfall ist das Identitäts- und Zugriffsmanagement, bei dem fein granular hinterlegt wird, wer auf welche Informationen unter spezifischen Bedingungen zugreifen darf. Nutzer können zu späteren Zeitpunkten jederzeit mittels einer Transaktion die Zugriffsberechtigungen anpassen.

3.6.5. Voting

Wahlen werden heutzutage noch überwiegend analog auf Papier und in Wahlkabinen durchgeführt. Stimmen müssen manuell gezählt werden, was zu Fehlern führen kann. Die Auszählungen müssen durch mehrere Personen (Vier-Augen-Prinzip) beaufsichtigt werden, um Missbrauch vorzubeugen.

Bei den Bundestagswahlen gibt es strikte Regelungen in welcher Form, wann und wie gezählt wird. Stimmzettel werden mehrfach in verschiedenen Runden gezählt und mit Wählerverzeichnissen abgeglichen. Auch die eigentli-

chen Stimmen, getrennt nach Erst- und Zweitstimme werden mehrfach manuell gezählt und abgeglichen (§60, §67 ff. BWO). Die Bekanntgabe von vorläufigen Ergebnissen kann erst nach Wahlschluss bekanntgegeben werden, da eine Öffnung der Wahlurne und vorzeitige Zählung nicht gestattet ist.

Viele dieser analogen Verfahren könnten potentiell durch Digitale ersetzt werden. Im Falle der Bundestagswahl ist zu beachten, dass dabei die Prozesse eingehalten sowie die durch das Grundgesetz zugesicherten Rechte einer allgemeinen, unmittelbaren, freien, gleichen und geheimen Wahl (Art 38 GGS) eingehalten werden. Zusätzlich müssen Wahlen und Wahlauszählungen öffentlich einsehbar und kontrollierbar sein, was derzeit nicht gewährleistet ist [Bundeswahlleiter 2015]. Der Einsatz von Wahlcomputern oder anderen internetbasierten Verfahren benötigt zusätzliche Absicherung gegen Manipulation in den Wahlgeräten, der Übertragungswege und der zentralen Wahlspeicher.

Um die Idee eines Blockchain-basierten Wahlverfahrens umzusetzen, sollten die digitalen Prozesse stark die der analogen Systeme reflektieren. Besonders hinsichtlich des Zuganges zur Wahl muss dies streng auf eins limitiert sein. Jedem Nutzer darf nur einmal die Möglichkeit gegeben werden an der Wahl teilzunehmen. Um Sybil Attacken ([Douceur 2002]) vorzubeugen, sollten Nutzer authentisiert werden, aber zugleich muss bei einer geheimen Wahl deren Identität beim Wählen unkenntlich sein. Es muss auch sichergestellt werden, dass der Wähler weiß, dass seine Stimme erfolgreich abgegeben und gezählt wurde und dass keine einzelne Entität die Stimmen zusammenzählt und das Ergebnis entscheidet [Hjálmarsson u. a. 2018].

Viele der Eigenschaften, die an aktuellen elektronischen Wahlverfahren kritisiert werden, können durch eine Blockchain-basierte Lösung verbessert werden. Die vollständige Transparenz aller Transaktionen sorgt für die notwendige Sicherheit sowie für die Möglichkeit, der Öffentlichkeit die Ergebnisse selbst nachzuvollziehen. Um die Identität der Wähler geheim zu halten, aber gleichzeitig für Authentizität zu sorgen, können gängige Verfahren aus der Kryptologie verwendet werden, wie etwa *Zero-Knowledge-Proofs*.

3.6.6. Assets und Tokens

Bei Digital- oder Kryptowährungen handelt es sich genau genommen nur um die digitale Darstellung von Werten. Demzufolge lassen sich auch beliebige andere Werte oder Besitztümer in einer Blockchain darstellen und auch deren Besitz transferieren. Im Zusammenhang mit Blockchains werden hierfür häufig die Begriffe *Asset* oder *Token* verwendet.

Ein digital repräsentierter Token kann innerhalb der Blockchain zwischen den Akteuren weitergegeben, aufgeteilt oder auch aufgebraucht werden, je nach vorliegendem Anwendungsfall. Beispielsweise könnte der Besitz von Gemälden oder Immobilien mit einer digitalen Representation als Asset oder Token eindeutig nachgewiesen werden. Die Notarisierung beim Anwalt und die damit verbundenen Kosten würden hier komplett entfallen, da die Beglaubigung des Besitzerwechsels durch den Konsensalgorithmus der Blockchain stattfindet.

Besonders in den Medienfokus gerückt, sind mit der Veröffentlichung von Ethereum Crowdfundings und Initial-Coin-Offerings (ICO, vgl. Initial-Public-Offering, dt.: Börsengang). Bei ICOs versucht ein Startup initial Geld für seine Idee, Umsetzung oder Produkt zu sammeln. Die Anleger erhalten dafür im Gegensatz Tokens. Je nach Art des ICOs bestimmen die Tokens, was sie bedeuten, so können sie Anteile an der Firma repräsentieren, ein Anteil an Stimmen in einem Entscheidungsprozess oder auch nur ein Vorkaufsrecht für das spätere Produkt. Generell werden hierzu standardisierte Smart Contracts (ERC20) auf Ethereum-basis verwendet. Nutzer zahlen die Blockchain-eigene Währung (hier Ether) an den Smart Contract und dieser konvertiert die Einzahlung in eine Menge an Tokens. Die Tokens sind immer nur innerhalb der Logik des Smart Contracts verwendbar.

Zusammenfassung

Die Blockchain-Technologie ermöglicht es, sich in verteilten Systemen mit mehreren unbekannten Teilnehmern auf einen gemeinsamen Systemzustand zu einigen. Dies geschieht vollkommen dezentral ohne eine dritte Instanz oder Intermediäre. Um dies zu ermöglichen, werden je nach Technologie verschiedene verteilte Konsensalgorithmen, wie Proof-of-Work, eingesetzt.

Die Zusammenfassung von Transaktionen in Blöcke schafft zuallererst eine distinkte Reihenfolge aller auszuführenden Veränderungen am globalen Systemzustand. Weiterhin sorgen Prüfsummen (Hashwerte) dafür, dass eine Veränderung an diesen Inhalten sofort durch falsche Prüfsummen auffällt. Die Verkettung neuer Blöcken, zu ihren Vorgängerblöcken anhand weiterer Prüfsummen erzeugt eine Unveränderbarkeit aller alten Werte, da bereits ein abweichendes Bit für falsche Hashwerte sorgt und von der Veränderung bis in den aktuellsten Block kaskadiert.

Entstanden ist die Idee für die heute als Blockchain bekannte Technologie, für den Transfer der digitalen Währung Bitcoin ohne auf Intermediäre, wie Banken, als Vertrauensinstanzen zurückgreifen zu müssen. Weiterentwicklungen wie neue Funktionalitäten oder besserer und schnellerer Konsensalgorithmen haben zumeist auch digitale Währungen als einzigen Anwendungszweck im Blick. Mit Ethereum entstand die zweite Generation der Blockchain-Technologien, die es erstmals ermöglichte, auch eine verteilte Codeausführung mit Hilfe von Smart Contracts zu gewährleisten. Mit Smart Contracts lassen sich viele Anwendungen, wie Marktplätze, Voting oder Tokens umsetzen. Sie haben jedoch auch den Nachteil, dass sie bestimmten Laufzeit- und Funktionsbeschränkungen unterliegen und somit nicht für jeden Geschäftsprozess eingesetzt werden können.

Für die Umsetzung von beliebigen Geschäftsprozessen bedarf es also einer Blockchain-Technologie, die einerseits nicht nur für digitale Währungen ausgelegt ist und andererseits auch die Ausführung von Maschinencode unterstützt, der deutlich komplexer ist. Im nachfolgenden Kapitel 4 soll deshalb

zuerst eine allgemeine Darstellung der Herausforderungen für die Blockchain-Technologie im Bezug auf Geschäftsprozesse und weitergehend eine genaue Analyse der bereits bestehenden Technologien erfolgen.

4. Analyse bestehender Blockchain-Technologien

Anschließend an die Grundlagen aus den vorhergehenden Kapitel zu Geschäftsprozessen, Konsensalgorithmen und der Blockchain-Technologie sollen nun im folgenden Kapitel die generellen Herausforderungen beim Einsatz von Blockchain-Technologien herausgestellt und die bereits bestehenden Technologien analysiert werden.

Hierfür werden zuerst die Kern-Herausforderungen, die beim Einsatz der Blockchain-Technologie aber auch für andere IT-Systeme gelten, vorgestellt, wie beispielsweise Sicherheit und Skalierbarkeit. Aber auch Blockchain-spezifische Herausforderungen, wie etwa der Umgang mit Smart Contracts, werden betrachtet. Danach wird untersucht, aus welchen Kern- und Zusatzkomponenten verschiedene Blockchain-Frameworks bestehen und wie deren Architektur sich zusammensetzt. Weiterhin werden die Prozesse innerhalb der Blockchains dargelegt. Dazu gehören unter anderem die Netzwerkstrukturen, Blockbildungs- und Validierungsschritte, die ultimativ jegliche Transaktion innerhalb der Blockchain betrifft. Ziel dieses Kapitels ist die Gemeinsamkeiten der Architekturen und die ablaufenden Prozesse verschiedener Blockchains aufzuzeigen und eventuelle Stärken und Schwächen zu identifizieren, um diese im nachfolgenden Kapitel in die Erarbeitung eines eigenen generalisierbaren Blockchain-Frameworks mit einzubeziehen.

4.1. Herausforderungen für den Einsatz in Geschäftsprozessen

Auch für die Blockchain-Technologie gilt, dass sie noch lange nicht für alle Einsatzgebiete hinreichend geeignet ist. Viele offene und ungelöste Herausforderungen verhindern den effektiven Einsatz in vielen Geschäftsprozessanwendungen. Entweder lassen sich die Prozesse anpassen, sodass ein Blockchaineinsatz mit gewissen Einschränkungen ermöglicht wird oder es wird auf alternative bewährte Technologien zurückgegriffen. Beispielsweise ermöglicht der Einsatz einer public Blockchain, dass viele Knoten die Transaktionen verifizieren und für einen globalen Konsens und deren Gültigkeit sorgen, andererseits bedeutet das auch für Privatsphäre-relevante Daten, dass viele Knoten die Transaktionen sehen. Woraus folgend das „Blockchain-Trilemma“ geprägt wurde.

4.1.1. Blockchain-Trilemma

Das Blockchain-Trilemma besagt, dass es für eine Blockchain unmöglich ist, alle der drei Eigenschaften *Dezentralisierung*, *Skalierbarkeit* und *Sicherheit* gleichzeitig vollständig zu erfüllen. Bestehende Public Blockchains, wie Bitcoin und Ethereum, sind nicht beliebig skalierbar, da sie auf maximale Dezentralisierung und Sicherheit Wert legen. Blöcke können nicht beliebig groß werden, da die Nachrichtenlaufzeiten im Netz ansteigen würden und keine distinkte Blocktopologie zwischen den Teilnehmern entstehen kann. Hingegen vernachlässigen Private- bzw. Konsortiums-Blockchains, wie Hyperledger, die Dezentralisierung durch einen feststehenden Teilnehmerkreis. Die Menge der Nachrichten und die Netzwerknachrichtenlaufzeiten werden so verringert, um folglich einen höheren Transaktionsdurchsatz zu ermöglichen. Meist resultiert dies in mehr Transaktionen pro Block oder kleineren Inter-Blockzeiten, also verändert es die Blockgröße oder die Blockfrequenz [Abadi und Brunnermeier 2018; Gomez 2017]. Abbildung 4.1 verdeutlicht nochmals

diese Zusammenhänge zwischen Sicherheit, Skalierbarkeit und Dezentralisierung mit ihren jeweiligen primären Anwendungsgebieten.

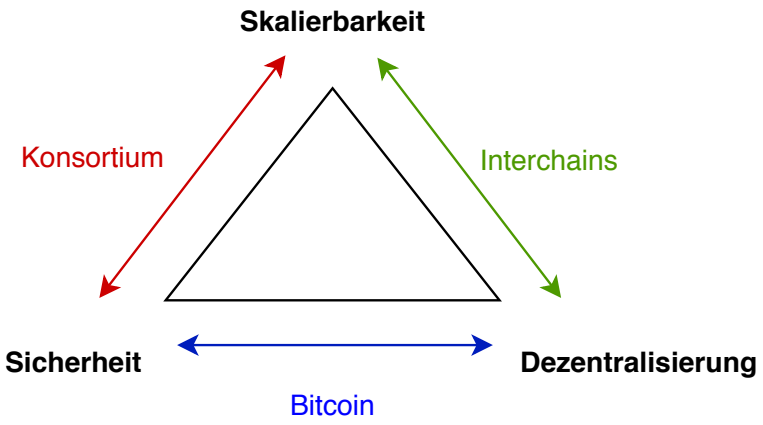

Abbildung 4.1.: Blockchain-Trilemma

4.1.2. Skalierbarkeit

Skalierbarkeit ist ein wünschenswertes Attribut eines Netzwerkes, Systems oder Prozesses, das die Fähigkeit beschreibt, eine ansteigende Anzahl von Elementen oder Objekten unterzubringen, um eine wachsende Menge von Arbeit problemlos handzuhaben oder besonders anfällig dafür zu sein [Bondi 2000].

Es gibt verschiedene Arten, wie man ein System praktikabel skalieren kann. Dabei lassen sich Skalierungsmethoden in horizontal und vertikal aufteilen. Unter vertikale Skalierung fallen Maßnahmen, die innerhalb einer Entität, beispielsweise durch das Hinzufügen von mehr Speicherplatz oder zusätzlichen virtuellen Maschinen, stattfinden. Horizontale Skalierung bedeutet aus Hardwaresicht hingegen, dass neue weitere Knoten ins Gesamtsystem aufgenommen werden [Bondi 2000].

Skalierung in Blockchains kann auf verschiedene Weise erreicht werden, ist aber durch die jeweiligen Design-Entscheidungen der gewählten Technolo-

gie begrenzt. Blockchains, die auf Proof of Work setzen, können beispielsweise skaliert werden, indem die Rechenleistung der Mining-Rechner erhöht wird, was aber nicht zur Skalierung des Netzwerkes beiträgt, sondern primär dem jeweiligen Miner dient. Zusätzlich sind durch die Selbstregulierung der „Difficulty“ in PoW Konsensprotokollen die Blockzeiten quasi gleichbleibend. Um in diesem Szenario also den Transaktionsdurchsatz zu erhöhen, gibt es nur zwei Möglichkeiten, die das Konsensprotokoll betreffen. Durch Verringerung der Difficulty kann die Blockzeit verringert werden, was zu einem höheren Durchsatz führt. Durch eine Erhöhung der maximalen Blockgröße können mehr Transaktionen in einen Block gefasst werden, was ebenso einen höheren Durchsatz bedeutet. Ein Beispiel für die Erhöhung der Blockgröße kann in Bitcoin in Form des sogenannten *SegWit* (Segregated Witness, dt.: getrennter Zeuge) gefunden werden. Durch SegWit wird es in diesem Fall ermöglicht mehr Transaktionen in einem Block unterzubringen. Jedoch passiert dies nicht durch die eigentliche Vergrößerung des Blocklimits, sondern durch eine veränderte Größendefinition von Transaktionen. Indem nur die Nutzdaten einer Transaktion und nicht mehr zusätzlich die Signaturdaten als Transaktionsgröße gezählt werden, verringert sich faktisch die Gesamtgröße der Transaktion und ermöglicht so, dass mehr Transaktionen in einem Block inkludiert werden können [Lombrozo u. a. 2015].

Andere Konsensprotokolle stehen vor anderen Skalierungsproblemen, die ebenfalls nicht durch zusätzliche Rechenleistung gelöst werden können, teilweise das Problem sogar noch verschlechtern. Wie auch PBFT benötigen BFT-PoS-basierte Konsensprotokolle einen deutlich höheren Nachrichtenaustausch pro Block. Ohne Verbesserungen am Konsensprotokoll hat der Nachrichtenaustausch bei PBFT eine Komplexität von $\mathcal{O}(n^2)$. Für BFT-PoS bedeutet dies, dass mit jedem hinzukommenden Validator die Nachrichtenanzahl exponentiell steigt, sodass ab einer gewissen Anzahl jede weitere Steigerung die Netzwerkperformanz degradiert. BFT-basierte Protokolle müssen zur Steigerung der Performanz den Validatorkreis möglichst klein halten, was hingegen der Dezentralität des Netzwerkes schadet [Vukolić 2015].

Im Bereich des *Internets der Dinge* (IoT) stehen Blockchain-Technologien auch noch vor einem anderen Skalierungsproblem. Die ständig größer werdende Blockchain ist vor allem für IoT Geräte mit geringem Speicher problematisch. Auch wenn nicht zwingend alle Geräte eine komplette Kopie der ganzen Historie benötigen, behindert dies jedoch einen Blockchain-Einsatz stark. Ein Ausweichen auf Serverhardware zur Verwaltung der kompletten Historie würde das Gesamtsystem zunehmend zentralisieren und von der Verfügbarkeit dieser abhängig machen. Zusätzlich verhindert das ständige Wachsen ein schnelles Hinzufügen von neuen Knoten. Diese müssen erst aufwendig die bisherige Historie synchronisieren, um aktuelle Transaktionen ausführen zu können [Reyna u. a. 2018].

4.1.3. Sicherheit

Neben den bisher vorgestellten möglichen Attacken auf Blockchain-Systeme, wie Double-Spending, 51%-Attacke oder Long-Range Attacke, gelten auch für Blockchains dieselben Netzwerkangriffsvektoren wie in anderen verteilten Systemen. Denial of Service, Man in the Middle und auch Sybil Attacken können Blockchain-Netzwerke genauso in der Funktion stören wie andere Netzwerke [Saad u. a. 2020].

Unter dem Sicherheitsaspekt fällt auch die mögliche Zentralisierung des Netzwerkes. Bei zunehmender Zentralisierung besteht die Gefahr von negativer Einflussnahme durch bestimmte Netzwerkakteure, wie beispielsweise Mining Pools, auf Bevorzugung von bestimmten Transaktionen, koordiniertes Zurückhalten von Blöcken oder auch von generellen Protokolländerungen [Kaiser u. a. 2018].

Auch was die Privatsphäre von Nutzern angeht, bieten hier öffentliche Blockchains nicht den hierbei teilweise gewünschten Schutz. Obwohl Transaktionen in Kryptowährungen zwischen gegenseitig unbekannten Teilnehmern durchgeführt werden können, sind hier die Teilnehmer dennoch nicht anonym. Die Nutzung von asymmetrischer Kryptografie verschleiert hier nur

die echten Identitäten zu Pseudonymen. Abhilfe sollen hier unterschiedliche Einmalschlüssel schaffen, welche sich aber durch Flussanalysen auch zu einer Identität zusammenführen lassen. Schlussendlich sorgt der Austausch von Kryptowährungen in Börsen zu Fiatgeld für die Aufdeckung der echten Identität [Koshy u. a. 2014].

Die Nutzdaten der Transaktionen, neben Sender und Empfänger, müssen auch einsehbar sein, denn nur so kann deren Gültigkeit validiert werden. Während in Kryptowährungen und tokenbasierten Blockchain-Technologien beispielsweise homomorphe Verschlüsselung und Zero-Knowledge-Proofs eingesetzt werden können, um Inhalte zu verschleiern und weiterhin für Validierbarkeit zu sorgen, ist dies nicht in allen denkbaren Anwendungsfällen sinnvoll einsetzbar [Gentry 2009; Sasson u. a. 2014].

4.1.4. Smart Contracts

Obwohl sich die Entwicklung und Programmierung von Smart Contracts in den meisten Fällen nicht von der in anderen Softwarebereichen unterscheidet, erfordert sie dennoch besondere Sorgfalt. Vor allem müssen Entwickler verstehen, dass jeder kleine Programmierfehler potentiell gravierende monetäre Folgen mit sich zieht, wie beispielsweise im „*DAO-Hack*“ ([Mehar u. a. 2019]) geschehen. Dabei ist eins der größten Probleme, dass Smart Contracts praktisch nicht aktualisiert werden können, denn wenn sie einmal in der Blockchain gespeichert sind, kann der Zustand zwar verändert werden, jedoch nicht mehr ihr Programmcode. Smart Contracts unterliegen also besonderen Herausforderungen bezüglich Code-Qualität und dessen Testung. [Atzei u. a. 2017; Delmolino u. a. 2016].

Zusätzlich zu Programmierfehlern sorgt die gewählte Netzwerk– und Zugriffsform von Blockchains, auf denen Smart Contracts ausgeführt werden, für Probleme. Meist sind diese in der Form von Public und Permissionless anzutreffen, wie etwa die Ethereum Blockchain. Hierdurch sind Smart Contracts jedem beliebigen bösartigen Teilnehmern ausgesetzt. Diese können zwar

nicht die Ausführung der Smart Contracts verändern, denn der Programmcode kann nicht verändert werden, jedoch können Fehler gezielt ausgenutzt werden. Das gezielte Suchen nach Fehlern ist zusätzlich begünstigt, dadurch dass der Programmcode in Form des Bytecodes für jeden einsehbar und lokal ausführbar ist [Christidis und Devetsikiotis 2016].

Auch können „Denial of Service"-Angriffe in Form von sog. Flood-Attacken gezielt ausgeführt werden. Das Fluten des Netzwerkes mit Transaktionen kann beispielsweise bei Auktionen verwendet werden, indem ein Angreifer an der Auktion teilnimmt und danach so viele andere Transaktionen ins Netzwerk einstellt, die wegen höherer Transaktionsgebühren bevorzugt von Minern in Blöcke aufgenommen werden. So kann ein Angreifer effektiv verhindern, dass andere Teilnehmer überhaupt noch an der Auktion teilnehmen können, da ihre Transaktionen eine niedrigere Priorität haben [Chervinski u. a. 2019; Saad u. a. 2019].

Die effiziente Ausführung der Programmlogik von Smart Contracts ist ebenfalls eine offene Herausforderung. Um ein hohes Vertrauen in die Richtigkeit der Blockchain-Daten legen zu können, muss jeder Teilnehmer die Transaktionen selbst überprüfen, also auch die Ausführung des Smart Contracts. Der Zustand eines Smart Contracts wird zwar innerhalb der Blockchain gespeichert, aber jede neue Zustandsänderung muss wiederum geprüft werden. Dies führt zu stark redundanter Ausführung von Logik, die nicht zwingend für jeden Teilnehmer von Relevanz ist. Eine selektive Ausführung nur auf einem Teil der Knoten würde wiederum die Sicherheit und Dezentralität gefährden, da nun größeres Vertrauen in eben diese gelegt werden muss [Dickerson u. a. 2019].

Ein weiterer Nachteil ist, dass Programmlogik sequentiell ausgeführt werden muss, damit die erreichten Zustände immer deterministisch sind. Es ist jedoch möglich, bestimmte Teile parallel auszuführen. Dazu muss vorher analysiert, werden welche Smart Contracts sich parallelisieren lassen, beispielsweise ob sie mit denselben Nutzer-Accounts interagieren und ob Transaktionen mit unterschiedlichen Smart Contracts interagieren [Yu u. a. 2017].

Auch die spekulative Ausführung von Smart Contract Logik kann einer besseren Parallelisierung dienen [Dickerson u. a. 2019]. Der größte Flaschenhals bleibt hier jedoch weiterhin, wie vorher beschrieben, die Notwendigkeit der mehrfachen Ausführung derselben Logik von allen Netzwerkteilnehmern aus Sicherheits- und Vertrauensgründen.

Geschützte Geschäftsprozesse sind durch eine Smart Contract Logik ebenfalls schwer umzusetzen. Da der Bytecode eines kompilierten Smart Contracts für alle Teilnehmer mit Zugang frei einsehbar ist, wäre es denkbar, dass mit genügend Aufwand Prozesse rekonstruiert werden können, die einem Konkurrenten eigentlich geheimgehalten werden sollten. Verfahren, den Programmcode in verschlüsselter Form in der Blockchain abzulegen existieren, aber sie reduzieren deutlich den Kreis der potentiellen Teilnehmer, die diese ausführen und validieren. Durch den Einsatz von Gruppenschlüsseln und Multi-Signaturen wird in diesem Szenario einerseits der Zugriff auf Progammlogik reguliert und andererseits für die korrekte Ausführung durch die Teilnehmer gebürgt.

4.2. Komponenten

Im Bereich der Softwarearchitektur beschreibt eine *Komponente* einen Teil der gesamten Software, mit denen andere Komponenten agieren können. Eine einzelne Komponenten, innerhalb der ganzen Software, ist möglichst losgelöst von den anderen, um eine starke Kopplung zu verhindern. Eine Komponentenorientierte Softwarearchitektur begünstigt weiterhin eine Trennung nach Verantwortlichkeiten, sodass jede Komponente auch nur für eine grundlegende Funktion innerhalb des Systems zuständig ist.

4.2.1. Basis-Komponenten

Beschränkt man sich bei der Betrachtung einer Blockchain-Technologie rein auf die dabei verwendeten und miteinander agierenden Softwarekomponenten, unterscheiden sich die meisten aktuellen Technologien nicht grundlegend voneinander. Allgemein betrachtet, sind die vier Hauptkomponenten einer Blockchain der Mempool, eine Persistenzkomponente, die Kommunikationskomponente und das Konsensprotokoll. Zusätzlich zu diesen vier Komponenten wird eine Anwendungsschicht benötigt, die den semantischen Kontext für die Nutzdaten liefert. Ohne sie kann keine Validierung der Transaktionen vorgenommen werden.

Persistenz

Die Persistenzkomponente in einer Blockchain ist für das permanente, nichtflüchtige Speichern von Daten verantwortlich. In den wenigsten Technologien werden klassische SQL-basierte Datenbanksysteme eingesetzt, wie beispielsweise *PostgreSQL* oder *MySQL*. Eine komplexe Abfragesprache und Abfragesyntax sind in den wenigsten Fällen notwendig, um Daten zu finden. Teilweise werden auch Rohdaten direkt auf der Festplatte gespeichert.

Transaktionen, Blöcke und teilweise auch Nutzdaten werden ausschließlich über deren Hashwerte oder IDs referenziert und miteinander verknüpft. Aus diesem Grund werden fast ausschließlich Key-Value Datenbanken verwendet, wie etwa *LevelDB* oder *RocksDB*. Eine Aufteilung der Blockchain-relevanten Daten findet in vielen Fällen nur nach Blöcken und Transaktionen statt.

Zusätzlich zur Persistenz auf Blockchain-Ebene ist eine Datenspeicherung auf Anwendungsebene erforderlich, welche auf den Anwendungskontext abgestimmt sein sollte.

Memory Pool

Der Memory Pool, kurz Mempool, ist eine Basiskomponente der Blockchain-Technologie, die dazu dient, Transaktionen flüchtig zwischenzuspeichern [Bitcoin Community 2018]. Bevor eine Transaktion in den Mempool aufgenommen wird, wird diese auf Gültigkeit geprüft, d.h. sowohl ob sie eigenständig gültig ist, als auch, ob sie in Bezug auf die Historie bzw. den aktuellen Anwendungszustand gültig ist. Für die Blockbildung bedient sich ein Teilnehmer der Transaktionen aus dem Mempool. Weitergehend löscht der Mempool selbstständig periodisch Transaktionen, die bereits zu lange auf Inklusion in einen Block warten oder bereits ungültig geworden sind.

Kommunikation

Damit Blöcke und Transaktionen verteilt werden können, benötigt jedes Framework eine Kommunikationskomponente, die es ermöglicht, in einem Peer-to-Peer Netzwerk zu agieren. Zu den Aufgaben der Kommunikationskomponente gehören, je nach Ausprägung, neben der reinen Übermittlung von Nachrichten auch das Peer-Management.

Unter das Peer-Management fallen alle Aufgaben, die es ermöglichen, einerseits neue Peers zu entdecken (Peer-Discovery), aber andererseits auch selbst entdeckt zu werden. Innerhalb der Peer-Discovery gibt es unterschiedliche Formen des Auffindens, wie beispielsweise hartcodierte Super Peers, DNS Seeding oder Anfragen über Nachbarknoten. Hierzu werden häufig Adressbücher geführt, welche bei Bedarf oder in regelmäßigen Abständen mit Nachbarknoten ausgetauscht werden, um für eine bessere Vermaschung im Netzwerk zu sorgen.

Konsens

Die Konsenskomponente ist das Herzstück eines Blockchain-Frameworks. Hier findet jedoch nicht nur der reine Prozess der Konsensfindung statt. Je nachdem, ob ein Knoten aktiv oder passiv am Konsens teilnimmt, werden hier auch die Prozesse des Minings und des Erstellens von Blöcken ausgeführt. Dazu müssen sowohl offene Transaktionen aus dem Mempool bezogen, als auch potentielle Blockkandidaten über das Netzwerk propagiert werden. Genauso müssen einkommende Blockkandidaten auf Gültigkeit geprüft werden, bevor diese an die aktuelle Historie angehängt werden, beziehungsweise in BFT-PoS muss darüber zusätzlich vorher abgestimmt werden. Während die Persistenzkomponente allein gestellt ist, muss die Konsenskomponente stark mit den anderen Komponenten gekoppelt werden, um korrekt zu funktionieren.

4.2.2. Komponenten am Beispiel Bitcoin

Als erste Blockchain überhaupt gilt Bitcoin. Diese stellte auch für andere Nachfolger die Grundlage dar, sowohl was Funktionalitäten und Architektur betrifft. Die erste Implementation von Bitcoin erschien bereits 2009 und besitzt bis heute noch eine stark verbundene Architektur, ohne dass die Komponenten klar voneinander getrennt oder erkennbar sind.

Aufgrund der Auslegung auf einen einzigen Anwendungsfall, das Transferieren von Währung, findet sich keine Entkopplung von Anwendungslogik und reiner Blockchainlogik im Bitcoin Framework. Abbildung 4.2 zeigt die Architektur des Bitcoin Frameworks, aufgeteilt nach Aufgaben der Komponenten. Wie hier dargestellt, verteilt sich die notwendige Konsenskomponente auf mehrere Schichten und Einzelkomponenten der Architektur.

Zur Persistenz verwendet Bitcoin mehrere eigenständige Datenbanken in der Form von Key-Value Stores. Die kompletten Blöcke werden dabei in Rohdaten innerhalb eigener Dateien (`blk.dat`) abgelegt und die Block-Index Da-

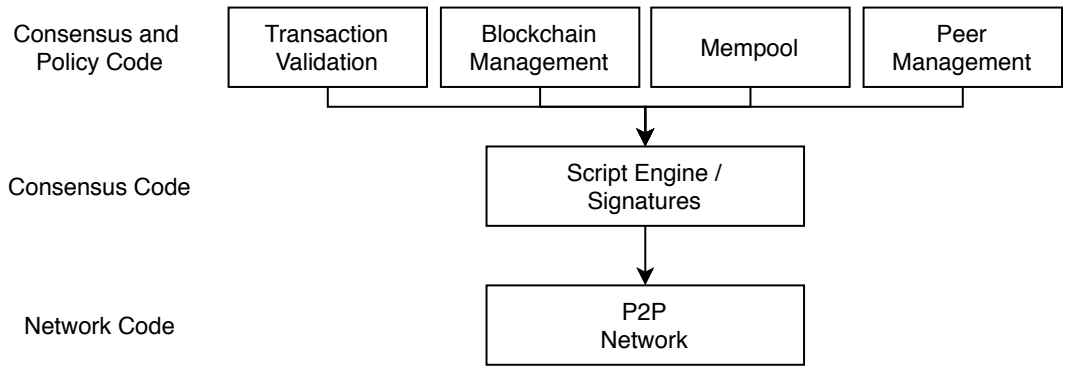

Abbildung 4.2.: Bitcoin Architektur

tenbank verwaltet den Speicherort dieser. Roh-Transaktionen werden nicht separat gespeichert. Da bei Bitcoin durch den eigesetzten PoW-Algorithmus Forks entstehen können, wird eine Möglichkeit benötigt, bereits getätigte Transaktionen rückabzuwickeln. Hierfür wird für jede Transaktion eines Blockes eine „Umkehr"-Transaktion innerhalb eines Rohblockes angelegt (`rev.dat`). Bei der Rückabwicklung werden alle Blöcke mit den Umkehrtransaktionen in der richtigen Reihenfolge ausgeführt.

Betrachtet aus der Anwendungsebene von Bitcoin, benötigt der Konsens semantische Informationen über den aktuellen Zustand der Blockchain. Dazu werden im `chainstate` Informationen darüber gespeichert, welche Transaktionen bisher nicht als Output in einer anderen Transaktion vorkommen. Zur Validierung einer neuen Transaktion kann so schneller geprüft werden, ob diese gültig ist, um zu vermeiden die gesamte Historie durchzugehen.

Obwohl Anwendungs- und Blockchainlogik stark miteinander verbunden sind, lässt sich anhand der Nutzung der verschiedenen Persistenzmechanismen eine Trennung erkennen. Für die reine Blockchainlogik werden die Rohblöcke gespeichert und mittels Block-Index verwaltet. Für die Anwendungslogik kommen der Chainstate mit den offenen Transaktionen und die jeweiligen Umkehr-Transaktionen zum Einsatz.

Über eine HTTP-Protokoll basierte *JSONRPC*-Schnittstelle können Anwendungen (beispielsweise Wallets) mit dem Framework synchron kommunizieren, um Inhalte abzufragen oder neue Transaktionen einzustellen.

4.2.3. Komponenten am Beispiel Tendermint

Tendermint Core ist ein reines Blockchain-Framework zum Erstellen von Anwendungen, die einen Konsens über einen verteilten replizierten Zustandsautomaten herstellen. Um eine Entkopplung von einer bestimmten Anwendungslogik zu erreichen, wird ein Basisframework mit den darin enthaltenen nötigen Blockchainkomponenten mit einer separaten Anwendung über eine spezifizierte Schnittstelle verbunden. Abbildung 4.3 zeigt die vereinfachte Architektur des Tendermint Core Frameworks. Tendermint Core verwendet das *Reactor*-Entwurfsmuster ([Schmidt 1995]) zur Entkopplung der Komponenten und einen EventBus zur Kommunikation zwischen diesen, um einen service-orientierten Charakter innerhalb des Frameworks zu erzeugen.

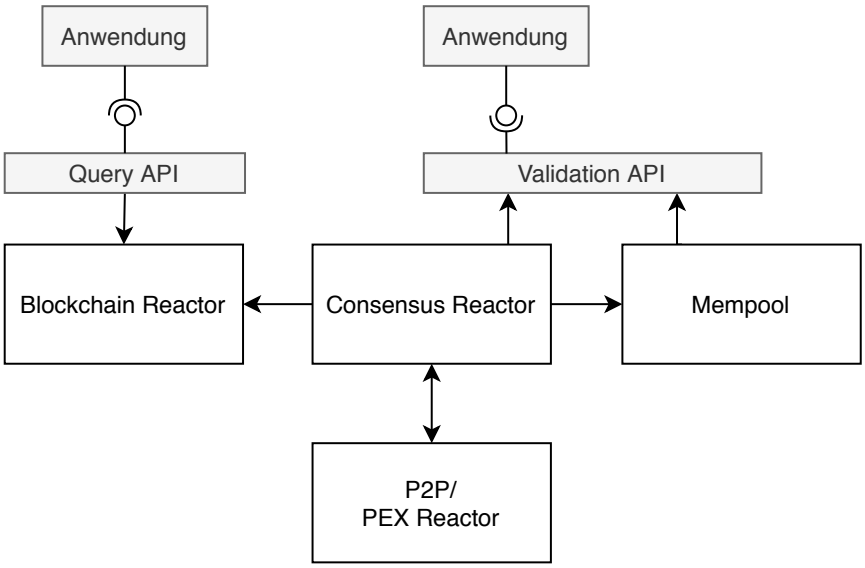

Abbildung 4.3.: Tendermint Architektur

Im Gegensatz zu Bitcoin werden hier Validierungsaufgaben nicht von einer Komponente innerhalb des Frameworks übernommen. Über eine spezifizierte API werden sowohl vom Mempool als auch vom Consensus Reactor periodisch Anfragen an die Anwendung gestellt, ob eine Transaktion gültig ist. Die übrigen Prozesse innerhalb der Blockchain werden ohne weiteres Zutun der Anwendung vollzogen. Dies zeigt sich auch dadurch, dass Tendermint nur blockchainrelevante Rohdaten und Netzwerkinformationen zur Peer-Discovery selbst persistiert. Eine Chainstate-Datenbank oder ähnliches ist nicht integriert und muss von der jeweiligen Anwendung selbst angelegt und verwaltet werden, soweit es erforderlich ist. Da Tendermint einen BFT-PoS Algorithmus verwendet, muss hingegen keine Rückabwicklung von Transaktionen vorgenommen werden, da es nicht zur Entstehung von Forks kommen kann.

Bei Tendermint wurden die Schnittstellen zum Blockchain-Framework in zwei separate APIs nach deren Zuständigkeit aufgeteilt. Über die Query-API kann mit dem Framework direkt interagiert werden, um beispielsweise neue Transaktionen einzustellen und bisherige Blöcke abzufragen. Über die Valdiations-API interagiert das Framework mit der Anwendung, um Transaktionen validieren zu lassen. Ungewöhnlich ist hier, dass die Schnittstellen mit zwei unterschiedlichen Methodiken angesprochen werden. In der Validations-API wird mittels *Protobuf* (von Protocol Buffers) auf eine datensparsame Serialisierung gesetzt, die aber durch eine Veränderung des Nachrichtenformats nicht mehr direkt zum Protobuf-Standard kompatibel ist, jedoch programmiersprachenunabhängig bleibt. Genau genommen, wird die Validations-API von der Anwendung bereitgestellt, sie ist aber für die Funktionalität des Frameworks essentiell.

Die Query-API hingegen setzt auf Web-Standards. Hier werden *JSONRPC* codierte Nachrichten verwendet, die mittels HTTP- oder WebSocket-Protokoll versendet werden. Die Query-API funktioniert nach einem asynchronen Kommunikationsschema, während die Validations-API ausschließlich synchron verwendet wird.

4.3. Transaktions-Phasen

Wenn ein Klient eine Transaktion erstellt und an einen Blockchain-Knoten sendet, durchläuft diese mehrere Phasen in ihrem Lebenszyklus, bevor sie persistiert und als final gültig akzeptiert wird, also auch eine Zustandsänderung in der Blockchain auslöst. Grundsätzlich lassen sich diese Phasen anhand des Validierungsprozesses der Transaktion wie folgt einteilen:

Bevorstehend Eine Transaktion wurde erstellt und befindet sich in Verteilung im Netzwerk.

In Validierung Eine Transaktion wurde empfangen und wird auf Gültigkeit geprüft.

Validiert Eine Transaktion wird aktuell als gültig befunden und wartet auf Finalisierung durch Inklusion in einem Block, oder sie ist ungültig und wird verworfen.

Persistiert Eine Transaktion ist in einem Block persistiert worden und somit final gültig.

Je nachdem, welcher Konsensalgorithmus vom Blockchain-Framework verwendet wird, wird die Reihenfolge der Transaktionen in einer Sortierphase festgelegt, dabei werden außerdem einige der Validierungsphasen der Transaktion durchschritten. Hierbei kann unterschieden werden zwischen Systemen, die wie in klassischer Zustandsautomatenreplikation, erst sortieren und dann ausführen (Order-Execute) oder solchen, die davon in irgendeiner Form abweichen. Hyperledger Fabric verwendet beispielsweise, abweichend vom klassischen Order-Execute, ein Execute-Order-Validate Schema, um den Transaktionsdurchsatz zu erhöhen [Androulaki u. a. 2018].

4.3.1. Transaktions-Phasen bei Bitcoin

In Bitcoin wird das aus der Zustandsautomatenreplikation bekannte Order-Execute Verfahren angewendet, wie in Abbildung 4.4 dargestellt. Das Sortieren findet hier durch die Miner statt. Diese suchen sich Transaktionen aus dem Mempool, die in Bezug auf die aktuelle Historie gültig sind und sortieren sie in einer beliebigen Reihenfolge in den Block ein. Hierbei muss bedacht werden, dass je nach Reihenfolge der Transaktionen nachfolgende Transaktionen invalidiert werden können, also ein Double-Spend bedeuten würden. Diese Transaktionen können folglich nicht inkludiert werden und müssen verworfen werden. Anschließend führen sie die Hashwertberechnung und Nonce-Findung durch und propagieren den Block. Andere Teilnehmer erhalten diesen Blockkandidaten, prüfen ihn auf Gültigkeit und wenden alle Zustandsänderungen der Transaktionen auf den aktuellen Zustand an. Alle Transaktionen in Bitcoin sind deterministischer Natur, was bedeutet, dass bei korrekter Reihenfolge immer derselbe Endzustand erreicht wird und das Netzwerk nicht im Zustand divergiert (mit der Ausnahme von temporären Forks).

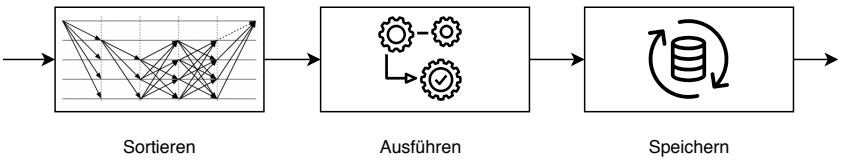

Abbildung 4.4.: Order-Execute-Modell

4.3.2. Transaktions-Phasen bei Hyperledger Fabric

Hyperledger Fabric (kurz Fabric) verwendet im Gegensatz zu früheren Blockchain-Entwicklungen aus Performanzgründen einen anderen Ansatz, wie in Abbildung 4.5 dargestellt. Da in Fabric multiple Anwendungen innerhalb einer Blockchain-Instanz laufen, sind Transaktionen nach Anwendung getrennt, aber in denselben Blockstrukturen zusammengefasst. Knoten werden

in Fabric nach ihren implementierten Anwendungen in eigene Gruppen gefasst, was für separate Validierungs- und Ausführungsgruppen sorgt. Die *Endorsing Policy* legt fest, welche, wieviele und nach welchem Anteil die *Endorsing Peers* (vergleichbar mit Validatoren in BFT-PoS) für den Konsens verantwortlich sind. Diese Policy kann je nach Applikation unterschiedlich ausfallen und legt auch eigene Mehrheitswerte fest, beispielsweise *„1 von 4“* Peers oder Mengenbeziehungen wie $(A \wedge B) \vee C$. Da Fabric für den Permissioned-Einsatz gedacht ist und die Teilnehmer bekannt und kontrolliert sind, kann in vielen Fällen auf $+\frac{2}{3}$ Mehrheiten und somit BFT-Toleranz verzichtet werden [Androulaki u. a. 2018].

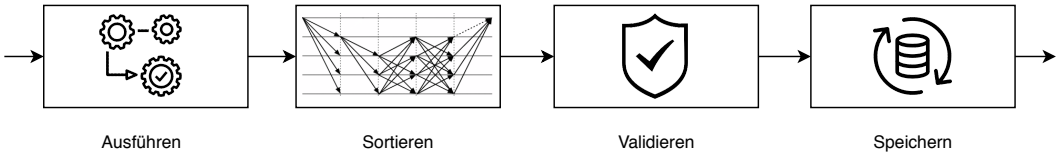

Abbildung 4.5.: Execute-Order-Validate-Modell

In der vorangestellten Ausführungs-Phase werden Transaktionen (hier Proposals) nach Anwendung sortiert an die jeweiligen Knoten ausgeliefert. Diese müssen eine Ausführung simulieren und dabei feststellen, ob diese Transaktion gültig ist und welche Auswirkung dies auf den Zustand hat. Sobald eine Transaktion als gültig erkannt wurde, wird sie vom jeweiligen Knoten „befürwortet“ (endorse).

Alle Befürwortungen werden nun vom Klient gebündelt und als Transaktion and den Ordering Service gesendet. Der *Ordering Service* überprüft die Endorsement-Policy und sortiert darauf die Transaktionen nach ihren Outputs und den erfolgten Zustandsänderungen.

Der Ordering Service stellt in Fabric eine Gruppe von Knoten dar, die für den Konsens verantwortlich ist, allerdings keinerlei semantische Inhalte der Transaktionen kennt und zusätzlich disjunkt vom Rest der Knoten ist. Weiterhin wird keinerlei Validierung von Transaktionen durchgeführt, es wird also lediglich eine Reihenfolge von Transaktionen festgelegt.

Das Validieren der Transaktionen in Bezug auf die abgestimmte Reihenfolge findet hier erst nach der Bestimmung der Reihenfolge statt. Da der Ordering Service kein semantisches Wissen über die Transaktionen besitzt, kann dieser auch nicht feststellen ob die abgestimmte Reihenfolge für ungültige Transaktionssequenzen sorgt. Transaktionen die als ungültig festgestellt werden, werden von den jeweiligen Knoten nicht behandelt, allerdings sind sie trotzdem Teil des Blockes, anders als beispielsweise in Bitcoin, wo nur gültige Transaktionen in Blöcken inkludiert sind. Jedoch ermöglicht dies bei nachträglichen Prüfungen herauszufinden, welche Teilnehmer sich potentiell fehl verhalten haben.

4.3.3. Transaktions-Phasen bei Tendermint

Tendermint ist ebenso wie Fabric ein Framework, was einen verteilten replizierten Zustandsautomaten mit Konsensalgorithmus ohne direkte Anwendung anbietet. Tendermint verwendet auch wie Bitcoin das Order-Execute-Modell, es kann aber ebenfalls dazu verwendet werden, um wie in Fabric, ein Execute-Order-Validate zu implementieren.

Generell werden in Tendermint vom Framework aus in einer bestimmten Reihenfolge verschiedene Schnittstellen aufgerufen. Dazu gibt es zwei verschiedene Socket-Verbindungen, die einerseits für das Überprüfen der Transaktionen im Mempool und andererseits für den Konsensablauf zuständig sind. Abbildung 4.6 zeigt die beiden Schnittstellen und deren wichtigsten Funktionen.

Über die Mempool-Verbindung wird fortlaufend `CheckTx` ausgeführt. Für jede Transaktion, die neu erstellt wird, wird dies separat ausgeführt. Zusätzlich wird dies nach jeder Runde des Konsensalgorithmus erneut für die im Mempool verbliebenen Transaktionen wiederholt, da sich der Gesamtzustand des Systems nun verändert hat und die entsprechenden Transaktionen eventuell dadurch invalidiert werden. Fällt die Validierung einer Transakti-

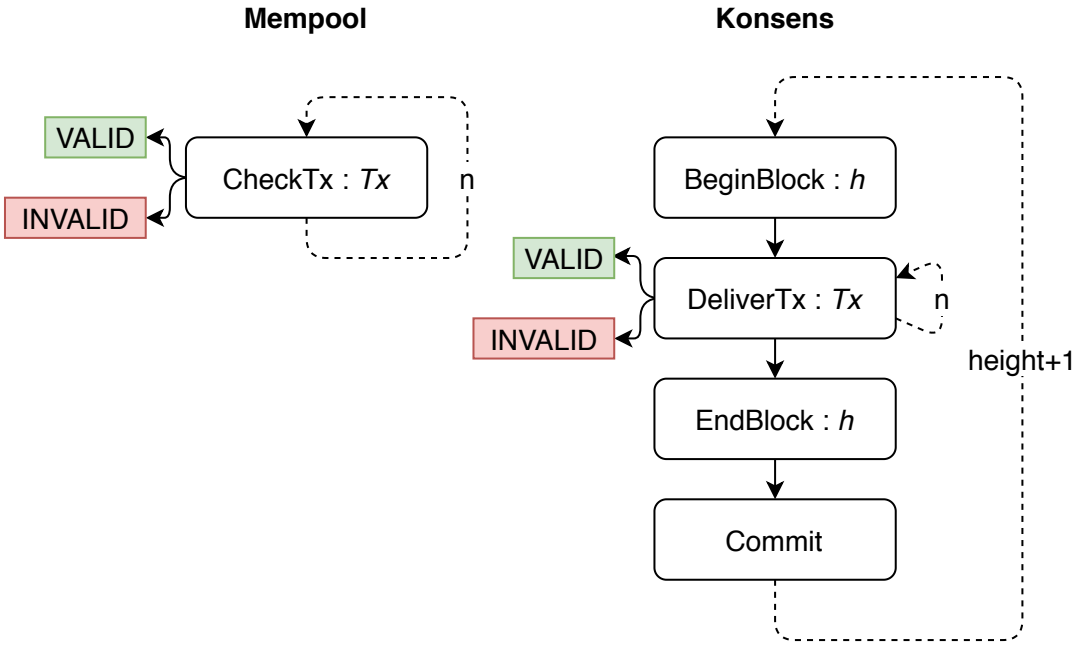

Abbildung 4.6.: Tendermint Mempool- und Konsens-Schnittstelle

on positiv aus, verbleibt diese im Mempool, fällt sie negativ aus, wird sie aus dem Mempool gelöscht.

Über die Konsens-Schnittstelle werden periodisch dieselben vier Funktionen in derselben Reihenfolge aufgerufen: Mittels `BeginBlock` wird der Anwendung signalisiert, dass ein neuer Block beginnt und gleichzeitig werden auch dessen Meta-Informationen übergeben. Darauf folgend wird für jede Transaktion im Block `DeliverTx` aufgerufen. Wird eine Transaktion hier als ungültig markiert, verbleibt diese jedoch im Block und wird lediglich als solche markiert. Mit `EndBlock` wird signalisiert, dass die Übermittlung von Transaktionen abgeschlossen ist und durch die Mitteilung des `Commits` wird signalisiert, dass die Anwendung den soeben übermittelten Block auf den lokalen Zustand anwenden soll. Anschließend inkrementiert das Framework die Blockhöhe um eins und wiederholt den Prozess von vorne.

Je nachdem, wie die Validierungslogik in `CheckTx` in der Anwendung implementiert wird, kann gesteuert werden, ob nur gültige Transaktionen in

Blöcken inkludiert werden oder eben alle Transaktionen. Sollten alle Transaktionen enthalten sein, sind die ungültigen als solche markiert. Worauf die Anwendung, jedoch keinen Einfluss hat, ist die Reihenfolge der Transaktionen in den Blöcken. Die Reihenfolge der Transaktionen wird vom Framework nach dem *FIFO*-Prinzip festgelegt und kann sich, basierend auf dem aktuellen Proposer, von der potentiellen Reihenfolge anderer Knoten unterscheiden. Hierdurch kann es durchaus passieren, dass zwei Transaktionen TX_A und TX_B die unabhängig voneinander als potentiell gültig erachtet werden, aber nur in der Reihenfolge $[TX_A, TX_B]$ gültig sind, vom Proposer als $[TX_B, TX_A]$ eingepflegt werden, wodurch TX_A seine Gültigkeit verliert und als ungültig markiert im Block verbleibt.

Zusammenfassung

Die Blockchain-Technologie steht, wie viele andere Technologien auch vor der Herausforderung, dass sie einerseits skalierbar sein muss, aber weiterhin für Sicherheit und maximale Dezentralisierung sorgen soll. Das Blockchain-Trilemma besagt nun, dass eine Verbesserung einer bevorzugten Eigenschaft – Sicherheit, Skalierbarkeit oder Dezentralität – zur Verschlechterung einer der anderen führt, was viele der aktuellen Technologien vor ungelöste Herausforderungen stellt.

Aus softwaretechnischer Sicht besteht ein großer Teil der bereits existierenden Blockchain-Technologien aus den selben vier Grundkomponenten. Dazu zählen die Persistenz-, Kommunikations- und Konsenskomponente sowie der MemPool, der für die flüchtige Speicherung der Transaktionen zuständig ist. Auf Architekturebene betrachtet, sind nicht alle Technologien gleich strukturiert. In Blockchain-Frameworks, die nur dem einzigen Zweck des Werttransfers dienen (z.B. Bitcoin), sind die einzelnen Komponenten enger miteinander verknüpft, während sie in anderen Technologien deutlich besser voneinander abgekapselt sind. Eine lose Kopplung erlaubt hier, potentiell ein flexibleres Austauschen von Komponenten für andere Anwendungszwecke

und muss deshalb für ein generalisiertes Blockchain-Framework bevorzugt werden.

Weiterhin werden auch die Transaktionen innerhalb der Frameworks unterschiedlich behandelt. Über Transaktionen werden atomare Veränderungen am gesamten globalen Systemzustand vorgenommen. Deshalb müssen sie dementsprechend mit größter Sorgfalt auf Validität und Konformität bezüglich der Geschäftsprozesse geprüft werden. Auch die Reihenfolge der Transaktionen innerhalb der Blöcke spielt für Geschäftsprozesse eine Rolle, jedoch nicht für die existierenden Frameworks. Diese ordnen Transaktionen meist nach der Reihenfolge, wie sie im MemPool eingetroffen sind (FIFO-Prinzip) und nicht nach der Relevanz innerhalb des Prozesses.

Im nachfolgenden Kapitel werden deshalb, um die genauen Anforderungen für Geschäftsprozesse und deren notwendige Anpassungen an Blockchain-basierte Prozesse aufzuzeigen, die generellen Herausforderungen bei der Umsetzung betrachtet und diese weitergehend an zwei konkreten Beispielszenarien vorgestellt.

5. Anforderungsanalyse verteilter B2B-Prozesse und Eignung für eine Unterstützung durch Blockchain-Technologie

Im Anschluss an die Vorstellung der relevanten Grundlagen und einiger bestehender Technologien wird im folgenden Kapitel eine Anforderungsanalyse bezüglich der Integration von Blockchains in Business-zu-Business-Anwendungen (B2B) anhand zweier Beispiele aus dem Energiehandel und dem Versicherungsmarkt vorgestellt. Beide bieten jeweils ein hohes Potential für eine softwareunterstützte Dezentralisierung auf der Basis der Blockchain-Technologie. Zuerst erfolgt eine generelle Beschreibung der speziellen Herausforderungen und Anforderungen für Geschäftsprozesse mit verteilten Akteuren in dezentralen Systemen. Anschließend werden die beiden Szenarien und deren aktuelle Prozesse vorgestellt sowie deren Eigenheiten und daraus resultierende Anforderungen und Herausforderungen bei einer Blockchain-basierten Umsetzung. Daraus wird dann ein Konzept mit den jeweiligen Anpassungen abgeleitet, die sich durch eine solche Umsetzung notwendigerweise ergeben.

5.1. Herausforderungen für Blockchain-basierte B2B-Prozesse

In heutigen Business-zu-Business Geschäftsprozessen finden sich eine Vielzahl von Akteuren. Einige von diesen sind aus regulatorischen Gründen zwischengeschaltet, um beispielsweise die korrekte Lieferung eines Produktes zu überwachen oder Marktmanipulationen aufzudecken. Andere Akteure sind aufgrund von Vertrauensproblemen unter den Teilnehmern notgedrungen ein Teil des Gesamtprozesses. Aber auch in zentralisierten Prozessen mit einer zentralen Plattform als Vermittler gibt es den Plattformbetreiber selbst als zusätzlichen Dritten.

Der Hauptgrund, von einem klassischen zentralen System auf ein Blockchain-basiertes System mit dezentralen Prozessen zu wechseln, ist die Disintermediation. Durch den Wegfall Dritter lässt sich faktisch Geld sparen, wie etwa Vermittlungsgebühren. Prozesse werden schneller und schlanker, wenn sie nur noch zwischen den eigentlichen Parteien ausgeführt werden.

Charakteristisch für zentrale Prozesse sind Dienstleister oder Broker, die zwischen den Endkunden die Zahlungsflüsse verwalten und für den Ausgleich von ausstehenden Zahlungen sorgen. Da diese Handels- und Zahlungsinformationen natürlich streng vertraulich sind, werden sie nur zwischen den Handelspartnern und den in den Prozess einbezogenen Intermediären ausgetauscht. Externe Parteien, obwohl sie Teilnehmer des Markplatzes sind, werden nicht über die Details, wie Preise und Volumina, oder überhaupt über das Zustandekommen eines Deals informiert.

Die erste Herausforderung bei einer Prozessumsetzung liegt hier in der grundlegenden Idee einer Blockchain. Alle Transaktionen sind für alle Teilnehmer einsehbar, denn sie müssen von allen Teilnehmern validierbar sein. Nur deterministisch validierbare Transaktionen können durch einen Konsensalgorithmus später zu einem gemeinsamen, gültigen Systemzustand führen. Um sowohl die Validierbarkeit der Transaktionen als auch die Wahrung von Ge-

schäftsgeheimnissen zu gewährleisten, müssen die Transaktionsinhalte also speziell auf die jeweiligen Prozesse angepasst werden. Einerseits müssen Teile des Prozesses identifiziert werden, die allgemein verfügbar sind, andererseits müssen private Inhalte vor Unbeteiligten geschützt werden.

Mit der offenen Verfügbarkeit der Blockchain kommt ein weiteres Problem einher. Viele Prozesse oder Geschäfte sind nur auf bestimmte Gruppen zugelassen, beispielsweise benötigen Banken eine Lizenz. Für ein Blockchain-System heißt dies also, dass der Zugang zum System beschränkt werden muss und folglich auch die Möglichkeit der Teilnahme. Dazu gibt es, wie in Kapitel 3.5 beschrieben, mehrere mögliche Zugriffs- und Netzwerkformen für Blockchains. Ein Konsortialzusammenschluss zwischen den Beteiligten eines Prozess kann sich beispielsweise für ein privates Netzwerk (private-permissioned) entscheiden, um Unbeteiligte komplett aus dem Prozess herauszuhalten. Für eine unabhängige Kontrolle eignen sich aber auch public-permissioned Netzwerke. Hier können nur akkreditierte Teilnehmer den Prozess beeinflussen, aber die generelle Öffentlichkeit kann jederzeit den Prozess überwachen. Da dies für jeden Anwendungsfall unterschiedlich sein kann, muss auch für jeden separat eine Netzwerkform anhand der Anforderungen gewählt werden.

5.2. Prozesse im Energiehandel

Mit der Liberalisierung des Energiehandels 1996 in Europa und der entsprechenden Umsetzung in Deutschland im Jahr 1998 wurden Marktstrukturen geschaffen, wie man sie heutzutage kennt. Viele verschiedene Akteure mit unterschiedlichen Funktionen sind an dem Handel mit Energie (Strom und Gas) beteiligt.

Das deutsche Stromnetz lässt sich in drei Bereiche einteilen, wie in Abbildung 5.1 beispielhaft dargestellt. Auf der höchsten Ebene befinden sich die vier Übertragungsnetzbetreiber (*TSO, Transmission System Operator*). Darun-

ter befinden sich lokale Anbieter, die Endkunden mit Strom versorgen, die Verteilnetzbetreiber (*DSO, Distribution System Operator*). Auf der untersten Ebene befinden sich die Endkunden mit weiteren kleineren lokalen Stromanbietern in sogenannten Microgrids. Sowohl auf den Ebenen der TSOs und DSOs als auch auf Endkundenebene befinden sich heutzutage Energieerzeuger und -verbraucher, was sich sowohl beim Handel mit Energie bemerkbar macht als auch bei den Maßnahmen zum Ausgleich der Netzbelastung und -spannung.

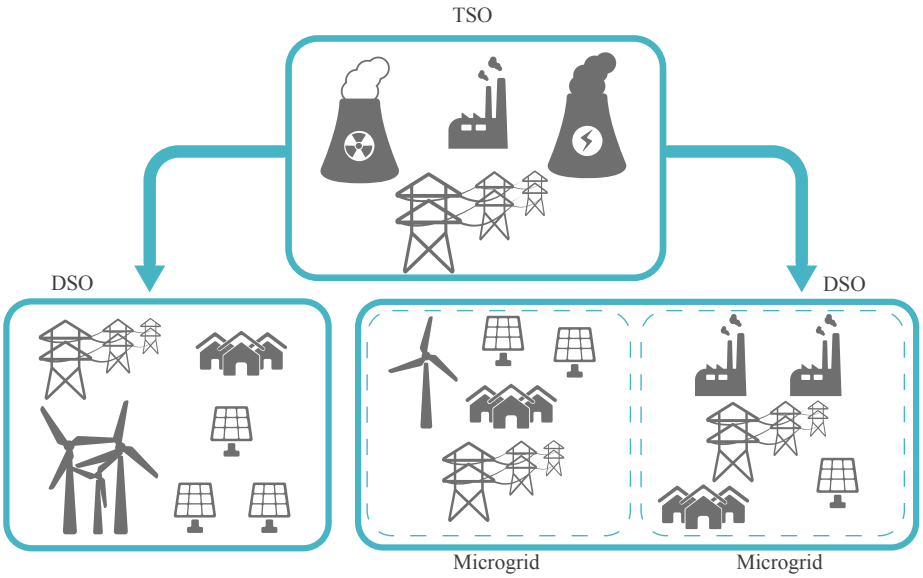

Abbildung 5.1.: Netzstruktur

5.2.1. Akteure und Prozesse im Energiehandel

Neben den aus dem privaten Bereich bekannten lokalen Anbietern für Energie gibt es darüber hinaus eine Vielzahl an weiteren Akteuren mit unterschiedlichen Rollen im Energiehandel, die für Endkunden meist unbekannt sind. Nachfolgend werden die wichtigsten Akteure und ihre Funktion für den börslichen und den außerbörslichen Handel (OTC, Over-The-Counter) vorgestellt [Merz 2019].

Börse Energiebörsen in Europa werden für den nationalen und grenzübergreifenden Handel mit EU-Auslandspartnern verwendet, um sowohl die Angebote als auch die Nachfrage für Strom und Gas abzuwickeln. Für den Raum Deutschland, Frankreich und Österreich ist die *EEX* zuständig. Die Börsen unterliegen dabei immer Auflagen durch Regulatoren (national und international). Auch „Fahrplan"- und Lastvoranmeldungen werden teilweise durch Börsen abgebildet.

Broker Broker werden im Energiehandel verwendet, um OTC-Geschäfte außerbörslich als Vermittler abzuwickeln. Teilweise bieten diese auch eigene Handelsplattformen für den bilateralen Handel. Sowohl Börsen als auch Broker tragen maßgeblich zur Preisbildung für Strom und Gas bei und dienen als Indikator.

Clearinghaus Clearinghäuser oder Clearingstellen werden in vielen Prozessen für die Saldierung und für den Ausgleich von Forderungen zwischen Vertragsparteien eingesetzt. Im Energiehandel führen sie die finanzielle und auch physische Abwicklung von Handelsgeschäften durch. Im Falle einer ausbleibenden Lieferung übernimmt das Clearinghaus diese.

Erzeuger Erzeuger sind alle Marktteilnehmer, die aktiv Energie ins Netz einspeisen. Früher waren dies ausschließlich große Teilnehmer wie Windparks, Kohle und Atomkraftwerke. Heutzutage kommen durch kostengünstige Photovoltaikanlagen in Eigenheimen diverse private Erzeuger hinzu.

Händler Händler kaufen und verkaufen Energie beim europäischen Großhandel. Dabei kann es durchaus sein, dass ein Produkt mehrere Male weiterverkauft wird, bis es abschließend beim Verbraucher ankommt.

Lieferanten Lieferanten sind die Schnittstelle der Verbraucher zum Energiemarkt. Mit Novellierung der Gesetze im Jahr 1998 ist es Verbrauchern erstmalig gestattet, ihre Lieferanten selbst auszuwählen. Diese kaufen

den Strom im Großhandel ein und veräußern ihn weiter an die Verbraucher. Der Einfluss der Lieferanten auf den Strompreis ist eher gering.

Regulator Regulatoren überwachen auf nationaler oder europäischer Ebene den Energiemarkt und -handel. Zu den Aufgaben gehören unter anderem die Marktüberwachung, Kontrolle und Genehmigung von Netznutzungsentgelten und die Schaffung von Zugang zu Stromversorgungsnetzen. In Deutschland sind dafür die Bundesnetzagentur bzw. die jeweiligen Landesregulierungsbehörden bei kleinen lokalen Anbietern zuständig.

OTC und Börsenhandel

Für die Betrachtung einer Blockchain-basierten Prozessumsetzung im Energiehandel wird im Folgenden der OTC-Direkthandel dargestellt. Die Handelsbeziehungen erstrecken sich dabei über Händler, Lieferanten und Erzeuger. In OTC Märkten werden langfristige termingebundene Produkte (Terminmarkt) wie die jährliche, quartalsbasierte oder monatliche Grundlast, aber auch kurzfristige Produkte (Spotmarkt) wie Day-Ahead (Folgetag) und Intra-Day (aktueller Tag) gehandelt. Kurzfristige Geschäfte sind dabei in 1-stündige oder auch 15-minütige Lieferslots für kurzfristig notwendige Regelenergie eingeteilt. Die ablaufenden Prozesse im Energiemarkt mit OTC- und börslichem Handel sind vereinfacht in Abbildung 5.2 dargestellt.

Wenn ein OTC-Handelsgeschäft über Broker oder direkt bilateral zwischen Händlern zustande kommt, werden die Details der Transaktionen zwischen den Händlern direkt ausgetauscht. Die Trade Confirmation wird meist händisch in die jeweiligen ETRM Systeme der Händler eingepflegt. Um das Lieferausfall- oder Zahlungsunfähigkeitsrisiko mit einzelnen Händlern zu vermeiden, werden häufig auch für den OTC-Handel Clearinghäuser mit einbezogen, um die Transaktion dort abzuwickeln. Jedoch erfolgt die Initiierung aufgrund von Komplexitätsgründen nicht direkt von den Händlern, sondern von den vermittelnden Brokern. Bis zum Ende des Folgetages muss jeder

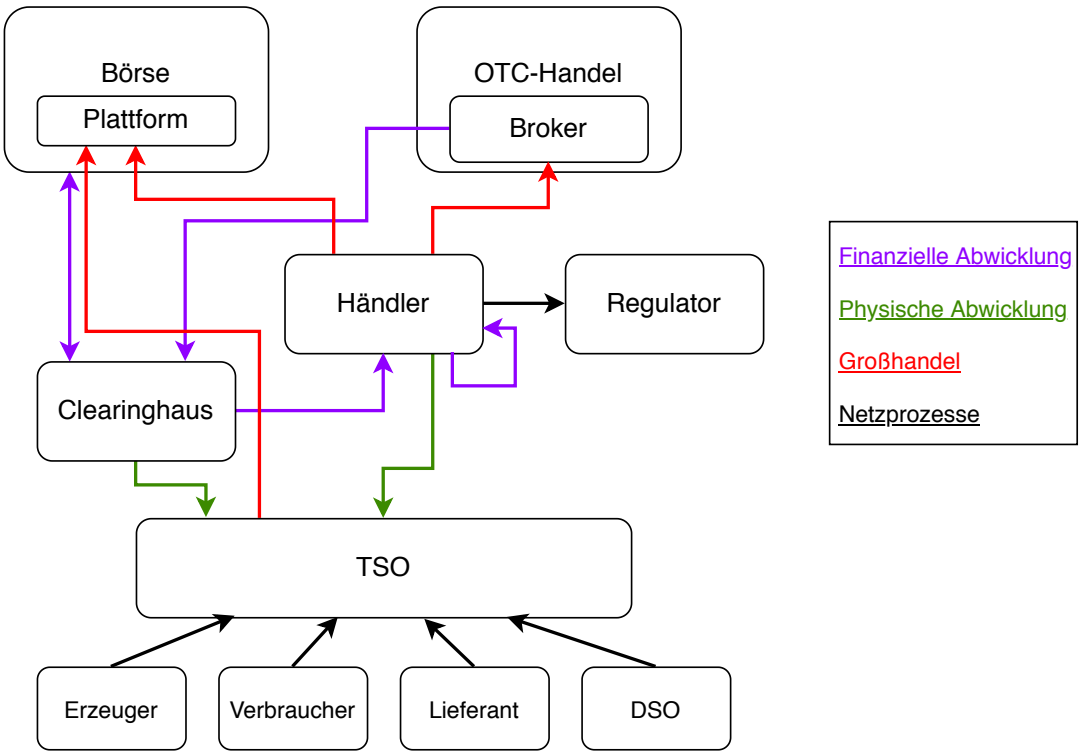

Abbildung 5.2.: Prozesse im Energiemarkt (nach [Merz 2019])

Händler seine Orders, Handelsgeschäfte und zusätzlichen Details der Transaktionen an den zuständigen Regulator melden. Dieser Teil des Prozesses ist der einzig flächendeckend standardisierte Prozess, während Plattformen von Brokern und Börsen jeweils individuelle Prozesse und plattformspezifische Datenformate benutzen.

Der börsliche Handeln unterscheidet sich grundsätzlich nicht vom OTC-Geschäft, jedoch sind Clearinghäuser obligatorisch und deutlich direkter eingebunden. Handelstransaktionen werden hier nicht mehr direkt zwischen zwei Händlern abgewickelt, sondern immer mit dem Clearinghaus als zusätzlicher Schritt in der Mitte. Das führt dazu, dass ein sonst stark vermaschtes Netz von direkten Transaktionen $A \to B$ nun sternförmig über das Clearinghaus in der Form $A \to CH \parallel CH \to B$ abgewickelt wird. Das Clearinghaus agiert somit für eine Partei als Käufer und für die andere als Verkäufer. Das Clearinghaus kann somit auch ein- und ausgehende Transaktionen mit der-

selben Partei miteinander saldieren und die Anzahl der Zahlungstransaktionen verringern.

5.2.2. Problemstellung

In diversen Bereichen mit komplexen Geflechten von Finanztransaktionen lassen sich durch Asset-basierte Blockchains diverse Dienstleister aus der Mitte entfernen. Gerade wenn in nahezu Echtzeit Transaktionen übermittelt werden, gibt es keine Notwendigkeit mehr, diese zentral zu begleichen, da beide Parteien über die Transparenzeigenschaft der Blockchain bereits über alle dafür notwendigen Details verfügen [Wyman 2016].

Im Energiemarkt gibt es mehrere Akteure, die als Intermediäre zwischen den Händlern agieren und dafür bezahlt werden müssen. Sowohl die Clearinghäuser, die für ihre Dienste bezahlt werden müssen, als auch die Plattformbetreiber, seien es Broker oder Börsen, wären in einem dezentralen Ansatz nicht mehr notwendig.

Die Rolle der Clearinghäuser ist zum einen, die vermittelte Sicherheit und zum anderen, die Zahlungsflüsse zu optimieren. Die sternförmige Transaktionsstruktur sorgt dabei für ein Ausgleichen von eingehenden und ausgehenden Zahlungen. Folglich muss für eine Blockchain-basierte Umsetzung dieses Prozesses eine Möglichkeit gefunden werden, wie Zahlungsflüsse und Waren abgebildet werden können.

Innerhalb eines dezentralen Marktplatzes gibt es noch weitere Probleme, die derzeit von Brokern und Börsen erfüllt werden. Zum einen ist die Sichtbarkeit von Handelsgeschäften auf die jeweiligen handelnden Parteien und die Plattform beschränkt. Hierdurch werden sowohl Geschäftsgeheimnisse als auch die konkreten Transaktionen vor Dritten verborgen. Hier spielen auch die sogenannten Kredit-Matrizen eine Rolle. Jeder Händler führt eine Glaubwürdigkeits- oder Kredit-Liste mit Informationen darüber, mit welchem anderen Händler er welche Volumina handeln darf oder kann. Teil-

weise sind die Volumina auch unterschiedlich je nachdem wer Händler und Verkäufer ist: $Kredit_{A \to B} \neq Kredit_{B \to A}$. Diese Kredit-Liste wird den Brokern und Börsen gegeben, um von vornherein einige Transaktionen zu unterbinden, wenn sie das Kreditlimit übersteigen würden. Aufgrund von Geschäftsgeheimnissen sind den Händlern nur ihre eigenen Limite bekannt, nicht aber die der anderen für sie selbst. Hier wird aktuell ein vertrauenswürdiger Dritter im Prozess benötigt.

5.2.3. Anforderungen

Im Folgenden werden die Anforderungen für einen Blockchain-basierten dezentralen Marktplatz für den Energiemarkt aufgezeigt und erläutert. Die allgemeinen Anforderungen wurden aus repräsentativen Anwendungsszenarien abgeleitet und auf dieser Basis wurden die konkreten Anforderungen im Rahmen mehrerer Kooperationsprojekte mit einem Dienstleister der Energie-Branche entwickelt. In unterschiedlichen Projekten mit der Firma *PONTON GmbH*, lag dabei der Fokus auf der Durchführung vieler, kurzfristiger Transaktion innerhalb eines dezentralen Energiemarktes. Nachfolgend wird der Fokus auf den Spotmarkt im OTC-Handel mit Intraday-Geschäften gelegt, der sich maßgeblich aus kurzfristigen Transaktionen zusammensetzt.

Zugang Für den Handel innerhalb einer Blockchain ist eine Zugangsberechtigung erforderlich, um Unbefugte auszuschließen. Nur zum Energiemarkt zugelassene Teilnehmer dürfen berechtigt sein zu handeln, um einerseits Missbrauch und Spam durch ungültige Transaktionen zu vermeiden, anderseits auch, um die Stromnetzstabilität nicht durch ausbleibende Lieferungen zu gefährden.

Identität In Europa sind alle Händler, Lieferanten, TSOs und DSOs und auch Liefergebiete eindeutig identifizierbar. Der *Energy Identification Code* (EIC) wird national und europaweit von den zuständigen Behörden vergeben. Anhand der EIC-Codes kann weiterhin zwischen der Funktion unterschieden werden, so besitzen Liefergebiete (korrekt „Bilan-

zierungsgebiet“) EIC-Y Codes, während Marktteilnehmer durch EIC-X Codes identifiziert werden. Der EIC Code sollte in Transaktionen für eindeutige Identifizierbarkeit von Handelspartnern verwendet werden, gerade auch, um erfolgreiche Geschäftsabwicklungen durchführen zu können.

Sichtbarkeit Abgeschlossene Transaktionen zu Handelsgeschäften gelten mitunter als Geschäftsgeheimnis. Diese müssen vor Konkurrenten geschützt werden, aber trotzdem eine Zuordnung, beispielsweise für Regulatoren, ermöglichen. Auch müssen Transaktionen durch eine Offerte als abgeschlossen betrachtet werden können, wenn ein gültiger Handel zustande gekommen ist, damit nicht weiter mit dieser interagiert werden kann.

Zu den sichtbaren Informationen einer Transaktionen zählen beispielsweise die Liefermenge, der Lieferort, der Lieferzeitpunkt und der Preis pro Megawattstunde. Teilweise ist es wünschenswert, wenn der Verkäufer initial verschleiert wird. Dies ist im Rahmen eines vollständigen offenen Marktes0 aber kontraproduktiv.

Validierung Die Transaktionen, die Angebot und Gebot von Handelsgeschäften darstellen, müssen in geeignetem Maße validierbar sein. Das heißt, dass mindestens auf plausible Parameter wie Lieferzeitraum und -menge geprüft werden muss. Dabei muss beachtet werden, dass auch neue Angebote in der Vergangenheit liegen können, um nachträglich bereits erfolgte Lieferungen abzuwickeln.

Transaktionsgeschwindigkeit Transaktionen müssen nahezu in Echtzeit durch das Netzwerk verteilt werden. Vor allem im kurzfristigen Spotmarkt mit Intraday-Geschäften für den nächsten 15-Minuten-Slot müssen Transaktionen auch kurzfristig verfügbar und abwicklungsfähig gemacht werden.

Blockzeit Für die Abwicklung von kurzzeitigen Handelstransaktionen ist eine geringe Inter-Blockzeit erforderlich. Dies ermöglicht es, schneller Ge-

schäfte abzuwickeln, da Blockchain-Transaktionen nur kurz auf eine Inklusion warten. Weiterhin ist eine sofortige Blockfinalität erforderlich, um Rückabwicklungen oder gegensätzliche Zustände im System durch Forks zu vermeiden.

Anbindung Der Blockchain-basierte Marktplatz muss eine Anbindung an die bestehenden ETRM-Systeme der Händler ermöglichen. Hierbei muss möglichst von Blockchain-spezifischen Inhalten abstrahiert werden und es dürfen jeweils nur Handelsdaten an das System weitergegeben werden.

Regulatoren der spezifischen lokalen Märkte müssen, wenn sie passiv am Marktplatz teilnehmen, auch an das System angebunden werden. Dabei müssen für diese unter Umständen die Sichtbarkeiten angepasst werden, um die Handelspartner eindeutig zu identifizieren.

5.2.4. Dezentrale Koordination des Energiehandels

Der Energiemarkt eignet sich prinzipiell für eine Blockchain-basierte Umsetzung aufgrund seiner Handelsform und Teilnehmerstruktur. Generell sind Prozesse, in denen Werte oder Assets zwischen Teilnehmern transferiert werden, besonders prädestiniert. Auch die große Anzahl von Intermediären macht es wirtschaftlich interessant diesen Prozess zu verschlanken. Letztlich sorgt auch der „1-zu-N“-Charakter von Handelstransaktionen der Energiebörse für eine gute Umsetzbarkeit, denn jeder Teilnehmer stellt seine Ware für alle anderen Teilnehmer offen zur Verfügung, wie es bei Blockchain-Transaktionen ohnehin der Fall ist.

Um die konkrete Eignung für eine dezentrale Koordination des OTC-Handels im Energiemarkt mittels der Blockchain-Technologie festzustellen, werden nachfolgend die aufgestellten Anforderungen und Aspekte des vorher gezeigten Prozesses betrachtet, die für eine Umsetzung und Adaption des Pro-

zesses sprechen. Es wird hier auch darauf eingegangen, wie diese Prozesse eine solche Umsetzung ermöglichen oder gar begünstigen.

Validierung von Angebotstransaktionen

Im aufgezeigten Marktplatz-Szenario kann die Validierung der Angebots-Transaktionen auf die Plausibilität der eingestellten Angebote beschränkt werden. Dies stellt lediglich hinsichtlich der abzugleichenden Stammdaten einen hohen Aufwand bezüglich ihrer Aktualität dar. Nur so kann sichergestellt werden, dass beispielsweise Erzeuger und Lieferanten ausschließlich gültige Angebote für ihre Lieferbereiche erstellen. Die Validierung von Geboten hingegen stützt sich einerseits auch auf die Stammdaten und andererseits auf die bestehende Gültigkeit des Angebotes. Gebote auf Angebote mit zurückliegenden Lieferterminen und bereits zustande gekommenen Geschäften können auch ohne die Zuhilfenahme der Stammdaten validiert werden.

Zugang, Identität und Sichtbarkeit

Wie in bereits bestehenden Börsen und Brokerplattformen kann auch der Zugang zu Blockchain-basierten Systemen beschränkt und für Dritte ohne Autorisierung unzugänglich gemacht werden. Da es sich bei diesem Marktplatz ohnehin nicht anbietet, ihn für die breite Öffentlichkeit zur Verfügung zu stellen, kann eine Autorisierung und Identifikation über die bekannten EIC-Codes jedes Marktteilnehmers erfolgen. Denkbar ist in diesem Falle eine Public-Key-Infrastruktur, die beispielsweise durch die zuständigen Regulatoren betrieben wird. Über Public-Keys, die durch die zuständigen Regulationsbehörden signiert sind, kann so die Autorisierung und die Authentizität der Teilnehmer festgestellt werden. Einerseits könnte das Blockchain-Netzwerk nur Verbindungen nach erfolgter Authentifizierung durch einen solchen signierten Schlüssel erlauben, wie beispielsweise bei SSH mittels *public key authentication* oder *mutual authentication* in SSL Verbindungen bei HTTPS. Andererseits könnten auch weiterhin eingestellte Transaktionen mit die-

sen Schlüsseln signiert werden und solche ohne gültige Signatur generell als invalide markiert und abgelehnt werden.

Für die Verwaltung, Verteilung und Signaturanfragen kann ebenfalls die Blockchain-Technologie selbst verwendet werden. So wird innerhalb desselben Mediums sichergestellt, dass alle Teilnehmer denselben Zustand aller Schlüssel besitzen und es keine Komplikationen durch beispielsweise veraltete Schlüssel gibt.

Ein gemeinsam geteilter und verwalteter Schlüsselbund ermöglicht zusätzlich weitere Möglichkeiten hinsichtlich der erforderlichen Beschränkung der Sichtbarkeit von Transaktionen. Wenn ein Händler ein Angebot regional beschränken will, beispielsweise auf Deutschland, kann er anhand der EIC-X Codes der relevanten Marktteilnehmer die Schlüssel heraussuchen und nun Teile der Transaktion nur für diese verschlüsseln. So werden Teilnehmer die nicht zum Handel berechtigt sind, davon ausgeschlossen, indem sie keinerlei Details über die Angebote erfahren.

5.2.5. Prozessadaption für eine Blockchain-basierte Umsetzung

Die Adaption des Prozesses für einen Blockchain-basierten dezentralen Marktplatz bedarf vergleichsweise weniger Eingriffe in den tatsächlichen Prozess. Wie in Börsen und Brokerplattformen gibt es Transaktionen, die Angebote einstellen, und Transaktionen, die Gebote für diese repräsentieren.

Dadurch, dass ein zentraler Betreiber wegfällt, der für eine sofortige Transaktionsfinalität sorgen würde, ergeben sich in einer Blockchain-Variante einige Verzögerungen durch den gewählten Konsensalgorithmus selbst und die Nachrichtenlaufzeiten, die für den Blockbildungsprozess, die Blockzeit und die Blockfinalität relevant sind. Abbildung 5.3 zeigt beispielhaft den Ablauf von Erstellung des Angebots, Abgabe eines Gebotes mit Zustandekommen des Vertrages und dem Reporting an einen Regulator. Ein weiterer Händler

(rot) erfährt durch das abgegebene und im Block inkludierte Gebot, dass das eingestellte Angebot bereits abgelaufen und nicht mehr handelbar ist.

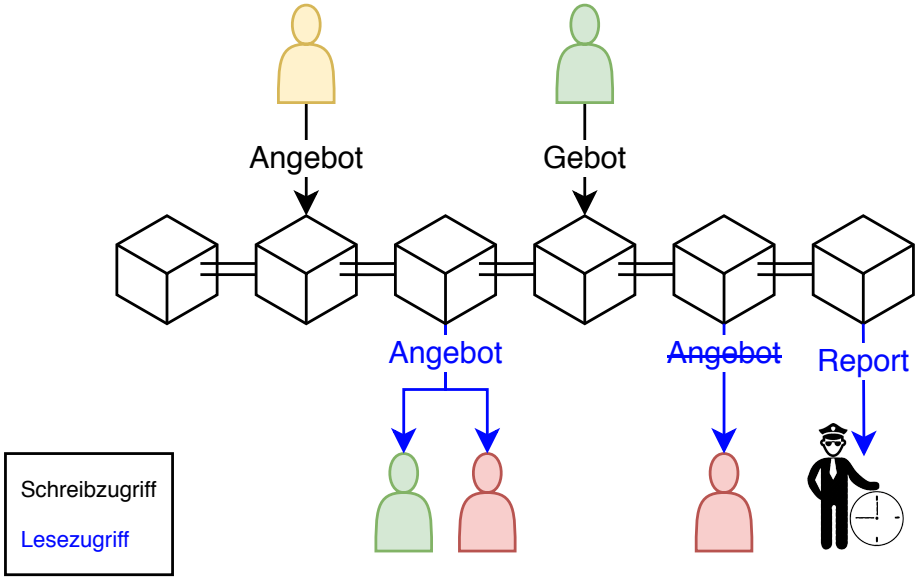

Abbildung 5.3.: Blockchain-basierter Handelsprozess

Während im klassischen Großhandel von Energie (siehe Abbildung 5.2) das Reporting der Händler direkt an die zuständigen Regulatoren erfolgt, ermöglicht es der Einsatz einer Blockchain, auf ein aktives Reporting zu verzichten. Dies kann nun passiv durch die Blockchain erfolgen, erfordert hingegen, dass die zuständigen Regulatoren auch an das System angeschlossen werden. Angeschlossene Regulatoren bekommen, wie andere Teilnehmer auch, jede Transaktion übermittelt, die innerhalb eines Blockes persistiert ist. Wenn diese Transaktion als Liefergebiet mit dem EIC-Y Code für beispielsweise Deutschland `10YDE-...` beginnt, kann der Regulator diese mit in seinen Report aufnehmen.

Die Struktur des dezentralen Marktes mit integrierter Reporting-Funktion macht es umso leichter auf externe Dritte wie Broker zu verzichten. Der Wegfall von Clearinghäusern, obwohl nicht mehr zwingend nötig, steht vor anderen Hürden. Denn Clearinghäuser fungieren häufig auch als Bürgen, um Kre-

ditrisiken zu minimieren und ausbleibende Lieferungen im Netzwerk auszugleichen.

Vor- und Nachteile einer Umsetzung dezentraler Koordination mittels Blockchain-Technologie

Abschließend werden nun noch auf die resultierenden Vor- und Nachteile eines dezentralen Prozesses bei der Koordination verteilter Anwendungen eingegangen. Vor allem der potentielle Wegfall von vielen Dritten und Intermediären begünstigt eine Blockchain-Umsetzung. Diese lassen sich ihre Vermittlungs- und Abwicklungstätigkeiten natürlich honorieren, sodass durch einen dezentralen Markt eine monetäre Kostenreduktion eintritt. Auch die „Zentralisierung“ auf einen einzigen Blockchain-Marktplatz, auf dem alle Teilnehmer gleichberechtigt sind und befreit von Transaktionskosten teilnehmen können, spricht aus dieser Sicht für eine derartige Umsetzung.

Ein kritischer Punkt des Systems sind hingegen die Nachrichtenlaufzeiten und die daraus resultierenden möglichen fachlichen Prozessverzögerungen. Da ein dezentraler Marktplatz und Blockchains im Allgemeinen, anders als eine zentrale Börse, keine sternförmige Netzstruktur aufweist, sondern vielmehr ein zufällig vermaschtes Netz darstellen, ergeben sich für Nachrichten andere und längere Laufzeiten von einem Teilnehmer zum anderen. Abbildung 5.4 zeigt in einem System mit fünf Knoten (hier in grau) wie eine Nachricht ausgehend vom gelben Händler im Netz verteilt wird. Während die Nachricht bei den meisten Knoten innerhalb von zwei Übermittlungen ankommt, sieht sie der grüne Teilnehmer erst nach der vierten Übertragung. Dies könnte unter Umständen zu einer Benachteiligung beim Handel führen, wenn ein Händler von einem Angebot erst später erfährt als ein anderer, vor allem in Systemen, wo eine KI automatisiert und eigenständig handelt. Ein PoW-Algorithmus eignet sich hier also unter keinen Umständen für einen Marktplatz, da ein einzelner Knoten den kompletten Block erstellt und ihn progressiv durchs Netzwerk propagiert.

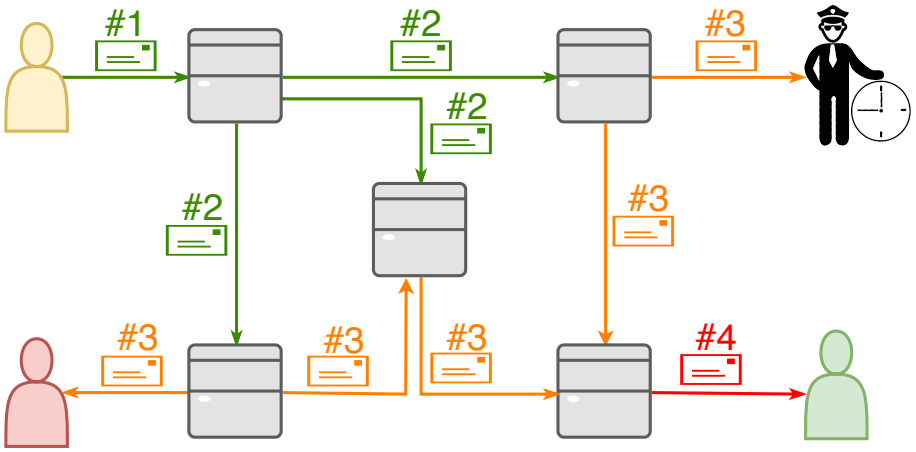

Abbildung 5.4.: Nachrichtenverteilung in einem zufällig vermaschten Netz

Zusätzliche rechtliche Betrachtung

Auch was rechtliche Fragen anbelangt ist die Blockchain-Technologie und ihre Möglichkeiten noch nicht in allen Bereichen angekommen, wodurch sich häufig Unklarheit über ihre tatsächliche Einsetzbarkeit für einen bestimmten Prozess ergibt. Dies ist auch bei einem dezentralen Marktplatz für den Energiehandel der Fall.

Während Konsumenten, Produzenten und Prosumenten in Smart Grids aus rechtlichen Gründen generell nicht direkt miteinander Strom handeln dürfen, sowohl zentral als auch dezentral, ist ein dezentraler OTC-Handel von einem anderen juristischen Problem betroffen: die zuständige EU-Verordnung je nach Marktplatzform.

Wenn dieser Blockchain-basierte Marktplatz ein Markplatz im Sinne von Finanzprodukten mit Sicherheiten, Derivaten und ähnlichem wäre, würde er unter die „MIFID“-Richtlinie der EU für Märkte und Finanzinstrumente [Europäisches Parlament und Rat 2014] fallen. Stellt dieser Marktplatz hingegen nur eine Plattform zur Kommunikation und die Übermittlung von Nachrichten bereit, mittels dieser er einen Handel von physischen Produkten am Energiemarkt abbildet, würde er unter die „REMIT“-Richtlinien der EU fallen, die

den Energiegroßhandelsmarkt regulieren [Europäisches Parlament und Rat 2011].

Aufgrund fehlender rechtlicher Expertise in diesem Gebiet wird auf eine Einschätzung oder Bewertung der effektiv resultierenden Marktform an dieser Stelle verzichtet.

5.3. Prozesse bei Mitversicherungen

Das vorige Beispiel zeigte vor allem die Möglichkeit einer Umsetzung von Prozessen, in denen unbeschränkte 1-zu-N Beziehungen der Waren zwischen den Teilnehmern vorliegen. Prozesse, die Börsen- oder Marktplatzcharakter haben eignen sich durch die Funktion eines Nachrichtenbrettes der Blockchain sehr gut für solche Umsetzungen. Im nachfolgenden Beispiel soll gezeigt werden, dass sich auch andere Anwendungsfälle, die keinen Marktplatz abbilden, für eine dezentrale Umsetzung eignen.

Im Versicherungsgeschäft wird ein Risiko, welches für eine Einzelperson bzw. für ein einzelnes Versicherungsunternehmen zu hoch ist, auf ein Kollektiv aufgeteilt. Üblicherweise zahlen dabei Versicherungsnehmer eine Prämie an einen Versicherer und erhalten so Anspruch auf eine Auszahlung der Versicherungssumme im Schadensfall.

Bei besonders hohen Versicherungssummen, wie beispielsweise in der Luftfahrtbranche, kann es vorkommen, dass das Risiko für einen einzelnen Versicherer zu hoch ist und ein Zusammenschluss mehrerer Versicherer notwendig wird. Zu diesem Zweck werden derzeit im sogenannten Beteiligungsgeschäft Konsortien aus mehreren Versicherern, den *Mitversicherern*, gebildet. Dabei wird das Risiko anteilig unter den beteiligten Versicherern aufgeteilt, wobei ein Teilnehmer des Konsortiums, der sogenannte Konsortialführer, Koordinationsaufgaben übernimmt.

Auftreten des Konsortiums gegenüber dem Versicherungsnehmer

Das Auftreten des Konsortiums gegenüber dem Versicherungsnehmer kann unterschiedliche Formen annehmen. In einem *verdeckten* Konsortium tritt der Konsortialführer gegenüber dem Versicherungsnehmer als einziger Vertragspartner auf, die Mitversicherer hingegen sind ihm unbekannt. In einem *offenen* Konsortium dagegen schließen alle Mitversicherer eigenständige Verträge mit dem Versicherungsnehmer ab.

In beiden Fällen beschränkt sich der Kontakt des Versicherungsnehmers nach dem Vertragsabschluss auf den Konsortialführer. Er erhält sämtliche Prämienzahlungen und ist für die Schadensabwicklung und -prüfung einschließlich der Auszahlung gegenüber dem Versicherungsnehmer zuständig. Sämtlicher Schriftverkehr vom und zum Versicherungsnehmer wird ebenfalls über den Konsortialführer abgewickelt.

Auftreten des Konsortialführers innerhalb des Konsortiums

Innerhalb des Konsortiums ist der Konsortialführer für die Aufteilung der Prämie sowie die Prüfung von Schadensfällen und das Zusammentragen der Versicherungssumme im Schadensfall zuständig. Die Aufteilung von Prämie und Risiko zwischen den Beteiligten wird individuell zwischen Führendem und den Beteiligten festgelegt. Dabei ist die Aufteilung des Risikos proportional zur Prämienaufteilung.

Zusätzlich werden Führungsprovisionen vereinbart, die von den Beteiligten an den Führenden gezahlt werden, um dessen zusätzlichen Aufwand zu decken. Diese Provisionen sind mit dem Konsortialführer individuell vereinbart und den anderen Mitversicherern nicht bekannt. Eine Festlegung der Provision ist anteilig (z.B. 2% der Gesamtprämie oder der anteiligen Prämie) oder absolut (z.B. 5000 €/Jahr) möglich. Die Provisionen können einmalig bei Vertragsabschluss (*Abschlussprovision*) oder bei jeder Prämienzahlung (*Bestandsprovision*) während der Vertragslaufzeit fällig sein. Die Provisionen

können separat gezahlt werden oder mit den Prämienzahlungen verrechnet werden.

Zahlungsflüsse

Grundsätzlich existieren in diesem Anwendungsszenario drei verschiedene Arten von Zahlungsflüssen zwischen den Mitversicherern und den Kunden:

- **Prämien** werden vom Versicherungsnehmer an den Konsortialführer gezahlt und von diesem an die Mitversicherer entsprechend der Verträge aufgeteilt.
- **Führungsprovisionen** werden von Mitversicherern an den Konsortialführer gezahlt.
- Bei der **Schadensregulierung** fließt Geld entsprechend der vereinbarten aufgeteilten Versicherungssummen von allen Mitgliedern des Konsortiums über den Konsortialführer an den Versicherungsnehmer.

5.3.1. Problemstellung

Auch in diesem Szenario entsteht grundsätzlich die Struktur einer 1-zu-N Beziehung der Teilnehmer, indem mehrere Teilnehmer untereinander und miteinander in den Prozessen agieren. Allerdings ergeben sich hier durch den Initiator einerseits Abhängigkeiten zu diesem und andererseits Sichtbarkeitseinschränkungen der Transaktionen zu anderen Teilnehmern.

Infolge der Verbindungen zwischen Konsortialführern und Mitversicherern und den bilateralen Verbindungen durch mehrere Verträge entsteht ein stark vermaschtes Netz zwischen den Versicherern. Bei Prämienzahlungen bzw. Zahlungen im Rahmen der Schadensabwicklung entstehen in diesem Netz eine Vielzahl von Einzelzahlungen. Dadurch, dass der Versicherungsnehmer

dem Konsortialführer den Gesamtbetrag überweist, muss der Führende diesen auf die Mitversicherer aufteilen. Andersherum kann der Führende in einem anderen Konsortium Mitversicherer sein und bekommt von dort wiederum eine Gutschrift durch die dortige Prämienzahlung.

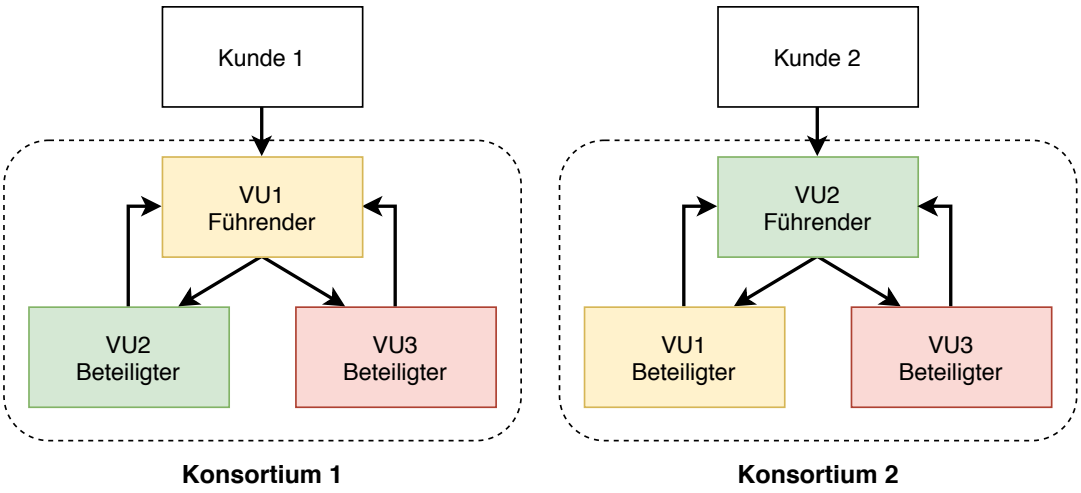

Abbildung 5.5.: Zahlungsflüsse in Konsortien

Die Zahlungen von Prämien und Provisionen ist in Abbildung 5.5 dargestellt. In der Abbildung sind insgesamt drei Versicherungsunternehmen (VU) gezeigt, die in zwei Konsortien unterschiedliche Rollen einnehmen. Daraus ergeben sich die unterschiedlichen Zahlungsflüsse.

Anstatt die Gelder miteinander zu verrechnen, werden derzeit für jede Zahlung einzelne Vorgänge ausgelöst, die einzelne Überweisungen zur Folge haben. Bei dem aufgezeigten Beispiel aus Abbildung 5.5 führt dies zu 10 Einzelbuchungen. Das bedeutet nicht nur einen hohen Aufwand durch die notwendigen Transaktionen, die zu versendenden Borderos und die zu prüfenden Vorgänge, sondern führt bei den beteiligten Versicherern auch zu mehr gebundenem Kapital, überflüssigen Buchungen und damit zu Liquiditäts- und/oder Zinsverlusten.

Ein weiteres Problem ergibt sich durch die unterschiedlichen Zahlungstermine von Prämien und Provisionen in unterschiedlichen Konsortien. Um den

eigenen Kontostand über mehrere Konsortien hinweg zu ermitteln, muss ein Versicherer in jedem Konsortium auf die Abrechnungslisten aller Beteiligten warten, diese mit den eigenen abgleichen und anschließend saldieren. Da diese Listen zu unterschiedlichen Zeitpunkten verfügbar sind und bislang manuell abgeglichen werden, ist der Aufwand erheblich und eine Saldierung nicht zu jedem Zeitpunkt möglich.

5.3.2. Anforderungen

Im Folgenden werden die Anforderungen an eine Blockchain-Plattform zur Abrechnung, die sich aus dem vorliegenden Anwendungsszenario ergeben, aufgeführt und erläutert. Die Anforderungen basieren auf einem repräsentativen Anwendungsszenario, welches einem Kooperationsprojekt mit der Firma *PPI AG*, einem Anbieter für Software- und Consulting-Lösungen für Banken, Versicherungen und Finanzdienstleister, entstammt.

Zugang Für die Abrechnung innerhalb einer Blockchain ist ein privates Zugangsmodell erforderlich. Dabei muss zunächst der generelle Zugang zur Blockchain auf die teilnehmenden Versicherungsunternehmen beschränkt werden. Weitere Restriktionen sind erforderlich, um die privaten Daten in den Transaktionen zu verschleiern, dabei handelt es sich jedoch nicht um Zugangsbeschränkungen an sich.

Identität Die Identitäten der Konsorten müssen zumindest dem Konsortialführer bekannt sein. Eventuelle Pseudonyme der Versicherer müssen für Saldierungszwecke in verschiedenen Konsortien gleich sein. Gleichzeitig darf es für Beteiligte nicht möglich sein, bei bekannter Identität eines Mitversicherers auf dessen weitere Mitgliedschaften in anderen Konsortien schließen zu können (siehe Abbildung 5.6).

Sichtbarkeit der Transaktionsinhalte Die Transaktionen können mitunter private Inhalte, wie Konditionen zu Prämien und Provisionen enthalten. Aus Gründen des Datenschutzes und der Wettbewerbsgeheimnisse dür-

fen diese Daten nicht für alle, sondern nur für die relevanten Parteien sichtbar sein.

Validierungsalgorithmus Die Transaktionen müssen mit geeigneten Mechanismen validiert werden, damit keine falschen Zahlungsströme oder Ergebnisse in die Blöcke aufgenommen werden. Weiterhin müssen alle Blöcke sofort final sein und dürfen nicht nachträglich durch etwaige Forks zurückgezogen werden.

Transaktionsgeschwindigkeit Transaktionen müssen in Echtzeit im Netz verteilt werden. Im Sinne der *Usability* für Endnutzer sollten Aktionen im Netzwerk so schnell abgearbeitet sein, dass ein Nutzer nicht den Eindruck erhält, als würde das System nicht auf Nutzereingaben reagieren.

Speicherplatz/-kapazität Der Speicherplatzbedarf der entstehenden Transaktionen und Blöcke sollte möglichst klein gehalten werden und auch nach mehrjährigem Betrieb nicht übermäßig viel Speicherplatz benötigen.

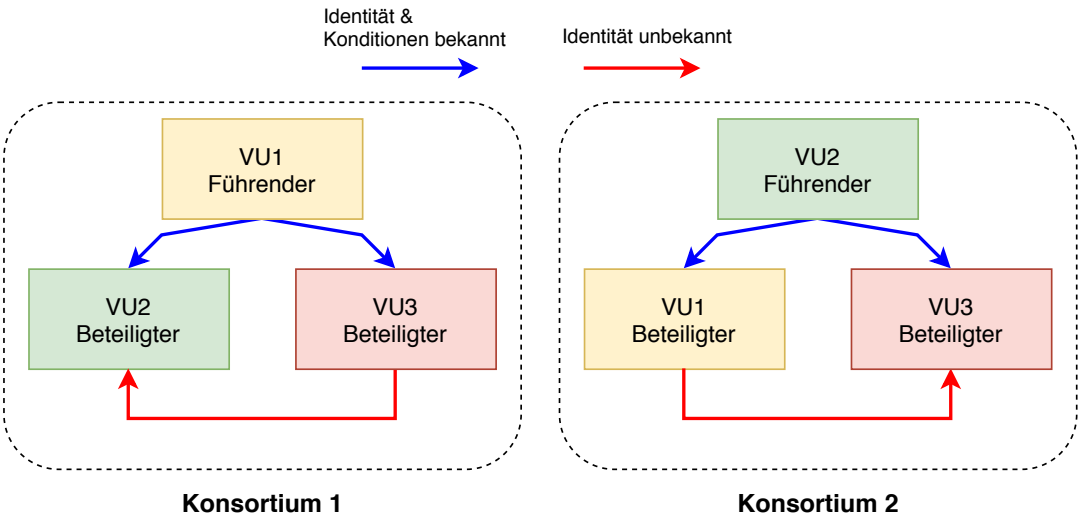

Abbildung 5.6.: Identitäten in verschiedenen Konsortien

5.3.3. Herausforderungen

In unterschiedlichen Konsortien gibt es Zahlungsströme von und zu Konsortialführern zu den jeweiligen Beteiligten, die miteinander nicht verrechnet werden (siehe Abbildung 5.5).

Insbesondere, wenn Versicherer in mehreren Konsortien beteiligt bzw. führend sind, und so sich gegenseitig ausgleichende Zahlungen entstehen, ließe sich die Anzahl der Buchungsvorgänge deutlich reduzieren.

Dazu soll ein dezentrales System auf Blockchain-Basis entworfen werden, welches über die zu tätigenden Zahlungen Buch führt, diese saldiert und anschließend die resultierenden notwendigen Buchungen ausgibt. Dabei ist zwischen zwei Ansätzen zu unterscheiden: Saldierung über Konsortiumsgrenzen hinweg und konsortiumsinterne Saldierung. Je nach gewähltem Ansatz ist eine weitere dritte unabhängige Partei erforderlich oder nicht.

Die ermittelten Herausforderungen für eine dezentrale, Blockchain-basierte Abbrechnung sind insbesondere:

Sichtbarkeit der Daten Da viele der Zahlungsströme und Beziehungen zwischen Versicherern nicht-öffentlich sind, stellt die Sichtbarkeit der Transaktionsinhalte die größte Herausforderung dar. Die Umsetzbarkeit der Sichtbarkeitsstufen selbst ist dabei grundsätzlich durch kryptografische Verfahren (asymmetrische Verschlüsselung) gegeben, zumal die Vertragsparteien nicht stark fluktuieren und einander bekannt sind.

Validierung Die Verschlüsselung der gesamten Transaktionsinhalte hätte zur Folge, dass keine Validierung durch alle Teilnehmer mehr möglich wäre. Ebenso ist eine versicherer- und konsortienübergreifende Saldierung damit nicht mehr möglich. Da es sinnvollerweise keine zentrale Instanz geben soll, die eine volle Sichtbarkeit auf die Transaktionen hat, muss ein anderer Weg gefunden werden, um „blind“ Transaktionen validieren zu können und die Saldierung vorzunehmen.

Die Sichtbarkeiten von weiteren Vertragsdaten, wie Daten über den Versicherungsnehmer oder das Versicherungsobjekt, sind keine Herausforderungen in diesem Sinne, da sie weder zur Validierung noch zur Saldierung zwingend erforderlich sind.

Anonymität der Teilnehmer Die Identitäten der Teilnehmer sind mindestens dem Konsortialführer bekannt. Darüber hinaus darf es keine Rückschlüsse auf Mitgliedschaften in anderen Konsortien geben, d.h. ein Konsortialführer, der in einem zweiten Konsortium nur beteiligt ist, darf die Identität eines dort Beteiligten nicht aufgrund seine Führungsrolle im ersten Konsortium kennen (siehe Abbildung 5.6).

5.3.4. Eignung einer Blockchain-basierten Umsetzung

Um die Eignung des Szenarios für eine Blockchain-basierte Umsetzung aufzuzeigen, werden zunächst die Herausforderungen bezüglich der Anforderungen *Sichtbarkeit* und *Validierung* aufgezeigt. Anschließend soll zunächst eine mögliche klassische Umsetzung ohne Blockchain skizziert werden. Darauffolgend werden Aspekte des Szenarios untersucht, die für oder gegen einen Lösungsansatz mit Blockchain-Einsatz sprechen. Abschließend wird aufgezeigt, ob und warum eine Blockchain-basierte Lösung Vorteile gegenüber einem konventionellem System aufweist.

Sichtbarkeit der Transaktionsinhalte

Die größte Herausforderung der MAP ist die eingeschränkte Sichtbarkeit der Transaktionsinhalte. In den meisten Blockchain-Technologien werden Transaktionen in Blöcken zusammengefasst. Um einen Block zu validieren, müssen alle Transaktionen ebenfalls validiert werden.

Im vorliegenden Szenario wäre die Validierung z.B. die Prüfung darauf, ob eine Buchung dem Vertrag des Konsortiums entspricht oder ob eine gegebene

Saldierung korrekt ist (z.B. Prämienbetrag = Summe der aufgeteilten Prämienbeträge bei Prämienzahlungen). Wenn in einer Mitversicherung pro Vertragsabschluss eine Transaktion erstellt wird, diese aber verschiedene Sichtbarkeiten hat, ist es nicht ohne Weiteres möglich, die einzelnen Geldbeträge aufzusummieren, um die Transaktion zu validieren. Da es keine zentrale Instanz gibt bzw. geben soll, die eine volle Sichtbarkeit auf die Transaktionen hat, muss ein anderer Weg gefunden werden, um „blind“ Transaktionen verifizieren zu können. Die Sichtbarkeiten von weiteren Vertragsdaten, wie Daten über den Versicherungsnehmer oder das Versicherungsobjekt, sind keine Herausforderungen in diesem Sinne, da sie weder zur Validierung noch zur Saldierung zwingend erforderlich sind.

Die Umsetzbarkeit der Sichtbarkeitsstufen selbst ist dabei grundsätzlich durch kryptografische Verfahren (asymmetrische Verschlüsselung) gegeben, zumal die Vertragsparteien nicht stark fluktuieren und einander bekannt sind.

Sichtbarkeit / Daten	Global	Intern	Führender
Versicherungsnehmer		x	
Versicherungsobjekt		x	
Versicherungssumme		x	
Prämienverteilung			x
Führungsprovisionen			x
Schadenszahlung			x
Beteiligungsbeziehung			x

Tabelle 5.1.: Sichtbarkeiten in Mitversicherung

Die Tabelle 5.1 zeigt die maximalen Sichtbarkeiten der verschiedenen Vertragsbestandteile bzw. Zahlungsströme. *Global* bezeichnet dabei die Sichtbarkeit unter allen Teilnehmern der Blockchain, *Intern* die Sichtbarkeit innerhalb

eines Konsortiums für alle Beteiligten und *Führender* die Sichtbarkeit nur für den Führenden des entsprechenden Konsortiums (sowie für den betreffenden Beteiligten selbst). Die Zeile *Prämienverteilung* schließt dabei die konkreten Zahlungen der Prämie ein, genauso wie die Zeile Führungsprovisionen Informationen sowohl über die Aufteilung als auch die Provisionszahlungen selbst umfasst.

Validierung

Aufgrund der Sichtbarkeitsbeschränkungen ist eine Transaktionsvalidierung nicht ohne weitere technische Maßnahmen erreichbar. Auch ein partielles Validierungsverfahren, in dem nur bestimmte Teile validiert werden, ist nicht trivial. Einige Konsensvarianten (beispielsweise *delegated-proof-of-stake*) erlauben es, einem Teilbereich der Validatoren die komplette Validierung zu überlassen. Hierbei ist die Identifizierbarkeit der Validatoren ein Problem. Angenommen, es würde nur den Teilnehmern, die Teil eines Versicherungsvertrages (Konsortiums) sind, die Validierung überlassen, dann könnten in diesem Fall Unbeteiligte durch Interpretation der ausgetauschten Nachrichten darauf schließen, wer Teil eines Vertrages ist.

Die Situation wird durch die notwendigen unterschiedlichen Sichtbarkeitsstufen noch einmal komplexer: Einige Details, wie etwa die Höhe der Provisionszahlung sowie der Prämien, sind nicht einmal innerhalb eines Konsortiums allen bekannt und können somit nicht von allen Teilnehmern des Konsortiums validiert werden.

Klassische Umsetzung

Die im Anwendungsszenario geschilderte Problemstellung ließe sich auch in einer klassischen Umsetzung lösen. Dabei wäre der naheliegende Ansatz, eine zentrale Instanz (*Intermediär*) für Abrechnungs- und Zahlungszwecke einzusetzen. Diese verwaltet alle existierenden Konsortien, empfängt Zah-

lungen der Versicherungsnehmer und teilt sie den Verträgen entsprechend auf. Schadensfälle lassen sich analog abbilden. Da alle Transaktionen über die zentrale Instanz abgewickelt werden, ist eine konsortienübergreifende Saldierung leicht möglich.

Einige der Anforderungen, wie eingeschränkte Sichtbarkeit von Transaktionsdetails, wären durch diesen Intermediär leicht umzusetzen, da er die Gesamtheit der Transaktionen kontrolliert. Eine Verschlüsselung der Daten auf Transaktionsebene (nicht Transportverschlüsselung) ist nicht zwingend erforderlich, da Sichtbarkeit der Daten durch ein Zugriffsrechtemanagement verwaltet werden kann. Der Intermediär kann Daten im Klartext speichern (ausgehend von einer sicheren Datenbankinstanz) und diese, basierend auf den jeweiligen Zugriffsrechten, dynamisch für den Abrufenden zuschneiden und bereitstellen.

Das Validieren der einzelnen Transaktionen kann durch die globale Sicht des Intermediärs trivial gelöst werden. Bei Disputen zwischen Teilnehmern kann der Intermediär schnell Missstände aufdecken, da ihm alle Transaktionen der Parteien offen liegen.

Insbesondere lassen sich in dieser Lösung die Verwaltung von Konsortien, Abrechnung/Saldierung und die eigentlichen Zahlungsvorgänge automatisieren und vereinen.

Vor- und Nachteile einer Blockchain-Umsetzung

Im Folgenden sollen die Vor- und Nachteile einer Umsetzung des Anwendungsszenarios mittels Blockchain gegenüber dem vorab skizzierten klassischen Ansatz analysiert werden.

Basierend auf einem Peer-to-Peer-Netzwerk bieten Blockchain-Technologien vor allem in einem Marktplatzszenario den Vorteil, dass es keine zentrale Instanz gibt, für deren Nutzung Entgelte zu entrichten sind, z.B. durch Provisionen. Die Notwendigkeit von Intermediären ist nicht mehr gegeben.

Die Nutzung einer verteilten Datenbank zwischen allen Teilnehmern gewährt die gleiche Sicht auf den aktuellen Zustand aller Transaktionen. Kein Teilnehmer kann Vorteile z.B. durch finanzielle Überlegenheit erlangen. Dies sorgt besonders in einem offenen Markt für eine faire und gleichwertige Handelsmöglichkeit aller Parteien. Es kann z.B. auch keine unterschiedlich hohen Gebühren für unterschiedliche Teilnehmer geben.

Gegenüber einer klassischen, zentralisierten Lösung führt ein Blockchain-basierter Ansatz daher zu einer Liberalisierung des Marktes und erübrigt eine von den Marktteilnehmern gegründete Institution, die den Marktplatz als gemeinsame Dienstleistung anbietet.

Trotz der Vorteile der zwei vorigen Punkte stellen diese auch Nachteile dar. In einem P2P-Netzwerk ist der Zugang zum selbigen nicht immer trivial. Für ein Marktplatzsystem muss es fest definierte Zugangspunkte geben, damit Teilnehmer dem Netzwerk beitreten können. Für Versicherungsunternehmen steht die Sicherheit beim Betrieb der Zugangspunkte (samt angebundener Softwarelösungen) im Fokus. Generell ist ein Zugriff durch Dritte in privaten Blockchains nicht vorgesehen oder gewünscht. Um Kunden hier Zugriff zu gewähren, muss ein Stellvertretersystem modelliert werden.

Genauso müssen Zugriffsrechte, insbesondere Schreibrechte, auf bestimmte Parteien limitiert werden. Das heißt, es muss verhindert werden, dass Akteure, die keine Versicherungsunternehmen oder Teil eines Konsortiums sind, am Prozess teilnehmen. Ebenso muss verhindert werden, dass das Netzwerk mutwillig mit Transaktionen geflutet werden kann und so überlastet wird. Beides ist im skizzierten klassischen Ansatz deutlich einfacher zu lösen als in einem dezentralen Blockchain-basierten Ansatz.

Abschließend erfüllt das dargestellte Anwendungsszenario einige wichtige Kriterien, die es für den Einsatz einer Blockchain-Lösung geeignet machen: Darunter fallen eine vollständige Transparenz der Transaktionsinhalte, soweit aus Datenschutzgründen möglich, sowie die (teilweise) Anonymität der Teilnehmer, 1-zu-N Kommunikationsbeziehungen zwischen den Teilnehmern, und es gibt keine Prüfungen oder Berechnungen, die von einer zentralen In-

stanz durchgeführt werden müssten. Auch bietet eine Blockchain-basierte Lösung eine Reihe von Vorteilen gegenüber einer klassischen Lösung, insbesondere den Verzicht auf die dann nicht mehr notwendige zentrale Instanz und deren Verwaltung.

5.3.5. Prozessadaption für eine Blockchain-basierte Umsetzung

Das hier vorgestellte Konzept für eine Prozessumsetzung basiert auf einer Blockchain, auf die alle Teilnehmer der Zahlungsplattform sowohl lesenden als auch schreibenden Zugriff haben. Technologisch kommt dafür entweder eine Smart Contract-fähige Blockchain (z.B. Ethereum) oder eine generalisierbare Blockchain, die arbiträre Transaktionen verarbeitet und zur Validierung an ein angebundenes Business-Plugin weitergibt, in Betracht. In jedem Fall ist ein PoS-Konsens oder BFT-PoS-Konsens vorzuziehen, aufgrund der vorgestellten unterschiedlichen Anforderungen wie der geringeren Betriebskosten, Blockbildungsgeschwindigkeit, Zugriffskontrolle und der zwingend erforderlichen Blockfinalität.

Alle Geschäftsprozesse werden nun innerhalb der Blockchain als Transaktionen abgebildet. Diese werden möglichst umfangreich validiert, um die Integrität der Daten und damit der dahinterliegenden Prozesse sicherzustellen. Die Validierung wird dabei entweder von allen (z.B. Ethereum) oder einer vorher festgelegten Menge von Validatoren durchgeführt.

Obwohl die (Roh-) Daten der Blockchain von allen Beteiligten lesbar sind, sollen dennoch verschiedene Sichtbarkeiten etabliert werden. Die Umsetzung einer partiellen Sichtbarkeit unter Gewährleistung einer Validierung wird nachfolgend mit homomorpher Verschlüsselung und Commitments erarbeitet.

Verschlüsselung und bindende Commitments

Ein Commitment-Verfahren ist ein kryptografisches Protokoll, welches es einer Partei A erlaubt, sich gegenüber einer Partei B durch das Senden eines Commitments *com* auf einen Wert *m* festzulegen, ohne diesen offenzulegen [Brassard u. a. 1988]. Dieser Wert kann später *aufgedeckt* werden, d.h. der Partei B wird der Wert *m* im Klartext übermittelt und Partei B kann prüfen, ob das Commitment *com* korrekt war.

Commitment-Verfahren müssen zwei wesentliche Eigenschaften erfüllen [Katz und Lindell 2014, S. 187]:

Hiding Es dürfen auf Basis des übermittelten Commitments keine Rückschlüsse auf den Wert *m* möglich sein.

Binding Es darf nicht möglich sein, ein Commitment im Nachhinein mit einem anderen als dem vorab festgelegten Wert aufzudecken, d.h. der Absender muss an sein Commitment *gebunden* sein.

Ein einfacher Anwendungsfall ist eine Münzwurfwette: Zwei Parteien A und B werfen eine Münze. Zeigen beide Münzen die gleiche Seite, gewinnt A, zeigen sie unterschiedliche Seiten, gewinnt B. Diese Wette ist sehr einfach durchzuführen, wenn sich die Parteien physisch gegenüber stehen. Wird dieselbe Wette jedoch über eine Entfernung (z.B. per Telefon) durchgeführt, ergibt sich ein Problem: Meldet Partei A zuerst das Ergebnis des Münzwurfs, kann Partei B immer so lügen, dass sie gewinnt. Gleiches gilt für den umgekehrten Fall. Durch ein Commitment-Verfahren können sich dagegen beide Parteien auf einen Wert festlegen, ohne ihn vorab preiszugeben, und die Wette durchführen.

Die Eigenschaften *hiding* und *binding* können in realen Commitment-Verfahren nicht beide uneingeschränkt erfüllt werden [Delfs und Knebl 2015, S. 129]. Stattdessen muss man sich bei einer Eigenschaft auf *computational* hiding/-binding beschränken. In diesem Fall ist es für einen Angreifer aufgrund der notwendigen Rechenleistung bzw. der Komplexität des zugrunde liegenden

mathematischen Problems nicht sinnvoll möglich, den Wert aufzudecken bzw. im Nachhinein das Commitment zu fälschen.

Commitment mit Hashfunktion

Ein simples Commitment-Verfahren kann durch die Nutzung von kryptografischen Hashfunktionen realisiert werden. Als Commitment der Nachricht m wird folgender Wert berechnet:

$$com(m) = Hash(m) \tag{5.1}$$

Das Ergebnis $com(m)$ wird anschließend an den Empfänger übermittelt. Das Problem dieses Verfahrens liegt in dem üblicherweise eingeschränkten Wertebereich der Commitments: Werden z.B. Geldbeträge oder sogar nur das Ergebnis eines Münzwurfs übermittelt, lässt sich leicht für alle sinnvollen Eingabemöglichkeiten das Commitment berechnen, beispielsweise durch Brute-Forcing, und mit dem Übermittelten vergleichen, um den richtigen Wert so zu erraten.

Blindingfaktor

Um dieses Problem zu lösen, wird ein Blindingfaktor bf eingeführt. Dieser wird zufällig gewählt und einfach mit in die Berechnung einbezogen:

$$com(m) = Hash(m \cdot bf) \tag{5.2}$$

Zum Aufdecken wird nun zusätzlich zum Klartext m der Nachricht der Blindingfaktor bf übermittelt, sodass das Commitment geprüft werden kann.

Der Empfänger kann nun nicht mehr den richtigen Wert erraten, da verschiedene Kombinationen aus m und bf zum selben Hash führen könnten und somit alle Werte m für ein bestimmtes Commitment $com(m)$ gleich wahrschein-

lich sind. Das Verfahren ist *unconditional hiding*. Allerdings ist der Absender des Commitments nicht auf einen Wert festgelegt: Er kann leicht zwei Werte m' und bf' wählen, so dass $m' \cdot bf' = m \cdot bf$ gilt, und damit einen anderen Wert aufdecken. Das Verfahren ist daher in dieser Form nicht bindend.

Pedersen Commitment

Das Verfahren von Pedersen [Pedersen 1991] ermöglicht Commitments auf Basis von zyklischen Gruppen, die *unconditional hiding* und *computational binding* sind. Dazu wird eine zyklische Gruppe der Ordnung N gewählt (z.B. ganze Zahlen *modulo* N), wobei N einige Eigenschaften erfüllen muss, auf die an dieser Stelle nicht näher eingegangen wird.

Es werden folgend die Werte g und h gewählt, die Generatoren der Gruppe darstellen, d.h. g^x erzeugt für $x \in \{1, 2, 3, \dots\}$ alle Elemente der Gruppe. h sei gewählt durch $h = g^a mod\ p$, wobei a geheim ist, sodass niemand $log_g(h)$ kennt. Die Parameter g, h und N sind dabei allen Kommunikationspartnern bekannt. Auch hier wird wieder ein Blindingfaktor bf benötigt, der beim Aufdecken mitgeschickt wird.

Ein Pedersen Commitment wird nun berechnet als:

$$PC(m) = g^m \cdot h^{bf}\ mod\ N \tag{5.3}$$

Um einen falschen Wert m' aufzudecken, müsste der Absender einen neuen Blindingfaktor bf' passend zum neuen Wert m' berechnen, so dass gilt:

$$g^m \cdot h^b f = g^{m'} \cdot h^{bf'} \tag{5.4}$$

Die Berechnung des neuen Blindingfaktors ist aber äquivalent zur Berechnung des diskreten Logarithmus [Pedersen 1991]:

$$log_g(h) = \frac{m - m'}{bf' - bf} \; mod \; N \tag{5.5}$$

Dieses System basiert in seiner Annahme der *binding*-Eigenschaft also auf dem diskreten Logarithmus Problem und ist somit *computational binding* und stellt bis dato ein mit endlicher Rechenleistung unlösbares Problem dar.

Zusammenfassend bieten Commitment-Verfahren eine Möglichkeit, sich öffentlich (z.B. in einer Blockchain) auf einen Wert festzulegen, der später von jedem, an den man diesen Wert übermitteln möchte, überprüfbar ist. Die homomorphen Eigenschaften der Pedersen Commitments gehen aber noch über diesen Anwendungszweck hinaus, wie im folgenden Abschnitt beschrieben wird.

Da auch auf elliptischen Kurven zyklische Gruppen definiert werden können, können Pedersen Commitments auch auf Basis elliptischer Kurven realisiert werden, was auch durch die kurzen Schlüssellängen und der homomorphen Eigenschaften vorteilhaft ist.

Homomorphe Verschlüsselung

Wenn ein Homomorphismus zwischen zwei mathematischen Gruppen $(G, *)$ und $(H, \star)$ existiert, dann gilt für eine Funktion $\phi : G \to H$ (Verschlüsselung) folgender Zusammenhang (mit $g_i \in G$):

$$\phi(g_1 * \cdots * g_n) = \phi(g_1) \star \cdots \star \phi(g_n) \tag{5.6}$$

Eine Verschlüsselung ist daher z.B. homomorph, wenn die Verschlüsselung einer Summe dem Produkt der verschlüsselten Summanden entspricht.

Die Pedersen Commitments erfüllen diese Eigenschaften: Einzelne Pedersen Commitments lassen sich summieren, ohne die Klartexte zu kennen. Konkret gilt im Fall der Pedersen Commitments auf Modulo-Restklassen:

$$PC(m_1) \times \cdots \times PC(m_n) = PC(m_1 + \cdots + m_n) \tag{5.7}$$

Bezieht man die Blindingfaktoren ein, muss die Summe als Pedersen Commitment als Blindingfaktor auch die Summe der Blindingfaktoren der Summanden verwenden:

$$PC(m_1, bf_1) \times \cdots \times PC(m_n, bf_n) = PC(m_1 + \cdots + m_n, bf_1 + \cdots + bf_n) \tag{5.8}$$

Diese homomorphe Eigenschaft lässt sich ausnutzen, um Validierungen durchzuführen. Commitments, die unterschiedlichen Zahlungen entsprechen, können von allen Teilnehmern einer Blockchain aufsummiert und auf Gleichheit mit einem anderen Commitment geprüft werden, welches die gewünschte Summe enthält.

Die Nutzung von asymmetrischer Kryptografie (bzw. hybriden Verschlüsselungsverfahren) ermöglicht es den Teilnehmern, Inhalte innerhalb einer Nachricht bzw. Transaktion verschlüsselt zu übermitteln. Diese Inhalte sind anschließend nur für denjenigen lesbar, der den privaten Schlüssel besitzt. Eine Transaktion kann sowohl verschlüsselte als auch unverschlüsselte Informationen enthalten, sodass andere Teilnehmer trotzdem Teile des Inhaltes validieren können.

Um asymmetrische Verschlüsselung einsetzen zu können, müssen die öffentlichen Schlüssel der partizipierenden Teilnehmer öffentlich bekannt sein. Der Austausch kann entweder *off-chain* (z.B. per E-Mail) oder *on-chain* durch Transaktionen erfolgen. Da es nicht erforderlich ist, den öffentlichen Schlüssel geheimzuhalten, sollte dieser vorzugsweise über einen *on-chain* Mechanismus verteilt werden, da er somit für alle Teilnehmer gleichzeitig bekannt wird.

Damit Außenstehende keinen Einblick auf die teilnehmenden Parteien gewinnen können, sollten verschlüsselte (Teil-) Transaktionen ohne Adressat sein. Bei herkömmlicher Kommunikation ist dies natürlich undenkbar, da z.B. beim Versand von E-Mails immer ein Adressat erforderlich ist. Da Blockchains allerdings Transaktionen immer an alle Teilnehmer weiterleiten, ist dies hier ohne Adressat möglich. Das hat zur Folge, dass ein Teilnehmer mehrere Entschlüsselungsversuche vornehmen muss, um herauszufinden, ob eine Teilnachricht für ihn bestimmt ist. Im Gegenzug bleibt er aber im Gesamtsystem anonym.

Für eine Umsetzung im Mitversicherungsgeschäft wird eine Transaktion deshalb in mehrere Teile mit unterschiedlichen Eigenschaften aufgeteilt:

public Ein öffentlicher Teil, der für alle Teilnehmer sichtbar ist und keinerlei sicherheits- oder datenschutzrelevante Inhalte enthält, beispielsweise eine verschleierte Vertrags-ID ($hash(contract\_id)$) zur Identifikation.

semi-private Ein nicht-öffentlicher Teil, der nur Informationen für die Mitversicherer eines Konsortiums enthält. Diese (bzw. ihr symmetrischer Schlüssel) sind einzeln für jeden Mitversicherer eines Vertrags verschlüsselt. Die Inhalte können beliebig gewählt werden und unterliegen keinerlei Restriktionen, wie zum Beispiel: Klartext-Vertrags-ID, Laufzeit, Zahlweise und andere allgemeine Informationen und Metadaten.

private Ein privater Teil, der Informationen enthält, die nur für einen bestimmten Mitversicherer bestimmt sind. Er ist mit seinem öffentlichen Schlüssel verschlüsselt und enthält beispielsweise Beträge und Blindingfaktoren in lesbarer Form.

commitments Eine Liste von Zahlungsein- oder ausgängen, die aus diesem Vertrag resultieren, als Commitment (PC). Diese sind aufgrund der Eigenschaft der Pedersen-Commitments ebenfalls nicht öffentlich einsehbar, können aber zur Validierung der Transaktion verwendet werden.

commitment-sum Die Summe aller Commitments (PC_{sum}) wird als Referenzwert benötigt, um Zahlungsanteile validierbar zu machen.
Für die Commitments gilt immer: $\sum_{i=1}^{n} PC_i = PC_{sum}$

Damit Pedersen-Commitments zu Validierungszwecken genutzt werden können, muss bei deren Konstruktion einiges beachtet werden. Zwar können die Commitments von jedem Teilnehmer summiert werden, das Ergebnis ist jedoch wieder ein Pedersen-Commitment und hat zunächst keine Bedeutung. Erst der Vergleich mit einer zweiten Summe kann eine Validierung darstellen.

Sollen zwei Summen verglichen werden, müssen auch die Summen der Blindingfaktoren gleich sein. Nachfolgendes Beispiel stellt ein Zahlungssystem dar, bei dem eine Transaktion n Eingänge in_x und m Ausgänge out_y hat, die jeweils als Pedersen-Commitments dargestellt werden und deren Summen immer gleich sein müssen ($\sum_x(in_x) = \sum_x(out_x)$). Die zu übermittelnden Beträge seien als $v_{in,x}$ bzw. $v_{out,x}$ festgelegt.

Anschließend werden die Pedersen-Commitments dieser Werte mit r_x als Blindingfaktoren berechnet:

$$in_1 = PC(v_{in,1}, r_{in,1}) \qquad out_1 = PC(v_{out,1}, r_{out,1})$$
$$in_2 = PC(v_{in,2}, r_{in,2}) \qquad out_2 = PC(v_{out,2}, r_{out,2})$$
$$in_3 = PC(v_{in,3}, r_{in,3}) \qquad out_3 = PC(v_{out,3}, r_{out,3})$$

Damit ein Teilnehmer nun die Eingangssumme $\sum_x(in_x)$ mit der Ausgangssumme $\sum_x(out_x)$ vergleichen kann, können die Blindingfaktoren beispielsweise wie folgt gewählt werden:

$$in_1 = PC(20, 23) \qquad out_1 = PC(1, 20)$$
$$in_2 = PC(40, 42) \qquad out_2 = PC(100, 13)$$
$$in_3 = PC(60, 11) \qquad out_3 = PC(19, 43)$$

Die Berechnung der Summen findet wie folgt statt:

$$\sum_x (in_x) = in_1 + in_2 + in_3 = PC(20 + 40 + 60, 23 + 42 + 11) \quad = PC(120, 76)$$
$$\sum_x (out_x) = out_1 + out_2 + out_3 = PC(1 + 100 + 19, 20 + 13 + 43) \quad = PC(120, 76)$$

So können nun die beiden Commitments auf Gleichheit geprüft werden. Das impliziert auch, dass ein Teilnehmer, der ein Pedersen-Commitment einstellt, welches in eine Validierung einbezogen werden soll, alle relevanten Blindingfaktoren kennen muss. Nur dann kann er den Blindingfaktor für das eigene Commitment so berechnen, dass die zu validierenden Summen gleich sind.

Zusammenfassung

Besonders verteilte Geschäftsprozesse unterliegen speziellen Herausforderungen in Bezug auf deren technische Umsetzung. Bei der Umsetzung auf eine Blockchain-basierte Lösung entstehen neue Herausforderungen, dadurch dass ein Intermediär als vertrauensvolle Instanz fehlt.

Bei der technischen Umsetzung innerhalb eines offenen Blockchain-Systems müssen also notwendigerweise Teile des Prozesses adaptiert werden, damit sie den besonderen Ansprüchen von Geschäftsprozessen genügen, beispielsweise hinsichtlich der Zugangsbeschränkungen, Sichtbarkeit und Sicherheit. Zusätzlich zu den generellen Anforderungen in Geschäftsprozessen kommen je nach Anwendungsszenario noch weitere prozessspezifische Anforderun-

gen hinzu, wie im vorangehenden Kapitel anhand der Beispiele des Energiehandels und der Mitversicherung dargestellt.

Aufbauend auf den in diesem Kapitel dargestellten Anforderungen und Herausforderungen soll im nachfolgenden Kapitel 6 der Entwurf für das Konzept eines generalisierten Blockchain-Frameworks stattfinden.

6. Wesentliche Konzepte einer generalisierten Blockchain

Nachdem im vorhergehenden Kapitels bereits Anforderungen, Herausforderungen und mögliche Wege für eine Prozessadaption für eine Blockchain-basierte Umsetzung für die dargestellten Prozesse erfolgt sind, wird nun nachfolgend ein Konzept zur Umsetzung einer generalisierten Blockchain vorgestellt. Zuerst wird, in Anbetracht der umzusetzenden Prozesse, auf die Nachteile der bestehenden Frameworks eingegangen, worauf folgend ein Konzept für eine Eigenentwicklung mit notwendigen Funktionen mit spezifischem Fokus auf Geschäftsprozesse und Generalisierbarkeit entwickelt wird.

6.1. Nachteile bestehender Frameworks

Aufgrund der starren Architektur und dem eingeschränkten Einsatzfeld von Cryptocurrency-Blockchains, wie Bitcoin, lassen sich diese nicht trivial für den Einsatz in einem der beiden Szenarien umfunktionieren, geschweige denn für beide Szenarien gleichzeitig verwenden. Hier müssten für beide Szenarien separat Anpassungen getätigt werden, was den Aufwand generell erhöht, aber auch für zukünftige weitere Prozesse keine Generalisierbarkeit und leichte Adaption ermöglicht.

Eine Umsetzung auf einer Blockchain mittels Smart Contracts, wie Ethereum, ist auf den ersten Blick durchaus denkbar, wird aber durch Laufzeit- und Kostenprobleme erschwert. Beide Prozesse erfordern, dass die gesamte

oder Teile der Transaktion vor anderen Teilnehmer verborgen werden, was durch Verschlüsselung umgesetzt werden muss, da es keine privaten Nachrichtenkanäle in einer offenen Blockchain geben kann. Private Nachrichtenkanäle würden das Sichtbarkeitsproblem lösen, jedoch wird der Transparenz-Charakter der Blockchain so zunichte gemacht, da nicht mehr alle Knoten an der Validierung und Überprüfung von Transaktionen beteiligt sind. Zusätzlich zu den Sichtbarkeitsproblemen, kommt hinzu, dass Verschlüsselungs- und Entschlüsselungsoperationen innerhalb eines Smart Contracts zu kostenintensiv sind. Jede ausgeführte Operation innerhalb eines Smart Contracts verbraucht Gas (im Falle von Ethereum). Einfache arithmetische Operationen benötigen relativ wenig, komplexere und vor allem Speicheroperationen hingegen deutlich mehr Gas. Weiterhin gibt es in Smart Contracts keine Möglichkeit sicher private Schlüssel zu hinterlegen, diese sind für alle Teilnehmer öffentlich einsehbar.

Um das Problem der Gas- und Laufzeitkosten in Ethereum aufzuzeigen, wurde eine Doppelauktion für Smart- und Micro-Grids prototypisch innerhalb eines Smart Contracts umgesetzt und evaluiert. Doppelauktionen erfordern das Sortieren von Teilnehmern in zwei Listen nach deren Kauf- und Verkaufsgeboten. Eine Liste muss aufsteigend und die andere absteigend sortiert werden. Anschließend wird der Gleichgewichtspreis berechnet und Angebot und Nachfrage entsprechend zugeteilt. Bei 100 Teilnehmern übersteigen die Gaskosten bereits das maximale Limit an Gas, was innerhalb eines Blockes auf der öffentlichen Ethereum-Blockchain verwendet werden kann, wie in Abbildung 6.1 dargestellt. Dies würde außerdem dazu führen, dass nur diese eine Transaktion, die die Doppelauktion ausführt, im Block inkludiert wäre und keine weitere [Stübs u. a. 2020].

Aus den vorher genannten Gründen können also nicht alle Geschäftsprozesse innerhalb von Smart Contracts oder Cryptocurrency-Blockchains umgesetzt werden. Es wird also eine generalisierbare Blockchain, wie *Hyperledger* oder *Tendermint*, zur Umsetzung benötigt. Diese haben hingegen andere Schwächen aufgrund der gewählten Designentscheidungen. Beispielsweise kann es in Hyperledger Fabric durch den gewählten Execute-Order-Validate

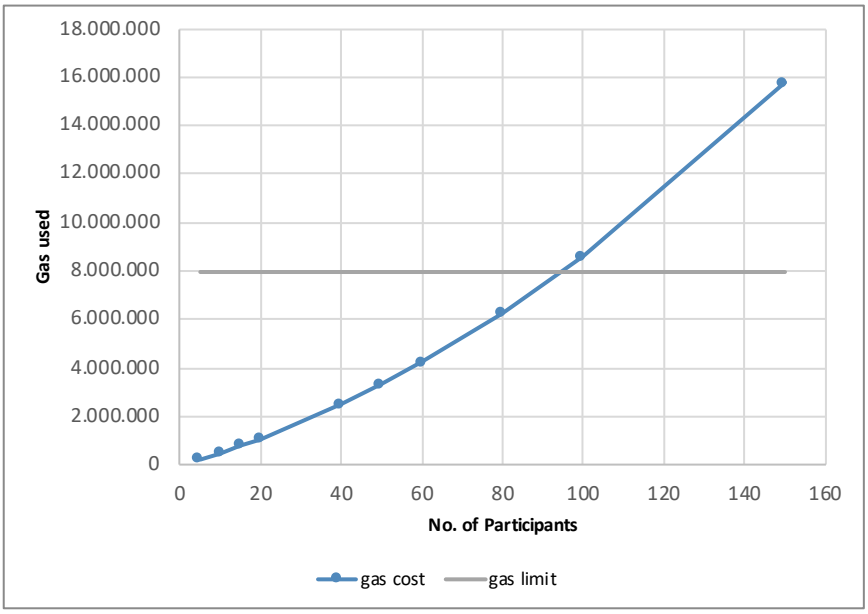

Abbildung 6.1.: Gaskosten für Doppelauktion [Stübs u. a. 2020]

Prozess durchaus dazu führen, dass ungültige Transaktionen im Block persistiert werden, was unnötig Speicherplatz belegt und besonders Prozessen mit vielen speicherintensiven Transaktionen auf lange Sicht zum Problem wird. Vor allem liegt dieses Problem in der Wahl einer dritten und immer semantikunabhängigen Komponente zur Festlegung der Reihenfolge. Der Nachteil einer nicht Geschäftsprozess-basierten Sortierung ist auch bei Tendermint zu finden. Hier werden Transaktionen vom Framework nach dem *FIFO*-Prinzip sortiert. Eine Transaktion, die bereits als invalide markiert wurde, weil sie im Block in der falschen Reihenfolge vorkam, kann auch nicht wiederholt eingestellt werden, da das Framework sie an dieser Stelle über ihren Hash als Duplikat erkennt und direkt ablehnt. Auch wenn Transaktionen vom Ersteller in der richtigen Reihenfolge ins Netzwerk gegeben wurden, heißt dies nicht, dass diese auch in dieser Reihenfolge beim aktuell zuständigen Proposer im Mempool vorkommen.

6.2. Erforderliche Komponenten und Prozesse

Die bisher vorgestellten Frameworks sind konzeptuell und strukturell im Aufbau und der verwendeten Komponenten sehr ähnlich zueinander. Wie bereits zuvor dargestellt, beinhalten diese die gleichen logischen Komponenten für die Abwicklung, Speicherung, Verteilung und Validierung von Transaktionen und Blöcken. Aufgrund der ähnlichen Komponenten und Abhängigkeiten dieser, sind folglich auch die Prozesse in ihren Grundfunktionen vergleichbar. Deshalb lassen sich diese Grundkonzepte vereinheitlichen und generalisieren. Nachfolgend werden dazu die erforderlichen Hauptkomponenten und deren Zusammenspiel sowie der daraus resultierenden Prozesse einer generalisierten Blockchain vorgestellt.

6.2.1. Komponenten

Für den Grundbetrieb einer Blockchain sind die vier Komponenten Netzwerk, für die Verteilung und Bearbeitungen von Nachrichten, Mempool für das flüchtige Zwischenspeichern von Transaktionen, Persistenz für das finale Speichern von Blöcken und Transaktionen, sowie die Konsens-Komponenten für die Ermittlung der Reihenfolge der Transaktionen, deren Gültigkeit und die Abstimmung darüber mit anderen Knoten, erforderlich. Zur Validierung der Transaktionen wird eine außenstehende Geschäftslogik-Komponente benötigt, da nur sie über die semantischen Bedeutung der Transaktionen Bescheid weiß.

Um die bestmögliche Generalisierbarkeit und Wiederverwendung beziehungsweise Austausch von Komponenten zu ermöglichen, sollten diese eine möglichst lose Kopplung haben. Das heißt auch, dass die einzelnen Komponenten möglichst wenige, aber für den Prozess innerhalb des Frameworks essenzielle Schnittstellen anbieten müssen. Abbildung 6.2 zeigt einen Vorschlag der notwendigen Komponenten und deren Schnittstellen. Sowohl die Komponenten innerhalb des Frameworks (Konsens, Mempool, Netzwerk und Persi-

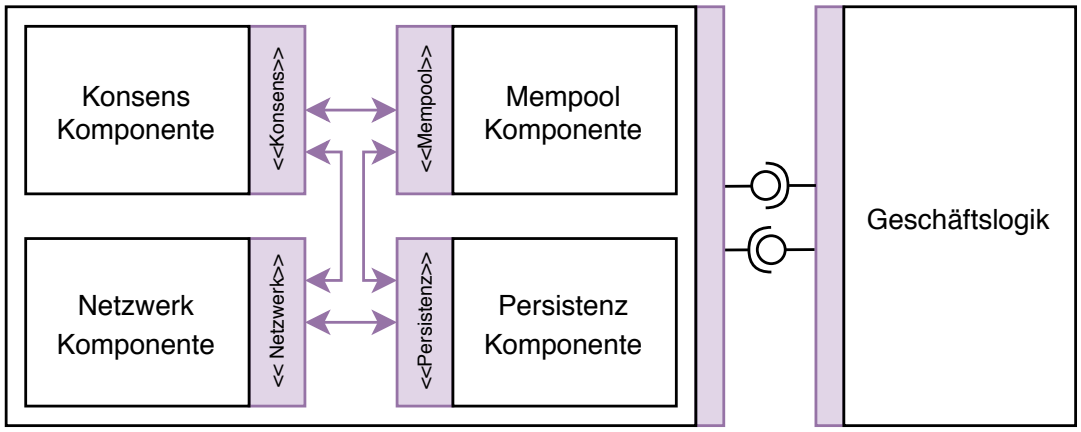

Abbildung 6.2.: Schnittstellenbasiertes Blockchainframework

stenz) als auch die ausgelagerte Geschäftslogik müssen Schnittstellen zur Interaktion anbieten. Die Schnittstelle zwischen Framework als Ganzes und der externen Geschäftslogik sollte auf ein Minimum reduziert sein und nicht alle internen Schnittstellen 1-zu-1 nach außen weitergeben, sondern nur die, die für den Betrieb notwendig sind, wie beispielsweise Validierung von Transaktionen oder Abfrage von Blöcken. Eine serviceorientierte bzw. dienstorientierte Architektur ermöglicht die Kapselung der Komponenten in wohldefinierte Dienste und Nutzungsschemas. Durch einen vordefinierten Service kann so auch die Logik einer Komponente durch eine andere ersetzt werden, beispielsweise ein Key-Value-Store durch eine dokumentenorientierte Datenbank.

6.2.2. Transaktionsphasen

Um möglichst alle Geschäftsprozesse zu unterstützen, muss innerhalb einer generalisierbaren Blockchain jede Transaktion mehrere Phasen bis zur finalen Gültigkeit, also Inklusion in einen Block, durchlaufen. Hierbei ist es erforderlich alle Phasen explizit zu benennen und in Abstimmung mit der Geschäftslogik, durch definierte Schnittstellen, auch zu durchlaufen.

Während das Framework eine Überprüfung der Syntax, beispielsweise maximale Byte-Länge der Transaktion, selbst übernehmen kann, sind die folgenden drei semantischen Phasen nur von der angeschlossen Geschäftslogik überprüfbar.

Atomar Eine Transaktion muss für sich selbst gültig sein. Validierung kann hier auf einzelne Bestandteile der Transaktion bezogen sein. Beispielsweise müssen für einen Brief sowohl Absender als auch Empfänger enthalten und deren Adressen gültig sein. Postleitzahlen können nach den entsprechenden Kriterien (5-stellig, nur Zahlen) zusätzlich validiert werden.

Historisch Eine Transaktion muss zusätzlich zu sich selbst auch in Bezug auf den aktuellen Zustand gültig sein, dies umfasst also auch alle vorhergehenden Transaktionen in finalen Blöcken. Beispielsweise kann von einem Konto keine Überweisung getätigt werden, wenn im vorhergehenden Block dieses bereits geleert wurde.

Intra-Block Die letzte Phase der Validierung schließt alle vorhergehenden ein. Eine Transaktion muss auf sich selbst bezogen, auf den bisherigen aktuellen Zustand und bezogen auf den neu entstehenden Zustand durch Anwendung der Transaktionen im selben Block gültig sein. Analog zum vorigen Beispiel, kann von keinem Konto eine Überweisung ausgehen, welches durch die vorige Transaktion innerhalb des Blockes bereits geleert wurde. Vor allem da diese sequentiell von der ersten bis zur letzten ausgeführt werden und schrittweise den Gesamtzustand des Systems verändern.

6.2.3. Konsensalgorithmen für Geschäftsprozesse

Der Konsensalgorithmus ist das Herzstück einer jeden Blockchain, ohne welchen keine einheitlichen globalen Systemzustände erreicht werden können, besonders unter Einfluss von böswilligen Teilnehmern. Wie bereits in Ka-

pitel 3.3 dargestellt, gibt es diverse Konsensalgorithmen mit unterschiedlichen Funktionsweisen. Proof-of-Work-Algorithmen (PoW) schaffen Vertrauen durch das Aufwenden von enormer Rechenleistung. Im Gegensatz hierzu ermöglichen Proof-of-Stake-Algorithmen (PoS), dies ohne Aufwand, dafür wird aber eine Zugehörigkeit oder eine Werteinsatz benötigt. Für Geschäftsprozesse sind einige Algorithmen vorteilhafter als andere.

PoW-Algorithmen, egal in welcher Ausprägung, sind für Geschäftsprozesse ungeeignet. Zuerst liegt dies an der Ausgangsannahme, die in PoW-Algorithmen getroffen wurden. PoW nimmt an, dass alle Teilnehmer im Netzwerk unbekannt und nicht vertrauenswürdig sind. Um hier Vertrauen zu suggerieren, muss viel Rechenleistung aufgewendet werden. Wenn ein Knoten einen ungültigen Block aber mit korrektem Kryptorätsel ins Netzwerk sendet, würde er von anderen Knoten abgelehnt werden und der Knoten hätte Rechenleistung verschwendet. Das nächste Problem entsteht durch das Kryptorätsel selbst. Wenn das Rätsel zu schwer ist, vergrößert sich die Blockzeit, also die Zeit zwischen dem Publizieren von zwei Blöcken. Gerade die Blockzeit kann für echtzeitkritische Geschäftsprozesse von großer Relevanz sein. Wird das Kryptorätsel so konfiguriert, dass es leichter ist, verkürzt sich dementsprechend die Blockzeit. Allerdings erhöht sich damit auch die Wahrscheinlichkeit von unbeabsichtigten Forks. Eine häufige Entstehung von Forks ist wiederum auch nachteilig für Geschäftsprozesse, denn solange ein Fork besteht, gibt es mehrere konkurrierende globale Systemzustände. Diese genannten Gründe machen PoW-Algorithmen ungeeignet für den Einsatz in Geschäftsprozessen.

PoS-Algorithmen hingegen benötigen keine enormen Mengen an Rechenleistung um das Vertrauen herzustellen. PoS beruht auf dem Setzen von Währung, beziehungsweise dem Wetten auf den nächsten richtigen Block. Ebenso wie PoW beruhen viele der PoS-Algorithmen auf der Annahme eines unbekannten Netzes, das heißt, es werden auch hier Ansätze verfolgt, die das Propagieren von falschen Blöcken unterbinden soll. In diesem Fall durch den Verlust von eingesetzter Währung. Das grundsätzliche Problem der längeren Blockzeiten bleibt auch hier bestehen. Zusätzlich kann es je nach PoS-

Algorithmus noch dazu kommen, dass Forks unendlich weiterbestehen, da sich Minter nicht final auf einen der beiden Äste festlegen und weiter auf beide Äste des Forks wetten.

Als letzte Varianten der Konsensalgorithmen bleiben noch Crash-Fault-Tolerant (CFT) oder Byzantine-Fault-Tolerant (BFT) Algorithmen übrig. Während CFT-Algorithmen tendenziell geeignet sind für Geschäftsprozesse, da im Netzwerk bekannte und teilweise vertrauenswürdige Teilnehmer sind, eignen sie sich schlecht für Prozesse, in denen Teilnehmer eine andere Wahrheit behaupten können, sich also mutwillig fehl verhalten, wofür es in bestimmten Prozessen durchaus Anreize gibt. CFT-Algorithmen tolerieren keine Byzantinischen Knoten und können aus diesem Grund in außerbetrieblichen Szenarien keinen Einsatz finden. Für innerbetriebliche Prozesse eignen sie sich hingegen.

BFT-basierte Algorithmen, wie der Tendermint-Algorithmus beispielsweise, tolerieren bis zu $\frac{1}{3}$ Byzantinischer Teilnehmer. Dieser ist eine Mischung aus BFT und PoS-Algorithmus, denn er erweitert den PBFT-Algorithmus um einen Stake. Dieser Stake wird aber nicht zur Wertvermehrung oder -minderung verwendet, sondern entscheidet wie oft ein Validator drankommt. In dem Round-Robin Verfahren hat ein Validator mit mehr Stake eine größere Anzahl an Runden, in denen er Proposer ist, als einer mit weniger Stake. Anstatt Stake zu verlieren bei Fehlverhalten, ignorieren alle anderen Teilnehmer sein Proposal und gehen in die nächste Runde mit einem anderen Proposer über. Bei zu häufigem Fehlverhalten kann über ein Ausschluss des Knotens abgestimmt werden. Die beiden weiteren Vorteile eines BFT-PoS Algorithmus sind die Geschwindigkeit und die sofortige Blockfinalität. Da keine komplizierten Kryptorätsel durch Brute-Force gelöst werden, kann der Algorithmus die Blockzeit vorgeben. Ein Proposer muss dazu nur eine bestimmte Zeit t warten, bevor er einen Block vorschlägt. Je kleiner das Netzwerk, desto schneller erhalten alle Teilnehmer den Vorschlag und können in den darauffolgenden 2 Runden darüber abstimmen. Weiterhin ist deterministisch festgelegt, welcher Validator in Höhe h und Runde r zum Proposer wird. So kann

es immer nur einen Vorschlag geben und es können keine Forks auftreten, was bedeutet, dass ein Block sofort final gültig ist.

Zusammenfassend kann hier gesagt werden, dass für einen Algorithmus mit mehreren Teilnehmer eines Konsortiums, die sich prinzipiell gegenseitig kennen und vertrauen, aber dennoch die Möglichkeit für mutwilliges Fehlverhalten besteht, nur ein BFT-PoS basierter Konsensalgorithmus zum Einsatz kommen kann.

6.2.4. Semantisch relevante Blockbildung

Für einige Prozesse ist eine First-In-First-Out basierte Reihenfolge innerhalb des Blockes ausreichend. Für Prozesse, in denen die Reihenfolge der Transaktionen von essenzieller Relevanz ist, besteht in FIFO-basierten Frameworks keine Möglichkeit diese einzuhalten. Entweder muss bereits bei der Entwicklung darauf geachtet werden, dass zwischen der Einstellung eben dieser Transaktionen genügend Zeit vergeht, damit sie auf jeden Fall in der richtigen Reihenfolge im Mempool respektive Block landen, oder es muss eine Möglichkeit bestehen, um bei der Validierung auch nachfolgende Transaktionen einzubeziehen, was nicht trivial ist.

In Prozessen, in denen kooperativ nach einem Optimum gesucht wird, beispielsweise in der Lastverteilung in dezentralisierten virtuellen Kraftwerken, sollten sogar, um unnötige Daten zu vermeiden, nicht-optimale Transaktionen aus dem Block rausgehalten und nur die optimale Lösung persistiert werden [Stübs u. a. 2019]. Hier stellt jede Transaktion einen Erzeugungsplan-Vektor dar. Über die Summe der Einzelkosten des Vektors können dessen Gesamtkosten bestimmt werden. Während das Erstellen der Fahrpläne ein Optimierungsproblem mit hohem Aufwand darstellt, lassen sich die Vektoren durch Summenbildung über ihre Kosten sehr leicht vergleichen und einen optimalen Fahrplan feststellen.

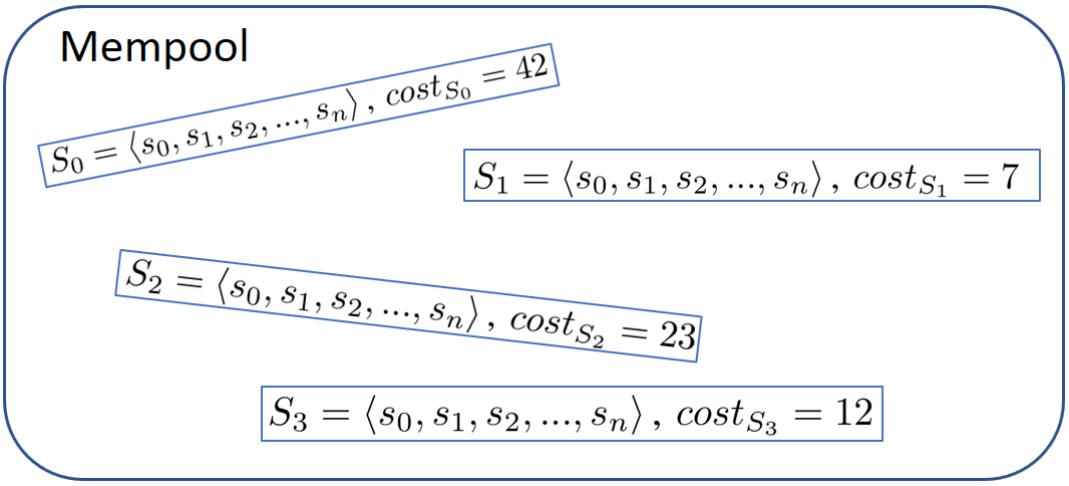

Abbildung 6.3.: Mempool mit vier Fahrplan-Vektor-Transaktionen

Abbildung 6.3 zeigt beispielhaft einen Mempool mit vier Fahrplan-Vektoren S_0 bis S_3 und deren Gesamtkosten. Würden in diesem Beispiel die Transaktionen nach dem FIFO Prinzip ausgewählt, wäre die erste Transaktion im Block S_0 und alle darauffolgenden Transaktionen ungültig, da bereits ein Fahrplan für diesen Slot existiert, obwohl dessen Kosten höher sind als die von S_1, S_2 und S_3. Die atomare Validierung garantiert zuerst, dass generell keine ungültigen Transaktionen in den Mempool aufgenommen werden, in diesem Beispiel durch Überprüfung der Fahrpläne und entstehenden Kosten. Weiterhin wird so garantiert, dass auch keine ungültigen Transaktionen im Netzwerk weiter verteilt werden. Da alle gutwilligen Knoten über dieselbe Kostenfunktion die Transaktionen validieren, wird der Blockvorschlag vom Proposer angenommen, da es tatsächlich die beste bekannte Lösung ist. Sollte eine Transaktion S_x mit $S_x < S_{proposal}$ im Mempool existieren, wird das Proposal abgelehnt.

Um bei diesem Beispiel und anderen Prozessen, die ebenfalls strikte Reihenfolgen erfordern, durch das Framework eine Reihenfolge abseits von FIFO zu garantieren, wird an dieser Stelle eine weitere semantische Anbindung an die Geschäftslogik benötigt. Fortlaufend muss die Geschäftslogikkomponente (GLK) in die Blockbildungsphase einbezogen werden. Dazu muss das

Framework einen Blockvorschlag über eine Schnittstelle an die GLK senden. Die GLK kann nun alle Transaktionen in eine dem Prozess begünstigende Reihenfolge bringen, oder diese auch temporär (zurückstellen in den Mempool) oder permanent aussortieren und wieder zurück an das Framework übermitteln.

Im vorgestellen Beispiel für Fahrplananmeldung und Lastverteilung würde der Vektor S_1 mit $cost = 7$ als aktuelles Optimum festgestellt und als einziger im Blockvorschlag verbleiben, während S_0, S_2, S_3 permanent aussortiert werden, da sie für diesen Prozess auch keinerlei weitere Relevanz haben. Im Konsensprozess stellen alle anderen teilnehmenden Knoten fest, dass S_1 die optimale Transaktion ist und stimmen dem Proposal zu.

6.3. Datenkorrekturverfahren

In vielen Geschäftsprozessen kommt es häufig dazu, dass Daten oder Verträge nachträglich korrigiert oder verbessert werden. Auch werden in regelmäßigen Intervallen Altdaten, die nicht mehr für Prozesse benötigt werden oder nicht mehr der Speicherungspflicht unterliegen, aus dem System gelöscht, um Speicherplatz freizugeben. In klassischen zentralisierten Diensten mit darunterliegender Nicht-Blockchain-Datenbank kann dies leicht vorgenommen werden, beispielsweise durch ein *Update-Statement* oder *Delete-Statement*, da es nur das Einverständnis der beteiligten Parteien benötigt und atomar auf der Datenbank ausgeführt werden kann. Innerhalb eines Blockchain-Systems ist das direkte Verändern aus zwei Gründen nicht trivial möglich. Zum einen bedarf jede Transaktion einer Zustimmung des kompletten Netzwerkes, entweder durch den aktuellen Miner oder in anderen Algorithmen von allen am Konsens partizipierenden Teilnehmern (Validatoren). Zum anderen beschränkt die Speichermethode der Blockchain grundlegend jedwede Veränderung von alten Daten durch Verkettung von alten mit neuen Datensätzen und Prüfsummen mittels Kryptografieverfahren. Jede Änderung an einer zurückliegenden Transaktion würde den eigenen Hash grundlegend

verändern und alle darauf aufbauenden Hashes (Root-Hash und Block-Hash) im selben Block und die der darauf folgenden Blöcke vollständig invalidieren und zu ungültigen Prüfsummen führen.

Während sich Systemzustände innerhalb der Applikation durch neue eingestellte Transaktionen verändern lassen, verbleiben die Altlasten dennoch innerhalb der stets wachsenden Blockchain-Historie. Da es für viele Geschäftsprozesse wichtig ist, Daten zu ändern und zu löschen und dabei den steigenden Speicherplatzbedarf der Blockchain nicht zu überreizen, wird nachfolgend ein Konzept für ein Datenkorrekturverfahren und anschließend ein Bereinigungsverfahren basierend auf *absichtlichem Forken* aufgezeigt. Ausgehend von einem BFT-PoS basierten Algorithmus wird nachfolgend eine Methode angedacht, wie die teilnehmenden Knoten absichtlich an einer früheren Blockhöhe einen Fork durchführen.

6.3.1. Absichtliches Forken

In Cryptocurrency Blockchains, wie Bitcoin, kommt es gelegentlich vor, dass das Netzwerk temporär divergiert. Diese sogenannten Temporary Forks sorgen kurzfristig für im Netzwerk unterschiedliche Systemzustände, wenn zwei Miner in Proof-of-Work Algorithmen (nahezu) gleichzeitig die Hash-Challenge des Blockes lösen und diesen im Netzwerk verteilen. Sobald einer der beiden „Äste“ länger wird, wechseln alle Knoten durch die Longest-Chain-Rule auf diesen über.

Hingegen können Konsensalgorithmen, die auf dem PBFT-Algorithmus basieren, nicht forken, da zu jedem Zeitpunkt für jede Blockhöhe und -runde nur von einem Proposer ein Blockvorschlag erstellt werden kann, alle anderen Blockvorschläge werden regelbasiert direkt abgewiesen. Dies sorgt einerseits für ständig konforme Systemzustände und sofortige Blockfinalität, was sie für viele Geschäftsprozesse prädestiniert.

Ausgehend von dem vorherigen Beispiel der Fahrplananmeldung und Lastverwaltung, bei dem nur der optimale Fahrplan in einen Block aufgenommen wird und andere sub-optimale Transaktionen verworfen werden, kann es natürlich dazu kommen, dass Teile des Netzwerkes diese optimale Transaktion gar nicht erhielten. In Peer-to-Peer Netzwerken kann das beispielsweise durch Paketverlust, Netzwerksegmentierung oder böswilliges Verhalten passieren.

Angenommen für einen Zeitpunkt P gibt es zwei konkurrierende Transaktionen T_{bad} und T_{good}, für die die Kostenfunktion $|T_{bad}| > |T_{good}|$ liefert, also T_{good} die bessere Lösung ist. Aus den vorher genannten Gründen können nun also zwei der folgenden Fälle eintreten, je nachdem ob die optimale Transaktion T_{good} beim aktuellen Proposer und den anderen Teilnehmern im Mempool enthalten ist. Wenn der Proposer nur T_{bad} und nicht T_{good} kennt, wird er zwangsweise T_{bad} als beste Lösung in den Blockvorschlag einschließen. Für diesen Blockvorschlag ergeben sich somit zwei Ausgänge, je nachdem welche Teilnehmer T_{good} im Mempool haben.

Für einen Blockvorschlag mit $B = \{..., T_{bad}, ...\}$ ergibt sich folglich:

> $> 2/3$ der Teilnehmer erhielten T_{good}. Das Proposal mit T_{bad} wird abgelehnt. Der nächste Proposer in der Reihenfolge kann einen neuen Vorschlag machen, bis der Block T_{good} enthält.
>
> $< 1/3$ der Teilnehmer erhielten T_{good}. Das Proposal wird angenommen, weil $+\frac{2}{3}$ nur T_{bad} kennen und dies für eine BFT-Mehrheit ausreicht. Hier wird die schlechtere Lösung im System persistiert ohne eine Veto-Möglichkeit.

Für den anderen Fall spielt dieses Problem keine Rolle. Sollte der Proposer T_{good} in den Blockvorschlag aufnehmen, ohne dass eine Mehrheit die Transaktion im Mempool hat, akzeptieren sie diesen dennoch, da die bessere Transaktion gleichzeitig mittels des Blockvorschlags im Netzwerk verbreitet und somit allen bekannt gemacht wird.

Eine Möglichkeit dem entgegenzuwirken wäre eine spezielle Widerspruchsnachricht in den Konsensalgorithmus zu integrieren. Diese würde die aktuelle Abstimmungsrunde im Konsens sofort beenden und gleichzeitig einen Änderungsvorschlag als Begründung für den Widerspruch darlegen. Dennoch ist auch diese Widerspruchsnachricht von denselben Problemen betroffen, wie die normalen Nachrichten, beispielsweise, dass nicht jeder Teilnehmer die Nachricht rechtzeitig bekommt und somit die Abstimmung über den nächsten Block bereits vollzogen wurde.

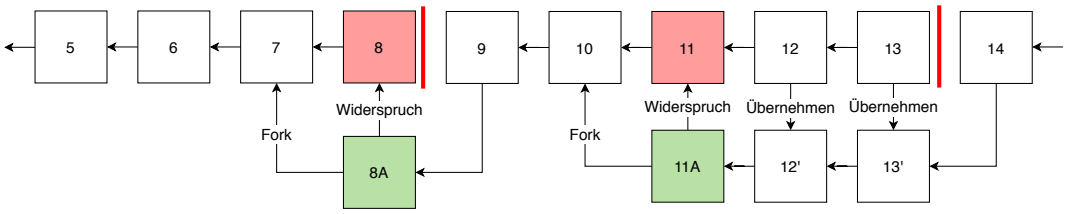

Abbildung 6.4.: Widerspruch von Blöcken

Mit Bedacht auf die Nachrichtenlaufzeiten und -übermittlungsverzögerungen ist die bessere Lösung, einen Widerspruch nach der Blockbildung nachträglich zu tätigen und diesen rückabzuwickeln, wie es beispielsweise bei Bitcoin in einem Fork passieren würde. Eine Rückabwicklung basiert demnach auf den zwei Schritten *Zurückrollen* und *Ersetzen*.

Über die Widerspruchsnachricht wird also ein neuer Block für eine bereits bestehende zurückliegende Höhe vorgeschlagen, der basierend auf einer Kostenfunktion über seinen Transaktionen ein besseres Ergebnis liefert, was durch die Teilnehmer leicht verifiziert werden kann. Angelehnt an die Longest-Chain-Rule ersetzen Teilnehmer nun den alten Block durch den neuen besseren Block. Über die normalen Konsensprozesse wird über Einigkeit und Gültigkeit des neuen Blockes abgestimmt und global in die Historie aufgenommen. Abbildung 6.4 zeigt beispielhaft, wie in einer Blockchain die Blöcke `8` und `11` durch Widersprüche mit `8A` und `11A` ersetzt werden. Da sich die Hashwerte der Folgeblöcke `12` und `13` durch die Ersetzung ändern, müssen hier die Hashes neu berechnet werden und wiederum den Konsens

zur Abstimmung durchlaufen. Algorithmus 1 zeigt den Ablauf zur Erstellung eines Widerspruches.

Algorithmus 1 Widerspruch

```
1:  BC ← Blockchain[Block_0, ..., Block_n]
2:
3:  function ERSTELLE_WIDERSPRUCH(height, tx_bad, tx_good)
4:      W ← [ ]
5:      B ← BC[height]
6:      B[tx_bad] ← tx_good
7:      W[height] ← B
8:      for i ← height + 1, sizeof(BC) do
9:          W[i] ← BC[i]
10:     end for
11:     berechneHashes(W, height, sizeof(BC))
12:     sendeNachricht(W)
13: end function
```

Sobald also ein Teilnehmer eine schlechte Transaktion (ausgehend vom vorherigen Beispiel) in einem Block B_a findet, verteilt er eine Widerspruchsnachricht im Netzwerk, die den neuen Block mit der besseren Transaktion T_{good} enthält.

Sobald der aktuelle Proposer einen Widerspruch erhält, prüft er diesen und schlägt in seiner Runde anstelle des ursprünglichen Blockproposals den Widerspruch vor. Diese Spezialrunde ersetzt also einen vorherigen Block anstelle einen neuen an die bisherige Historie anzuhängen. Der Widerspruchs-Block darf sich in seiner Gesamtheit, mit Ausnahme der zu ersetzenden Transaktion und daraus resultierenden Hashwertänderungen, nicht von dem Ursprungsblock unterscheiden.

Die Ersetzung des aktuell letzten Blockes ist trivial möglich. Sollte der Widerspruch zu einer Transaktion mehrere Blöcke zurückliegen, müssen diese Blöcke unverändert mit in die Widerspruchsnachricht übernommen werden, offensichtlich mit angepassten Hashwerten. Algorithmus 2 zeigt das Erhalten eines gültigen Widerspruchs. Um darüber Buch zu führen, was und wann ersetzt wurde, sollten *Archivknoten* auch die ersetzten Blöcke vorhalten.

Algorithmus 2 Erhalt eines Widerspruches

```
1:  BC ← Blockchain[Block_0, ..., Block_n]
2:
3:  function BEI_WIDERSPRUCH(W, height)
4:      archivKnoten ← ist_archiv_knoten?
5:      if cost(BC[height]) > cost(W[height]) then //Widerspruch ist besser?
6:          verifiziereHashes(W, height, sizeof(BC))
7:          for i ← height, sizeof(W) do
8:              BC ← W[i]
9:              if archiveNode then
10:                 OB ← BC[B_height] //Originalblock archivieren
11:                 Archive[OB_height, OB_hash] ← OB
12:             end if
13:             BC[B_height] ← B
14:         end for
15:         benachrichtigeApplikation(tx_bad,tx_good)
16:     end if
17: end function
```

Um den Konsens zu beschleunigen, kann bei einem Widerspruch, der schon einige Blöcke zurückliegt, direkt ein Widerspruch mehrere Blöcken inkludieren und als Proposal verwendet werden. Da sich nur der erste Block im Widerspruch inhaltlich verändert hat, muss auch nur dieser erneut auf semantische Gültigkeit geprüft werden, bei den anderen reicht es aus, diese auf Gültigkeit der Hashwerte zu prüfen, angenommen keine der späteren Transaktionen haben Abhängigkeiten zu Tx_{bad}. Bei Gültigkeit werden alle Blöcke, wie bisher im Konsens auch, einzeln signiert, aber als eine Nachricht weitergeschickt. So wird nur eine Runde benötigt, um einen Block zu ersetzen und die Hashes der darauf folgenden anzupassen.

6.4. Abhängigkeiten in Blockchain-Transaktionen

Bei Blockchains handelt es sich primär um eine neue technische Lösung für replizierte Zustandsautomaten. Transaktionen werden verwendet, um den

Zustand der Blockchain, beziehungsweise deren verknüpfter Anwendung, zu verändern. Der tatsächliche Zustand wird meist nicht in der Historie festgehalten, stattdessen beinhalten die Blöcke die einzelnen Übergänge zwischen den Zuständen.

Durch ein wiederholtes Ausführen aller Blöcke der Historie kann ein Knoten sich wieder mit dem Netzwerk synchronisieren. Um einen neuen Teilnehmer ins Netzwerk aufzunehmen, muss dieser alle Blöcke und Transaktionen der Reihenfolge nach ausführen, um den aktuellen Systemzustand herzustellen.

Mit zunehmender Laufzeit einer Blockchain kann die Größe der Historie mit potentiell Millionen Transaktionen für eine deutlich höhere Dauer bei der Neu- oder Wiedersynchronisierung führen. Nicht nur das reine Übertragen Hunderter von Gigabytes (ca. 210 GB bei Bitcoin und ca. 180GB für Ethereum Full-Nodes, aber 2,3TB für Archivknoten) in einem dezentralen Peer-to-Peer Netzwerk, sondern auch das erneute Ausführen der Transaktionen benötigt Zeit. Tendenziell kann ein Knoten vieles der Historie später verwerfen, nachdem er diese verifiziert hat, dennoch müssen sie erst einmal in Gänze übertragen werden.

Um den Prozess der Neusynchronisierung zwischen Teilnehmern innerhalb des Netzwerkes zu beschleunigen, müssen die zu übertragenen Daten minimiert werden. In Bitcoin gibt es beispielsweise keinerlei Verfahren, um dies zu erreichen, da es keine Zustände im UTXO-basierten System gibt, muss die ganze Historie synchronisiert werden. Es gibt durch Dritte betriebene Snapshot Verfahren, die etwa alle 300 Blöcke einen Snapshot über das offene UTXO-Set an Transaktionen bilden. Diesen muss aber blind vertraut werden.

Im Account-basierten Transaktionsmodell von Ethereum müssen je nach Anwendungsfall nicht alle Blöcke komplett übertragen werden. Light-Nodes beziehen und speichern nur die Metainformationen der Blöcke, wie Hashes und Zeitstempel. Benötigte Daten zu Accounts und Zuständen werden bei Bedarf von Full-Nodes oder Archive-Nodes bezogen und mittels der bereits über-

tragenen Hashes verifiziert. Full-Nodes wiederum beziehen bei Neueintritt direkt die Metainformationen und die Zustandsdatenbank der Accounts bis zu einer bestimmten Blockhöhe. Beim Erreichen der gewählten Synchronisierungshöhe wechselt der Full-Node in den normalen Betriebsmodus und speichert wieder komplette Blöcke. Archiv-Knoten speichern hingegen immer alle Informationen, Transaktionen und Blöcke und benötigen somit am meisten Speicherplatz.

Während Light-Nodes für IoT-basierte Netzwerke die beste Möglichkeit an einer Blockchain teilzunehmen bieten, eignen diese sich nur bedingt für geschäftsbasierte Use Cases. Auch die Möglichkeit schnell auf ältere Daten zurückzugreifen, wird stark eingeschränkt. Wenn alle Teilnehmer innerhalb eines Konsortiums also Full- beziehungsweise Archive-Nodes betreiben, bleibt als letzte Option, um das Synchronisieren zu beschleunigen nur noch die Integration von „Zustands-Snapshots“ als Zwischenstände oder das Bereinigen der Blockchain-Historie selbst, um dessen Größe zu reduzieren.

Um eine generelle Bereinigung der Blockchain-Historie, unabhängig von der Anwendungslogik, zu ermöglichen, sind verschiedene Einflussfaktoren von Bedeutung. Zuallererst muss betrachtet werden, in welcher Form Transaktionen zueinander in Relation stehen und wie die Zustände daraus generiert oder beeinflusst werden. Dies wird nachfolgend dargestellt.

6.4.1. Transaktionen in Kryptowährungen

Durch den Anwendungsfall bedingt, haben Blockchains für Kryptowährungen nur einen einzigen Use Case: der Transfer von Werten von einem Besitzer zu einem anderen. Darum sind die Transaktionen in Kryptowährungen meist auch stark limitiert, wie sie den Systemzustand verändern und auf welchen vorherigen Zuständen sie aufbauen oder davon abhängen. In Bitcoins UTXO-Transaktionsmodell gibt es auch nur zwei Arten von Transaktionen, die den Zustand verändern.

Erzeugungstransaktionen (engl. *generation* oder *coinbase* transaction) werden verwendet, um einen neuen Wert einzuführen. Sie hängen dadurch auch von keinen anderen Transaktionen ab. Sie erzeugen Werte aus dem Nichts und fügen dem Markt neue Münzen hinzu. Die Erzeugung neuer Werte ist meist auf wenige Knoten beschränkt oder wird als Belohnung in Mining- oder Minting-basierten Konsensalgorithmen ausgegeben.

Eine Erzeugungstransaktion hat also keine Eingangswerte und mindestens einen Ausgangswert (0-zu-1, bzw. 0-zu-n).

Transfertransaktionen werden verwendet, um Werte zwischen einem oder mehreren Besitzern zu einem oder mehreren neuen Besitzern zu übertragen. Sie verändern also den Zustand von vorher erzeugten Werten und hängen damit also immer von mindestens einer vorhergehenden Elterntransaktion ab, da nur Erzeugungstransaktionen für sich allein stehen können.

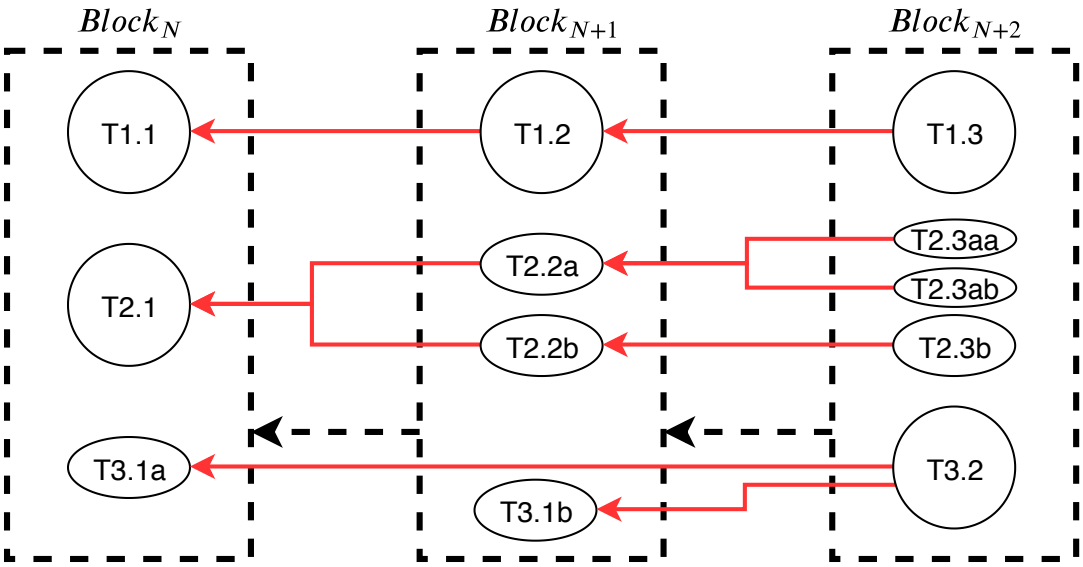

Abbildung 6.5.: Abhängigkeiten zwischen Transaktionen

Betrachtet man nun die möglichen Beziehungen zwischen den Transaktionen, lassen sich diese durch die folgenden Fälle charakterisieren:

- 1-zu-1, eine Transaktion hängt nur von einer Elterntransaktion ab.

- 1-zu-n, mehrere Transaktionen hängen von einer Elterntransaktion ab.
- n-zu-1, eine Transaktion hängt nur von mehreren Elterntransaktionen ab.
- n-zu-n, mehrere Transaktionen hängen von mehreren Elterntransaktionen ab.

Transaktionssequenzen bestehen also immer aus einer Erzeugungstransaktion, worauf eine oder mehrere Transfertransaktionen folgen, welche wiederum auf einem oder mehreren Eingängen beruhen und einen oder mehrere neue Ausgänge erzeugen. Abbildung 6.5 veranschaulicht diese Zusammenhänge nochmal grafisch mit drei Beispielen.

Sobald eine Transaktion auf einer anderen beruht, kann man vereinfacht sagen, dass sie den vorherigen Zustand verbraucht oder zerstört, was ihre Vorgänger in Bezug auf die Zustandsreplikation nun irrelevant macht.

6.4.2. Transaktionen in Geschäftsprozessen

Wenn man sich die Lebensspanne einer Kryptomünze in der Blockchain anschaut, kann man leicht feststellen, dass jede Transaktion eine unbegrenzte Sequenz von Wertübertragungen darstellt, sofern eine Münze nicht bewusst *verbrannt* wird und somit die Sequenz beendet. In Geschäftsprozessen sind unendliche Sequenzen nicht unbedingt realistisch und somit lassen sich hier noch zwei weitere Transaktionsarten erkennen.

Als **atomare Transaktionen** bezeichnen wir Transaktionen, die nicht von einem vorigen Zustand abhängen und außerdem keine Kind-Transaktionen haben können (∅-Beziehung). Konkrete Beispiele für diese Transaktionstypen wären das Speichern von Geburtsurkunden ([Kshetri 2017]) oder persönlichen Identifikationsnummern ([Manohar und Briggs 2018]), welche normalerweise nur einmal erstellt und dann nie wieder verändert oder invalidiert werden.

Weiterhin gibt es in Geschäftsprozessen auch Endzustände, welche wir durch eine **Abschluss-Transaktion** kennzeichnen. Eine Transaktion kann also eine Sequenz mit einem bekannten Ende darstellen (1-zu-∅ oder n-zu-∅), wenn der Geschäftsprozess auf natürliche Weise durch ein bestimmtes Ereignis beendet wird. Beispiele hierfür sind Lieferketten ([Kim und Laskowski 2018]) oder diverse Marktplatzausprägungen ([Serban u. a. 2008]). In einem Lieferketten-Szenario werden Assets erzeugt, dann x-mal transferiert, mit anderen Assets kombiniert oder weiterverarbeitet, bevor es dann letztendlich vom Endkunden „verbraucht“ wird. Wenn der physische Zustand des Assets (off-chain) nicht mehr existiert, ist sein digitaler Zustand (on-chain) vertretbarerweise auch nicht mehr von Relevanz und könnte entfernt werden.

Nach der Feststellung der Transaktionsabhängigkeiten, ist ein weiterer notwendiger Schritt zur Bereinigung der Blockchain-Historie die eigentliche Minimierung der Zustandsübergänge, die durch Transaktionen ausgelöst werden, was im nächsten Abschnitt folgt.

6.5. Wiederkehrende Blockminimierung

Mit dem Ziel einer generellen Herangehensweise, die Blockchainhistorie und somit auch den benötigten Speicherplatz zu minimieren, muss das Transaktionsmodell einer generalisierbaren Blockchain angepasst werden. Damit das Blockchain-Framework dies unabhängig von der angeschlossenen Anwendungslogik tätigen kann, muss spezifisches semantisches Wissen über die Transaktionen bekannt sein, um die Beziehungen zwischen den Transaktionen in Bezug auf den Anwendungszustand zu erkennen.

Für Kryptowährungs-Blockchains, wie Bitcoin, ist dies vergleichsweise trivial, da die Anwendungslogik und die Blockchain-Logik bereits innerhalb eines einzelnen Frameworks miteinander verbunden sind. Darum hat der Blockchain-Teil der Software bereits Zugriff auf die semantischen Informationen der Transaktionen. Anhand der Semantik der Transaktionen können

leicht Beziehungen und Abhängigkeiten zwischen den Transaktionen extrahiert werden und ein Abhängigkeitsgraph erstellt werden. Eine Zusammenführung der Transaktionen, bei gleichbleibendem Anwendungszustand kann hier durch das Aktualisieren der Kontostände durchgeführt werden, wie es bereits in *Sidecoin* ([Krug und Peterson 2015]), *Iota* ([Popov 2016]) oder der *Mini-Blockchain* ([Bruce 2014]) erfolgt.

Dieses Prinzip kann jedoch nicht auf generalisierte Blockchain-Frameworks, wie Hyperledger oder Tendermint, übertragen werden. Diese Frameworks behandeln Transaktionen wie eine Black Box. Jede Funktion des Frameworks, die semantisches Wissen über die Transaktionsinhalte benötigt, wird über eine API an die angeschlossene Anwendung oder Business-Plugin delegiert. Deshalb sind bereits bestehende Abhängigkeiten innerhalb der Transaktionssemantik für das Framework unbekannt.

Um dem Blockchain-Framework die Möglichkeit zu geben, Abhängigkeiten zu erkennen, vor allem ohne semantisches Anwendungswissen der Transaktionsinhalte, müssen Transaktionen explizit Abhängigkeiten zu anderen Transaktionen ausweisen. Dadurch kann ein Framework ohne die Anwendungsschicht bereits unabhängig Abhängigkeitsgraphen erstellen und erst im darauffolgenden Schritt durch die Anwendung eine Zusammenführung der Transaktionen durchführen lassen. Das Ziel der Zusammenführung ist eine geringere Anzahl an Transaktionen zu erstellen, während die Ausführung dieser zum selben Anwendungszustand führt. Nach der Zusammenführung ist es wiederum die Aufgabe des Blockchain-Frameworks, diese Änderungen mittels des vorher vorgestellten Widerspruchverfahrens in Ersatz-Blöcke zu integrieren.

6.5.1. Explizite Transaktionsabhängigkeiten

Transaktionen in UTXO-basierten Blockchains (z.B.: Bitcoin) haben logischerweise bereits Abhängigkeiten integriert. Jede Transaktion beinhaltet als Eingangswerte Abhängigkeiten zu vorhergehenden Ausgangswerten anderer

Transaktionen. Für generalisierbare Blockchains müssen die Transaktionen also explizit Abhängigkeiten kenntlich machen, damit das Framework eigenständig mit diesen arbeiten kann. Die folgende Definition 6.1 stellt ein Standard-Transaktionsmodell für Blockchains dar:

$$\begin{aligned} Tx &= data \\ Tx_{hash} &= hash(data) \end{aligned} \tag{6.1}$$

Um dieses um explizite Abhängigkeiten D und weitere Kontextinformationen C zu ergänzen, schlagen wir folgende Definition 6.2 für eine Transaktion Tx vor:

$$\begin{aligned} Tx &= data, D, C \\ Tx_{hash} &= hash(data, D, C) \\ D &\subsetneq \{Tx_{hash_1}, \ldots, Tx_{hash_n}\} \\ C &\epsilon \{\varnothing, sequence, sequence\_end, \ldots\} \end{aligned} \tag{6.2}$$

Durch das Hinzufügen von Abhängigkeiten und Kontextinformationen, kann bereits zusätzlicher Aufwand für die Anwendungsschicht in das Framework selbst verlagert werden, da Transaktionen nun bereits in Abhängigkeitsgraphen gruppiert werden können. Weitergehend können durch Kontextbeschreibungen in C bereits Transaktionen kenntlich gemacht werden, die zu einer längeren Sequenz (`sequence`) gehören oder eine Sequenz komplett beenden (`sequence_end`). Einerseits beschleunigt dies die Suche nach weiteren Transaktionen, die zum Abhängigkeitsgraphen gehören, andererseits könnten beendete Sequenzen vorbeugend vom Framework gelöscht werden ohne Bestätigung von der Anwendungsschicht, wenn der Geschäftsprozess dies zulässt.

6.5.2. Sammeln und Zusammenführen von Transaktionen

Damit die angeschlossene Anwendung das Zusammenführen von Transaktionen in neue verkleinerte Zustandstransitionen übernehmen kann, müssen vorher vom Blockchain-Framework alle voneinander abhängigen Transaktionen gesammelt werden. Diesen Prozess bezeichnen wir als Anlehnung an das „Squashing“ aus dem Versionskontrollsystem *git* ebenfalls als *squash* (dt. zusammenquetschen), bei dem mehrere sequentielle Änderungen am Quelltext zu einer einzigen zusammengeführt wird.

Weil innerhalb der Blockstruktur der Blockchain bereits ein leicht traversierbarer Graph an Transaktionen vorliegt, mit Blöcken die auf andere Blöcke verweisen und expliziten Verweisen zwischen den Transaktionen und ihren Elterntransaktionen, können diese nun zu eigenständigen Graphen zusammengefasst werden. Hierzu eignen sich beispielsweise die Tiefen- und die Breitensuche, um Abhängigkeitsgraphen aus Transaktionen aufzubauen.

Ausgehend vom letzten Block mit der Blockhöhe h_{max}, müssen alle Transaktionen gesammelt werden, die eine oder mehrere Abhängigkeiten haben. Jede dieser Transaktionen wird zusammen mit ihren Geschwistern und jeweiligen Elterntransaktionen in einen Abhängigkeitsgraphen G gefasst, wie in Abbildung 6.6 dargestellt. Dies kann nun ohne Zutun durch die Anwendungsschicht erfolgen, da kein semantisches Wissen über die Transaktionsinhalte mehr nötig ist und die Abhängigkeiten explizit deklariert sind.

Da ein generalisierbares Blockchain-Framework kein semantisches Wissen über sowohl eine einzelne Transaktion als auch einen ganzen Graph an Transaktionen hat, muss der nächste Schritt des Zusammenführens (squash) von der Anwendungsschicht übernommen werden. Im zweiten Schritt muss also der Graph G über die dafür vorgesehene Schnittstelle (API) an die Anwendung weitergegeben werden. Diese muss nun das Zusammenführen vornehmen, indem sie den Graph G in einen neuen Graph G' überführt, der exakt

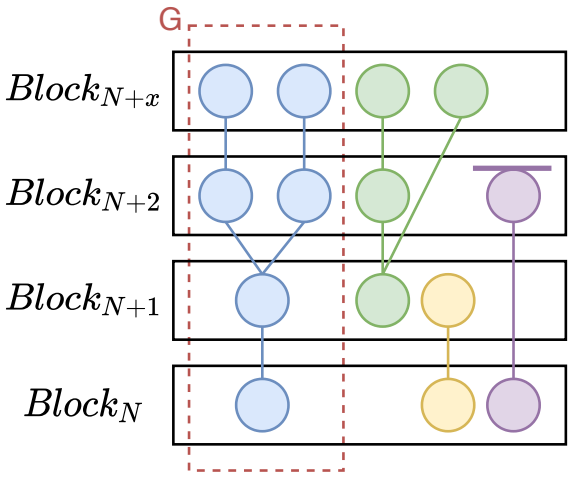

Abbildung 6.6.: Zusammenfassen von Transaktionen in einen Abhängigkeitsgraphen

demselben Anwendungszustand entspricht. Das Ziel muss sein, einen neuen Graphen zu erstellen für den folgende Eigenschaften gelten:

$G' \neq G$, Graphen sind distinkt

$|G'| < |G|$, Graph G' hat weniger Transaktionen

$S_{G'} = S_G$, Die Zustände der Graphen S sind gleich

Zusammengeführte Graphen werden über die Schnittstelle wieder zum Framework übertragen, was in Abbildung 6.7 veranschaulicht wird.

Das Zusammenführen von Transaktionen in einer Kryptowährung kann beispielsweise durch eine Aufsummierung aller kontorelevanter Transaktionen erfolgen. Der neue Kontostand wird dann als eine *Erzeugungstransaktion* wieder dem Framework übergeben. In anderen Use Cases muss der Squash-Prozess bedingt durch andere Semantik daraufhin angepasst werden.

Um die Vorteile des Squash-Prozesses zu maximieren, ist es von Vorteil, wenn jeder neue Graph G' nur eine einzige Transaktion enthält, ansonsten sollte immer $|G'| < |G|$ gelten. Als Spezialfall kann die Anwendung einen $\varnothing$-Graph zurückgeben, um zu kennzeichnen, dass der Zustand aus G für den

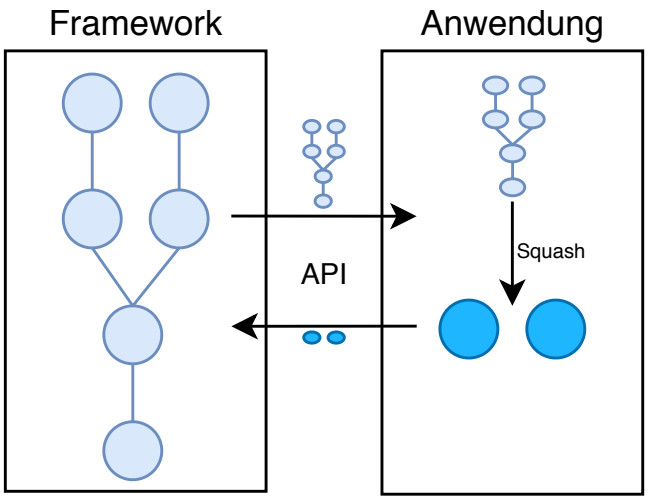

Abbildung 6.7.: Zusammenführung der Transaktionen

Geschäftsprozess nicht mehr von Relevanz ist und alle Transaktionen gefahrlos komplett aus der Historie gelöscht werden können.

Als letzter Schritt im gesamten Prozess bleibt nun die erneute Integration in das Block-Konstrukt der Blockchain übrig.

6.5.3. Neuerstellung von Blöcken

Nachdem die Anwendung die zusammengeführten Graphen wieder an das Framework zurückgegeben hat, muss dieses nun mit den Transaktionen neue Blöcke erstellen. Diese Ersetzungsblöcke sollten dabei denselben Regeln folgen, wie auch normale Blöcke. Sollte es Größenlimits der Blöcke oder eine Beschränkung der Anzahl an Transaktionen geben, müssen diese natürlich befolgt werden, was dazu führen kann, dass die Transaktionen über mehrere neue Blöcke verteilt werden müssen.

Die neuen Ersetzungsblöcke sollten dabei von unten nach oben sequenziell erstellt werden, so dass zusammengeführte Graphen in der Reihenfolge in den Blöcken vorkommen, in der auch die Ursprungsgraphen in den Original-

blöcken enthalten waren. Ein Graph, der aus einer Sequenz entstand, die in Block 5 begann, wird also vor einer Sequenz aus Block 12 aufgenommen. Unveränderte Graphen oder Einzeltransaktionen werden in ihrer Ursprungsform neben den neuen in derselben Weise integriert. Weiterhin werden ∅-Graphen, also Graphen die obsolete Zustände darstellen, ignoriert und nicht weiter behandelt oder inkludiert. Ein Beispiel für diesen Prozess ist in Abbildung 6.8 dargestellt. Hier wird ausgehend von vier distinkten Graphen eine Zusammenführung vorgenommen mit dem Spezialfall, dass der violett gefärbte Graph komplett gestrichen wird und für den gelben Graphen keine Zusammenführung möglich ist.

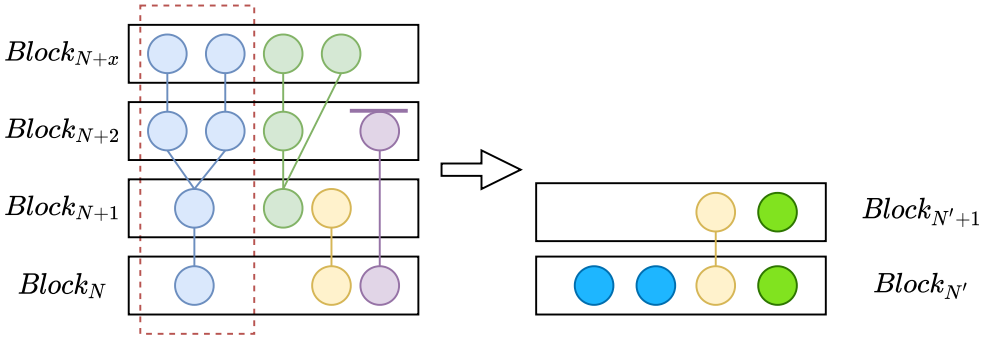

Abbildung 6.8.: Neuintegration von Transaktionen

Nachdem nun aus den zusammengeführten Graphen vom Framework neue Blöcke als Ersatzblöcke gebildet wurden, müssen diese wieder im Netzwerk verteilt und durch die Validatorknoten signiert und zusätzlich neu über deren Gültigkeit innerhalb des Konsensalgorithmus abgestimmt werden.

Die Verifikation, dass die neue Blockchainhistorie denselben Zustand wie die Originalhistorie hat, kann trivial ermöglicht werden. Dazu sollte ein Anwendungshash (*AppHash*) in jeden Block aufgenommen werden, der eine Prüfsumme über den aktuellen Zustand der Anwendung bereitstellt. Dieser AppHash muss von der Anwendung selbst bereitgestellt werden und muss in jedem Blockheader mitgespeichert werden. Wenn der AppHash nun korrekt den aktuellen Zustand widerspiegelt, dann ist der Vergleich zwischen Originalzustand und dem Zustand aus den zusammengeführten Transaktio-

nen und den Ersatzblöcken lediglich der Vergleich der beiden AppHashes und somit effizient und schnell überprüfbar. Hierdurch ist es essenziell, dass der AppHash so gebildet wird, dass er akkurat den echten Zustand beinhaltet.

Während die Neuerstellung von Blöcken, unter Rücksicht auf die im Algorithmus vorgegebenen Größen, wie maximale Anzahl an Transaktionen, relativ trivial umsetzbar ist, ist der richtige Zeitpunkt hierfür nicht leicht ermittelbar und hängt stark vom Geschäftsprozess ab. Potenzielle Zeitpunkte können unter anderem alle n-Blöcke, jeden Montag, alle m Megabytes an Blöcken oder nach speziellen Ereignissen sein.

Weil für passende Zeitpunkte semantisches Wissen notwendig ist, kann das Framework den Prozess nicht eigenständig starten, sondern muss von der Anwendung dazu aufgefordert werden oder Regeln mitgeteilt bekommen, da nur sie den besten Moment für den Geschäftsprozess kennt. Auch müssen alle Teilnehmer im Netzwerk am Prozess beteiligt werden, da es je nach Konsensalgorithmus sein kann, dass der Prozess des Zusammenführens und der Neuverteilung der Blöcke länger dauert als die normalen Konsensrunden für neue Blöcke. So würden neue Blöcke schneller angehängt werden und der Squash muss ständig ohne Ergebnis wiederholt werden.

In einem BFT-basierten Konsens könnten hierzu spezielle Squash-Transaktionen im Netzwerk verteilt und darüber abgestimmt werden, sodass bei einer $+\frac{2}{3}$ Mehrheit der normale Konsens für eine vereinbarte Zeit unterbrochen wird, um das zuvor skizzierte Katz-und-Maus Problem zu umgehen und einen vollständigen Squash durchzuführen, bevor der Konsens wieder aufgenommen wird.

6.5.4. Wiedereinführung von Blöcken

Die neuerstellten durch den Squashing-Prozess erzeugten Blöcke müssen als letzten Schritt noch in die aktuelle Blockchainhistorie reintegriert werden

und dabei die alten Blöcke in situ ersetzen. Dazu eignet sich die absichtliche Erzeugung eines Forks, wie er bereits im vorangehenden Kapitel 6.3.1 für die Ersetzung von einzelnen Transaktionen vorgestellt wurde und in Anlehnung an die *Longest-Chain-Rule* aus PoW-Konsens basierten Blockchains. Während bei BFT-PoS Algorithmen niemals, mit Ausnahme von böswilligem Handeln, Forks entstehen, passiert dies bei PoW häufiger und ist Teil des normalen Prozesses. Durch ein absichtliches Forken, durch die Konsensteilnehmer bestätigt, können in BFT-PoS Blöcke und ganze Ketten der Historie durch andere ersetzt werden. Um Teilnehmern zu erlauben, über das absichtliche Forken abzustimmen, muss vorerst über eine geeignete Kostenfunktion festgestellt werden, dass dies überhaupt zielführend ist. Während in PoW die Kostenfunktion für die Longest-Chain Rule nur die Kettenlänge ($|\boxed{\texttt{A}}| > |\boxed{\texttt{B}}|$) vergleicht, müssen für diesen Prozess die Ziele der Graphen-Reduktion aus Kapitel 6.5.2 übernommen und eingehalten werden. Für zwei Blöcke beziehungsweise Ketten $\boxed{\texttt{A}}$ und $\boxed{\texttt{B}}$ muss somit gelten, dass die Anzahl der Transaktionen $|Tx|$ bei $\boxed{\texttt{A}}$ geringer ist als bei $\boxed{\texttt{B}}$, aber die Anwendungszustände beider identisch sind, wie folgend dargestellt:

$$\begin{aligned} |Tx_A| &< |Tx_B| \\ AppState_A &= AppState_B \end{aligned} \tag{6.3}$$

Die Anzahl der Transaktionen pro Block oder Kette lässt sich durch die Summierung ihrer Transaktionen feststellen. Für das Vergleichen des Anwendungszustandes kann der *AppHash* aus den Block-Metadaten des höchsten Blockes der beiden Ketten verglichen werden und ist somit trivial umsetzbar. Sollte die Anzahl an Transaktionen gleich, aber die Menge an Blöcken in der neuen Kette geringer sein, ist fraglich, ob eine Ersetzung notwendig ist, da die zu erwartende Reduktion in der Speichergröße nur auf den Metadaten der Blöcke basiert und vergleichsweise gering ist.

Andernfalls, wenn der Squashing-Prozess zu keinem gültigen Ergebnis führte, oder nicht in dem vereinbarten Zeitfenster abgeschlossen werden konnte,

sollte der Konsensalgorithmus wieder aufgenommen und normal weitergeführt werden. Um den Prozess weniger anfällig für beispielsweise *Denial-of-Service* Angriffe zu machen, sollten die Initiatoren des Squashings auf einen kleineren Teil der Teilnehmer beschränkt sein. Dazu könnten die Validatoren, die sowieso in BFT-PoS eine vertrauensvollere Rolle innehaben, den Squashing-Prozess durch eine signierte Transaktion anstoßen. Hierdurch wird einerseits für Vertrauen gesorgt, andererseits können Validatoren bei Fehlverhalten direkt aus dem Konsens entfernt werden.

Zusammenfassung

Für die Umsetzung von Geschäftsprozessen ist es unabdingbar, eine Blockchain-Technologie zu verwenden, welche von vornherein für die Unterstützung von Geschäftsprozessen entworfen wurde. Bisherige Technologien, wie Bitcoin oder Ethereum, sind in ihrer Funktionsweise eingeschränkt, da ihre Anwendungsszenarien hauptsächlich der Werttransfer von digitalen Währungen sind. Im Falle von Ethereum wird weitergehend auch die Möglichkeit für die Ausführung von Geschäftsprozessen durch Smart Contracts geschaffen, welche jedoch auf Grund von Laufzeit- und Funktionsbeschränkungen nicht alle Anforderungen an Geschäftsprozesse erfüllen können.

Daher wurde basierend auf den bisherigen Erkenntnissen der vorangehenden Kapitel ein Konzept für ein generalisiertes Blockchain-Framework entwickelt, um jegliche Geschäftsprozesse damit umsetzen zu können. Wohldefinierte Schnittstellen zwischen den Komponenten innerhalb des Frameworks sorgen für eine Austauschbarkeit dieser, um beispielsweise einen anderen Konsensalgorithmus zu verwenden. Klar definierte Transaktionsphasen sorgen weiterhin für eine ständige Validität und Konformität, sowohl syntaktisch innerhalb des Frameworks als auch semantisch innerhalb des Geschäftsprozesses.

Weiterhin wurden neue notwendige Konzepte für eine bessere Integration von Geschäftsprozessen in Blockchain-Frameworks aufgezeigt. Die Verkettung von Blöcken durch Hash-Werte verhindert eine Datenkorrektur, welche jedoch in Geschäftsprozessen häufig nötig ist. Um dies zu ermöglichen, können innerhalb von BFT-PoS Konsensalgorithmen gezielt Abstimmungen zum „Neuschreiben“ der Historie durchgeführt werden. Dieses Verfahren fusst auf der Longest-Chain-Rule, wie sie in Proof-of-Work Algorithmen vorliegt. Dies wird hier jedoch von der Gemeinschaft der Validatoren mutwillig durchgeführt, wodurch ein absichtliches Forken stattfindet.

Neben der reinen Datenkorrektur auf Transaktionsebene entsteht eine weitere Nutzungsmöglichkeit durch das absichtliche Forken. Hierzu wurden explizite Transaktionsabhängigkeiten in die Transaktionen eingeführt, mit dem Ziel, dass sie losgelöst von der Semantik vom Blockchain-Framework selbst identifiziert werden können. So können zusammenhängende Transaktionen in zusammenhängende Graphen zusammengefasst werden, welche nun unabhängig von anderen auf Basis ihrer Semantik verkleinert werden können. Dies ermöglicht es, zusammen mit absichtlichem Forken, die Blockchain-Historie zu verkleinern, aber den globalen Systemzustand unverändert beizubehalten. Irrelevante oder nicht mehr benötigte Transaktionen für den Geschäftsprozess können so aus der Historie der Blockchain gestrichen werden, was ihren Speicherbedarf verringert und auch das Onboarding neuer Knoten beschleunigt.

Basierend auf dem erarbeiteten Konzept innerhalb dieses Kapitels soll im nachfolgenden Kapitel 7 die exemplarische Realisierung dessen stattfinden. Dazu wird ein Systementwurf auf Basis eines noch auszuwählenden Entwicklungsparadigmas stattfinden und anhand dessen konkret dargestellt.

7. Exemplarische Realisierung eines generalisierbaren Blockchain-Frameworks im Rahmen des cadeia-Projekts

Nachdem bereits im vorigen Kapitel, basierend auf den zwei vorgestellten Anwendungsszenarien, ein Gesamtkonzept für eine generalisierte Blockchain entwickelt wurde, soll im Folgenden anhand eines konkreten Systementwurfs eine Umsetzung auch praktisch gezeigt werden. Dabei werden die zuvor aufgezeigten Probleme und Konzepte weiterhin berücksichtigt.

Zunächst werden verschiedene Software-Entwicklungsparadigmen vorgestellt, um daraus ein geeignetes für den Systementwurf auszuwählen. Ziel der Auswahl ist vor allem ein geeignetes Paradigma zu finden, welches es erlaubt, den Fokus auf die notwendige Generalisierbarkeit des anvisierten Blockchain-Frameworks zu legen. Anschließend wird exemplarisch der Entwurf eines Prototypen mit den wichtigsten Komponenten für ein generalisiertes Blockchain-Framework und des entsprechenden Frameworks in seiner Gesamtheit vorgestellt. Abschließend werden die Integrationen für die zusätzlichen Merkmale des Konzepts, wie explizite Transaktions-Abhängigkeiten, Squashing und Widerspruchsverfahren, in cadeia und den für dieses Konzept ausgewählten BFT-Konsensalgorithmus dargestellt.

7.1. Softwareentwicklungsparadigmen

Grundsätzlich ist ein Softwareentwicklungsparadigma ein genereller Stil wie Software entwickelt wird und hat dadurch ebenso Auswirkungen auf die Softwarearchitektur im Ganzen. Einige Paradigmen befassen sich dabei nur mit der Art, wie eine Sequenz an Operationen ausgeführt wird, andere wiederum, wie die Syntax der Programmiersprache und Grammatik aussieht. Die letzte Gruppe behandelt wie der Quelltext angeordnet wird, also in welche Art von Einheiten einzelne Fragmente gruppiert werden und wo oder wie deren Zustand modifiziert wird, sie beschreibt also ganze Softwarearchitekturen.

Da heutzutage viele der vorhandenen Programmiersprachen viele Paradigmen, wie Objekt-Orientierung, Imperative-Programmierung und Funktionale-Programmierung, gleichzeitig unterstützen, wird nachfolgend der Fokus mehr auf Konzepte der höheren Ebenen gelegt, wie etwa Komponenten- und Agentenorientierte Programmierung. Zusätzlich werden Aspekte hervorgehoben, die gerade für die Entwicklung verteilter Systeme von besonderer Rolle sind. Hierunter fallen vor allem die Möglichkeiten bezüglich Skalierbarkeit, Robustheit und Sicherheit und zusätzlich die Fähigkeit für Nebenläufigkeit und Parallelisierung. Besonders Nebenläufigkeit und Skalierung sind für die Entwicklung eines Blockchain-Frameworks von großer Bedeutung, da es sich um ein potentiell diffuses Netzwerk von untereinander unbekannten Teilnehmern mit einer hohen Menge an Nachrichten zwischen ihnen handelt.

7.1.1. Ereignisorientierte Softwareentwicklung

Ein ereignisorientiertes Softwareentwicklungsparadigma (*EP*) wird verwendet, um den Programmfluss zu steuern. Das Programm wird hier nicht linear abgearbeitet, sondern es werden immer wieder durch das Auslösen von bestimmten Ereignissen spezielle Ereignisbehandlungsroutinen (z.B. *Observer, Listener, etc.*) durchlaufen [Meier und Cahill 2005].

Durch das Warten bzw. Polling auf den Eintritt eines Ereignisses in älteren Paradigmen, erscheint es so, dass der Programmfluss still steht oder dies sogar tatsächlich tut. Wenn zwei Prozesse wechselseitig auf das Eintreten eines Ereignisses warten/pollen, ergibt sich ein Deadlock, welcher beide Prozesse potentiell unendlich blockiert. Aus diesen Gründen wird im EP nicht auf Ereignisse gewartet, sondern es wird mit speziellen Funktionen reaktiv auf diese eingegangen. Der Hauptkontrollfluss des Programmes läuft permanent und Ereignisse werden nebenher in eigenständigen Prozessen bearbeitet, aber können auch Einfluss auf den Hauptprozess nehmen. EP eignet sich somit gut für Programme, die auf nebenläufige Prozesse angewiesen sind [Meier und Cahill 2005].

In einem Blockchain-System, in dem mehrere Prozesse nebenläufig stattfinden und permanent auf diverse eingehende Nachrichten eingegangen werden muss, die eigenständige Ereignisse auslösen, wie beispielsweise Validierung, ist ein ereignisorientiertes Paradigma somit generell zu bevorzugen, da hier viele Prozesse nur mit vorigem Ereignis starten.

7.1.2. Komponentenorientierte Softwareentwicklung

Komponentenorientierte Softwareentwicklung beschreibt eine Methodik, bei der es darum geht, Anwendungen als Komposition von mehreren wiederverwendbaren Einzelteilen (Komponenten) zusammenzusetzen. Einzelne Softwarekomponenten sind hierbei so zu verstehen, dass sie selbstständig und eigenständig Teile an Funktionalität abdecken, ganz nach dem Prinzip der *Trennung nach Zuständigkeiten* (engl. separation of concerns). Da Komponenten über definierte Regeln unter fest definierten Regeln und in einer definierten Umgebung miteinander agieren und ihre einzelnen Rollen streng definiert sind legen sie den Grundstein für eine komponentenorientierte Systemarchitektur [Szyperski u. a. 2002].

Während Objekte aus der objektorientierten Softwareentwicklung einzelne Entitäten aus der echten Welt abzubilden versuchen, gehen Komponenten

einen Schritt weiter. Wie auch Objekte kapseln sie ihren internen Zustand von der Außenwelt und stellen Schnittstellen für die Benutzung zur Verfügung, allerdings gruppieren sie intern mehrere Objekte, Prozeduren und Funktionen, die eine starke Abhängigkeit zueinander haben, um sie so als eine gemeinsame wiederverwendbare Funktionalität für Dritte bereitzustellen [Szyperski u. a. 2002].

In Hinblick auf das generalisierbare Blockchain-Framework lässt sich klar feststellen, dass ein komponentenorientiertes Paradigma bzw. eine Architektur förderlich ist. Die bisher vorgestellten konzeptuellen Ideen zeigen, dass viele der Einzelfunktionalitäten, wie beispielsweise der *Mempool*, gekapselte Funktionalitäten darstellen, die nicht stark von anderen Teilen des Systems abhängig sind. Teilweise werden Eingaben aus anderen Teilen des Frameworks benötigt, jedoch lassen sich hier ganz klare Schnittstellen definieren, wie diese zu benutzen sind. Weiterhin lassen sich durch unabhängige Komponenten und klare Schnittstellen eine bessere Austauschbarkeit von einzelnen Teilen des Systems ermöglichen. In Bitcoin (vgl. Kapitel 4.2.2) sind die Einzelteile des Frameworks stark voneinander abhängig, sodass sich nicht leicht einzelne Teile ersetzen lassen, ohne weitere Komponenten zu überarbeiten, um beispielsweise eine andere Anwendungslogik einzubauen.

7.1.3. Agentenorientierte Softwareentwicklung

Agentenorientierte Softwareentwicklung (AOSE) ist ein Entwicklungsparadigma, welches als Schlüsselabstraktion Agenten verwendet. Eine Software kann grundsätzlich aus einem einzigen Agenten bestehen, aber die großen Vorteile von AOSE kommen erst in der Verwendung in Multi-Agenten Systemen (MAS) zum Tragen.

Bis heute gibt es keine einheitliche Definition, was genau ein Agent ist. Die folgende Definition ist hingegen weitgehend akzeptiert und definiert dies für den Rahmen einer Vorstellung von Agenten und AOSE ausreichend.

> *An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.*
>
> M. Wooldridge [Wooldridge 2001]

Agenten agieren autonom, also ohne den Eingriff von außen oder anderen Systemen, sind in einer bestimmten Umgebung aktiv und verfolgen ein Ziel. Zusätzlich hat sich zum Begriff des reinen *Agenten* noch der des *intelligenten Agenten* über die Jahre geprägt. Während Agenten aus ihrer Umgebung Eingaben, beispielsweise über Sensoren, bekommen und dazu Ausgaben produzieren, verfügen intelligente Agenten über die Möglichkeit flexibel autonom zu agieren, um ihr Ziel bestmöglich zu erreichen. Dazu müssen sie nach [Wooldridgey und Ciancarini 2000], über die folgenden Eigenschaften verfügen:

Autonomität Agenten besitzen einen eigenen inneren Zustand, abgeschottet von anderen. Basierend auf diesem treffen sie die folgenden Entscheidungen und Aktionen.

Reaktivität Agenten benötigen die Fähigkeit, Änderungen in der Umgebung wahrzunehmen und daraufhin eigenständig Aktionen auszulösen. Man kann feststellen, dass Agenten auch ereignisorientiert agieren.

Pro-Aktivität Agenten müssen nicht nur bloß auf Änderungen in der Umgebung reagieren, sondern sie müssen auch eigenständig und zielgerichtet Verhalten ausüben, um ihrem Ziel entgegenzuarbeiten.

Sozial Mittels eines geeigneten Kommunikationskanals können Agenten miteinander interagieren, beispielsweise, um für die Erreichung eines Ziels zu kooperieren oder auch zu konkurrieren.

Sobald in einem System mehrere Agenten vorkommen, wird von einem Multi-Agenten System (MAS) gesprochen. MAS charakterisieren sich dadurch, dass jeder darin enthaltene Agent nur unvollständige Informationen oder Fähigkeiten hat sein Ziel zu erreichen und so eine eingeschränkte Sichtweise innehat. Weiterhin gibt es keine globale Systemkontrolle durch eine übergelagerte

Instanz, sodass auch alle Daten dezentralisiert vorliegen und alle Berechnungen asynchron ablaufen müssen [Jennings u. a. 1998; Sycara 1998].

Besonders die Komponierbarkeit von einzelnen Agenten in ein MAS bieten in vielen Systemen deutliche Vorteile gegenüber klassischen Systemarchitekturen. Wie auch in komponentenbasierten Architekturen lässt sich so die Wartbarkeit und Wiederverwendbarkeit von einzelnen Teilen des Systems deutlich erhöhen. Die Wartbarkeit wird dadurch erhöht, dass jeder Agent unabhängig von den anderen aktualisiert werden kann, ohne dass die anderen davon beeinträchtigt werden. Ebenso können sie in anderen Systemen wiederverwendet werden, da sie eine gekapselte Funktionalität anbieten [Sycara 1998].

Ein MAS kann außerdem deutlich recheneffizienter (*computational efficiency*) sein, da Parallelisierung durch Asynchronität und Kommunikation erreicht wird. Die Verlässlichkeit bzw. Robustheit ist durch redundant angelegte Agenten höher und erlaubt eine bessere *Recovery* bei Ausfällen. Agenten lokalisieren sich erst dynamisch bei Bedarf und sind somit nicht davon beeinträchtigt, dass von einem gewünschten Funktionstypus von Agenten nur drei statt vier zur Verfügung stehen. Eine Skalierung des Gesamtsystems kann schnell erreicht werden indem von ausgelasteten Agenten ad-hoc neue zusätzliche Instanzen gestartet werden [Jennings u. a. 1998].

Für ein generalisierbares Blockchain-Framework bietet sich ein Agenten- bzw. Multi-Agenten-System aufgrund seiner inhärenten Eigenschaften, wie Komposition, Effizienz, Robustheit und Asynchronität an. Im Vergleich zu einer reinen komponentenbasierten Architektur bieten intelligente Agenten noch zusätzliche Möglichkeiten, wie beispielsweise flexibel und autonom auf Ereignisse einzugehen, aber auch proaktiv selbstständig tätig zu werden. Inwiefern alle Eigenschaften von intelligenten Agenten, wie beispielsweise Pro-Aktivität, wirklich benötigt werden, lässt sich an dieser Stelle nicht leicht beantworten. Denkbar wäre aber eine ständige Selbstorganisation der Transaktionen innerhalb des Mempools durch den entsprechenden Agenten, oder

einen Konsens-Agenten, dessen Ziel es ist den Konsens selbstständig aufrechtzuerhalten beziehungsweise weiterzuführen.

7.1.4. Service-orientierte Architekturen

Service- oder dienstorientierte Architekturen (SOA) stellen ein lose gekoppeltes Architekturmuster für zumeist verteilte Systeme dar, um die Anforderungen von Firmen an Geschäftsprozessen abzubilden. Ähnlich zu Komponentenorientierung, werden in SOA mit Komposition mehrerer Services und deren zur Verfügung gestellten Dienste ein Gesamtsystem zusammengesteckt [Huhns und Singh 2005].

Über die reine Komposition von Services beschreibt SOA noch zusätzliche Systemdienste und Rollen innerhalb des Systems. Jeder Dienstanbieter (Service Provider) meldet bei einem Registrierungsdienst (Service Registry & Broker) über eine definierte Sprache, beispielsweise *Web Services Description Language* (WSDL), seine eigenen Kapazitäten, Funktionen, Daten und Datentypen und Nachrichtenprotokolle. Über die Registry können nun andere Dienstnehmer (Service Requestor) genau erfahren wie mit dem benötigten Dienst kommuniziert wird und welche Funktion er anbietet. Über die Registry können natürlich auch mehrere Anbieter dieselbe Schnittstelle und Funktion anbieten, sodass sich der Dienstnehmer erst dynamisch zur Laufzeit für einen entscheidet. So wird auch eine hohe Skalierbarkeit des Systems durch redundante Dienste erreicht. Sobald ein konkreter Dienst verwendet wird, wird dieser „gebunden“ und jegliche Kommunikation läuft nun direkt über die beiden Parteien. Die direkte Kommunikation unterliegt in webbasierten Systemen meist dem XML-basierten *SOAP* Standard [Huhns und Singh 2005; Papazoglou 2003].

Eine Weiterentwicklung von SOA stellt die „Service Component Architecture“ (SCA) dar. Diese vereint die Serviceorientierung mit der Komponentenorientierung zur Erstellung von dienstorientierten, komponierbaren und wiederverwendbaren Einzelteilen [Marino und Rowley 2009].

Für das zu entwickelnde Blockchain-Framework eignet sich SOA insofern, als dass eine klare Strukturierung der Komponenten und ihrer Schnittstellen nach außen hin definiert und spezifiziert sind. So wird einerseits erreicht, dass innerhalb des Frameworks ganz deutlich die jeweiligen Funktionen und Interaktionsmöglichkeiten der Komponenten dargestellt sind, aber auch dass die Schnittstellen zwischen zwei Instanzen des Frameworks, also zwischen zwei Blockchain-Knoten, beschrieben sind.

7.1.5. Aktive Komponenten

Das Konzept der aktiven Komponenten stellt ein Architektur- und Entwicklungsparadigma dar, dass die vorher vorgestellten Paradigmen in einem vereint. Aktive Komponenten vereinen die Paradigmen der serviceorientierten und komponentenorientierten Softwareentwicklung (bzw. SCA) mit den autonomen Eigenschaften von intelligenten Agenten. Dies schafft die Möglichkeit, passive Komponenten durch aktive Akteure zu erweitern, die selbstständig gegenseitig ihre angebotenen Dienste wahrnehmen. So können besser realitätsnahe Szenarien mit vielen aktiven Teilnehmern in Software repräsentiert werden [Braubach und Pokahr 2011].

Aktive Komponenten sind also autonome, geführte, hierarchische Softwareentitäten, die Funktionen bereitstellen und andere Funktionen mittels definierter Services verwenden und anhand ihrer internen Architektur selbst definiert sind. Im Gegensatz zu SOA und Komponentenorientierung bzw. SCA , werden in Aktiven Komponenten explizit bereitgestellte und benötigte Services deklariert und können zusätzlich potentiell mehrere Subkomponenten vereinigen, wie in Abbildung 7.1 veranschaulicht.

Mit der Verwendung des Paradigmas der Aktiven Komponenten werden Entwickler zusätzlich entlastet. Da Aktive Komponenten notwendige Mechanismen für die Entwicklung verteilter Systeme mitbringen, müssen viele sonst aufwendige umzusetzende Funktionen, wie asynchrone Funktionsaufrufe in objektorientierter Softwareentwicklung, nicht nachträglich hän-

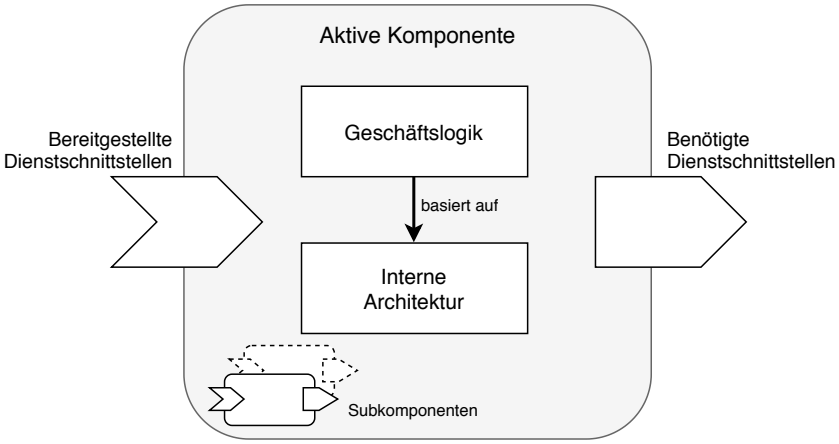

Abbildung 7.1.: Aktive Komponente (nach [Pokahr und Braubach 2011]

disch eingepflegt werden. Asynchrone Funktionen und Dienstaufrufe werden mithilfe des Aktormodells deutlich vereinfacht [Hewitt u. a. 1973]. Auch das Warten auf Ergebnisse nebenläufiger Kontrollflüsse, beziehungsweise die nicht blockierende Ausführung, werden durch Konzepte für Platzhalter von bisher unbekannten Ergebnissen durch sogenannte *Futures* (bzw. *Promises*, *Proxies*) ermöglicht [Pokahr und Braubach 2011].

7.1.6. Auswahl eines Paradigmas zum prototypischen Systementwurf

Aktive Komponenten vereinen viele der vorher vorgestellten Paradigmen in einem einzigen Softwareentwicklungs-Paradigma. Die Vorteile der anderen Paradigmen werden hier in einem vereint und bietet somit die beste Möglichkeit für die Umsetzung eines generalisierbaren Blockchain-Frameworks. Abbildung 7.2 zeigt die drei Grundsäulen für verteilte Anwendungen: Nebenläufigkeit, Verteilung und Nicht-Funktionale Kriterien.

Durch Services werden spezifische Schnittstellen aller Komponenten nach außen bekannt gemacht, sodass eine Interaktion mit diesen klar definiert ist. Mittels SOA werden die Anforderungen an Nicht-Funktionale Kriterien und

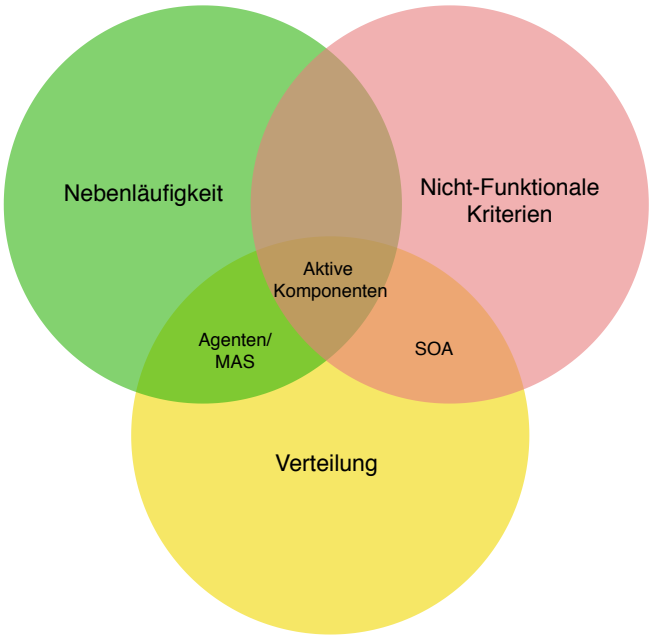

Abbildung 7.2.: Aktive Komponenten (nach [Braubach und Pokahr 2011]

Verteilung abgedeckt, jedoch müssen Probleme oder Unzulänglichkeiten bezüglich Nebenläufigkeit selbst behoben werden.

Die Verwendung von autonomen Agenten erlaubt eine pro-aktive dynamische Verhaltensweise der einzelnen Teilkomponenten wo nötig, kann aber auch auf reines reaktives Verhalten begrenzt werden. Während Agenten oder Multiagenten-Systeme besonders gut für Nebenläufigkeit und Verteilung des Systems geeignet sind, decken sie nicht zwingend alle nicht-funktionalen Kriterien aus Geschäftsprozessen ab.

Aktive Komponenten bilden also die Brücke zwischen Multiagenten-Systemen und service-orientierten Architekturen und ermöglichen es die drei erforderlichen Eigenschaften zur Nebenläufigkeit, Verteilung und den zusätzlichen nicht-funktionalen Kriterien abzudecken. Die Hilfestellungen für Entwickler bezüglich Nebenläufigkeit und einer ereignisähnlichen Struktur der Futures erleichtern zusätzlich die Umsetzung eines verteilten Systems enorm.

Somit eignet sich das Paradigma der Aktiven Komponenten am besten für einen den genannten Kriterien entsprechenden Entwurf und für die Umsetzung des vorgeschlagenen Konzeptes. Dies trifft auch zu, obwohl nicht zwingend in allen benötigten Komponenten die vollständigen Funktionalitäten der Aktiven Komponenten ausgeschöpft werden.

7.2. Prototypischer Entwurf einer generalisierbaren Blockchain

Nachdem im vorhergehenden Abschnitt eine Auswahl des Entwicklungsparadigmas stattgefunden hat, soll nun nachfolgend eine konkrete Umsetzung eines Prototypen für ein generalisiertes Blockchain-Framework stattfinden. Um das Paradigma der aktiven Komponenten innerhalb eines Prototypens umzusetzen, muss vorerst eine geeignete Software-Bibliothek beziehungsweise eine gesamte Laufzeit- oder Ausführungsumgebung ausgewählt werden, welche nachfolgend vorgestellt wird.

7.2.1. Ausführungsumgebung

Generell beschreibt oder legt eine Ausführungsumgebung fest, welche Vorraussetzungen, Sprachkonstrukte oder Funktionen für den Entwickler bereitgestellt werden und wie diese ausgeführt werden.

Für die Entwicklung eines generalisierbaren Blockchain-Frameworks mit aktiven Komponenten bietet sich das Middleware-Framework *Jadex Active Components Framework* (kurz Jadex) an. Jadex war ursprünglich eine Erweiterung des Agenten-Frameworks JADE. Jadex erweitert dabei die Jade-Middleware um zusätzliche Funktionen zur komponenten- und serviceorientierten Softwareentwicklung, zur Umsetzung von aktiven Komponenten. Weitere Funktionen sind die Erstellung von Modellen auf Basis von Business Process Modelling Notation (BPMN) und Goal Oriented Process Modelling Notation

(GPNM) Workflows, so wie die Erweiterung der Agenten um Belief-Desire-Intention (BDI) Funktionalitäten.

Konzeptionell vereint der in Jadex genutzte Ansatz der *Aktiven Komponenten* die Eigenschaften der *Service Component Architecture*, die ihrerseits eine Vereinigung von Diensten und Komponentenansätzen darstellt, mit denen des agentenorientierten Paradigmas [Pokahr und Braubach 2011, 2013]. Durch die anderen integrierten Modellierungsarten lassen sich andere Architekturen umsetzen. Im Einzelnen sind dies:

Micro-Agenten Bilden die Basis-Architektur durch einfache Java-Klassen, die das Aktive Komponenten Paradigma unterstützen und ihr eigenes Verhalten durch Java-Code und sogenannte Komponenten-Schritte ausdrücken.

BDI-Agenten Bieten eine abstrakte Modellierung der Verhaltensweise durch Beliefs, Desires und Intentions. Diese Konzepte erlauben es, intelligente Verhaltensweisen abzubilden, die an das menschliche Denkvermögen angelehnt sind [Georgeff u. a. 1998].

BPMN-Workflow Prozesse lassen sich grafisch in BPMN modellieren und in einer Komponente gekapselt ausführen.

GPMN-Workflow Erlaubt das zielorientierte Modellieren von Prozessen und somit eine flexiblere Ausführung im Vergleich zu BPMN-Workflows.

Die Entwicklung mit Jadex bietet zwei wesentliche Vorteile, die bei dem Entwurf und der Umsetzung des Konzeptes für ein generelles Blockchain-Framework genutzt werden können. Erstens bietet der komponentenorientierte Ansatz eine Möglichkeit zur Dekomposition eines Softwaresystems in Komponenten, die in sich abgeschlossene Funktionseinheiten darstellen. Eine Komponente bietet nach außen eine explizit definierte Schnittstelle an, die wiederum von anderen Komponenten genutzt werden kann. Dies kann im cadeia Prototypen zur Modellierung der Beziehungen zwischen den einzelnen Bestandteilen des Systems innerhalb eines Knoten, aber auch zwischen zwei Knoten, genutzt werden.

Der zweite wesentliche Vorteil bei der Nutzung von Jadex wird durch den Ansatz der *Aktiven Komponenten*, der in Jadex verwendet wird, und die inhärente Unterstützung für die Entwicklung verteilter Systeme in Form von Verteilungstransparenz gewährleistet. So können Instanzen von verschiedenen Softwarekomponenten auf unterschiedlichen Systemen laufen und miteinander kommunizieren. Während der Entwicklung spielt es daher keine Rolle, ob Komponenten später lokal oder entfernt ausgeführt werden, da die Zugriffe und Funktionsaufrufe identisch ablaufen.

Neben der internen Architektur verfügt eine Aktive Komponente über weitere zentrale Eigenschaften: Sie kann Dienste anderer Komponenten nutzen (required services) und selbst Dienste anbieten (provided services). Die genutzten Dienste stellen eine Abhängigkeit zu anderen Komponenten dar und bilden gleichzeitig auch die Kommunikationsbeziehungen der Komponenten. Die Dienste werden durch ein Java-Interface definiert und anschließend innerhalb der Komponente implementiert. Es können auch verschiedene Komponenten die gleiche Schnittstelle anbieten, aber unterschiedlich implementieren, für den Aufrufenden ist dies transparent. Über die *Awareness*-Komponente der Jadex-Middleware werden zur Laufzeit geeignete Dienstanbieter und andere Plattformen gefunden und gebunden, ähnlich zur Service-Registry in SOA. Jede Kommunikation über die eigene Plattformgrenze hinaus wird standardmäßig Ende-zu-Ende verschlüsselt, was für den Einsatz in Geschäftsprozessen und -anwendungen meist essenziell ist. Weiterhin können Komponenten durch Properties von außen konfiguriert werden und hierarchisch verschachtelt werden, um eine sinnvolle funktionale Dekomposition zu erreichen.

Um eine Aktive Komponente auszuführen, wird die Jadex-Plattform als Laufzeitumgebung benötigt. Diese bietet eine Umgebung mit allgemeinen Funktionalitäten wie Komponenten-, Lebenszyklusmanagement und Kommunikationsschicht an, die von allen Komponenten genutzt werden können. Die konkrete Umsetzung des Blockchain-Framework Prototyps wird von Jadex weiterhin durch die folgenden Funktionen erleichtert:

Service-Orientierung Jegliche Funktionalitäten werden nach außen hin als Dienste angeboten und definieren somit die Schnittstellen zwischen verschiedenen Komponenten auf unterschiedlichen Plattformen klar. Im Falle eines entfernten Funktionsaufrufens sorgen Service Proxies für den transparenten Aufruf verteilter Dienste, sodass bei der Entwicklung nicht unterschieden werden muss.

Asynchrones Programmiermodell Rückgabetypen von Diensten sind in Jadex üblicherweise sogenannte *textit*. Diese symbolisieren das Versprechen (ähnlich zu *Promise*, *Proxies*), zu einem späteren Zeitpunkt ein Ergebnis zu liefern. Wird dieses Programmiermodell konsequent umgesetzt, kann systematisch mit nebenläufiger und paralleler Ausführung umgegangen werden. Das Warten auf Ergebnisse kann entweder durch Blockieren des Threads erreicht werden, oder durch das Anmelden von *Beobachtern* am Future selbst.

Kommunikationsschicht Jadex bietet die Kommunikation auf der Ebene von Nachrichten oder Service-Aufrufen an, so dass keine eigene Kommunikationsschicht implementiert werden muss. Durch die Awareness-Komponenten können automatisch entfernte Plattformen und Komponenten aufgefunden und wie lokale Komponenten verwendet werden.

Simulationsumgebung Durch eine integrierte Simulationsumgebung wird den Entwicklern zusätzlich ein Werkzeug an die Hand gegeben, mit dem sie Software unter Realbedingungen und Ausführungszeiten, aber auch unter verlangsamten oder beschleunigten Laufzeiten testen und simulieren können.

Die Jadex-Middleware ist komplett als Java-Bibliothek verfügbar und benötigt somit als eigene Ausführungsumgebung die *Java Virtual Machine* (JVM). Der Vorteil hiervon ist, dass durch die JVM von betriebssystemspezifischen Eigenschaften abstrahiert wird und der zu entwickelnde Prototyp auf jedem Betriebssystem einsatzfähig ist. Weiterhin ist durch die Wahl von Java als Programmiersprache die Einstiegshürde in die Entwicklung gering, da Java im Geschäftsumfeld sehr stark verbreitet ist.

7.2.2. Architektur des Frameworks

Für die exemplarische Realisierung der vorgeschlagenen Konzepte eines generalisierbaren Blockchain-Frameworks bieten das Jadex-Framework und das Aktive Komponenten-Paradigma, wie bereits vorausgehend erwähnt, besondere Vorteile bei der Umsetzung. Um die Vorteile zu nutzen, muss auf die Vorgaben des Frameworks geachtet werden, was bedeutet, dass möglichst alle benötigten Teilfunktionen für eine Blockchain als einzelne gekapselte Komponenten umgesetzt werden.

Wie schon im Konzept in Abschnitt 6.2 erläutert, werden für die Grundfunktionalitäten einer Blockchain die Komponenten Persistenz, Kommunikation, Mempool und Konsens benötigt, zusätzlich zu einer integrierten oder externen Geschäftslogik zur Validierung. Hierbei wird für den Prototypen eine externe Geschäftslogik, die über eine Service-Schnittstelle mit dem Framework kommuniziert, bevorzugt. So bleibt das Blockchain-Framework unabhängig von bestimmten geschäftlichen Use-Cases, anders als beispielsweise in Bitcoin, wo diese direkt ineinander verbunden sind.

Abbildung 7.3 zeigt die anvisierte Aufteilung der Komponenten sowohl auf Frameworkebene als auch in der Anwendungsschicht. Über eine gebündelte Schnittstelle im Framework werden interne Schnittstellen vor der Anwendungsschicht verborgen, sodass diese nur die nötigen Funktionen des Frameworks abrufen kann. Die Geschäftskomponente muss ihrerseits eine vordefinierte Schnittstelle bereitstellen, damit das Framework Funktionen, beispielsweise zur Validierung, aufrufen kann.

7.2.3. Komponenten und deren Service-Schnittstellen

Aufbauend auf der vorhergehend konzipierten Architektur werden nachfolgend die benötigte Schnittstelle der Anwendung für die Geschäftslogik für das Blockchain-Framework und die einzelnen vier Hauptkomponenten des

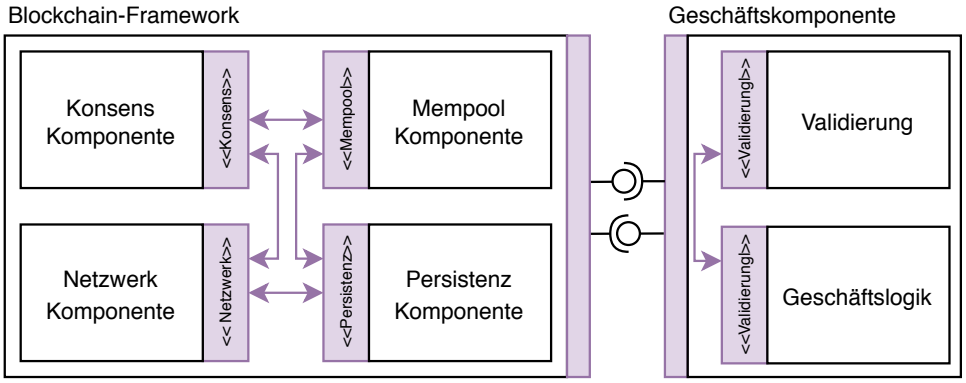

Abbildung 7.3.: Vorgeschlagene Komponenten-Architektur

Frameworks selbst und ihre bereitgestellten sowie erforderlichen Schnittstellen vorgestellt.

Anwendungsschnittstelle

Um ein generalisierbares Blockchain-Framework zu entwickeln, dass intern frei von jeglicher Anwendungssemantik funktioniert, muss eine Schnittstelle zu einer Geschäftskomponente vorgegeben werden. Dies sorgt dafür, dass alle Aufgaben und Entscheidungen, die nicht vom Framework getroffen werden können, von dieser externen Komponente übernommen werden. Die folgenden drei Funktionen werden in verschiedenen Phasen des Zyklus aufgerufen und beziehen sich auch direkt auf die in Kapitel 6.2.2 vorgestellten Transaktionsphasen *atomar*, *historisch* und *Intra-Block* .

`validateTransaction(Tx):Future`
: Überprüft eine Transaktion auf ihre eigenständige Gültigkeit. Dies stellt die atomare Gültigkeit der Transaktion fest. An dieser Stelle kann in der Anwendung zusätzlich schon über die historische Gültigkeit, also Bezug zum bisherigen Zustand, entschieden werden.

`validateProposal(List<Tx>):Future`
: Wenn ein Blockvorschlag (Proposal) beim lokalen Knoten erstellt wird,

ist es wichtig, dass dieser validiert wird und die gewählte Reihenfolge gültig ist. Über den Rückgabewert kann dem Framework mitgeteilt werden, welche Reihenfolge zu befolgen ist und welche Transaktionen temporär oder permanent ausgeschlossen werden sollen. Es kann an dieser Stelle in der Anwendung davon ausgegangen werden, dass Transaktionen, die in einem Vorschlag enthalten sind, bereits atomar gültig sind.

`validateBlock(Block):Future`

Wenn ein Block von anderen Knoten eintrifft, muss dieser validiert werden. Hierbei sind alle drei Phasen der Transaktionen zu validieren: atomare, historische und Intra-Block Gültigkeit. Die Reihenfolge der Transaktionen spielt ebenfalls eine Rolle. Die Validierung kann mit einem Abbruch antworten, sobald die erste ungültige Transaktion gefunden wurde.

Die Herausforderung für Entwickler bei der Implementierung der Validierungslogik ist gegeben durch die drei Transaktionsphasen. Für eine atomare Überprüfung ist beispielsweise eine Überprüfung der einzelnen Werte bereits ausreichend und trivial umsetzbar. Für die historische Überprüfung muss anhand des aktuellen Anwendungszustands geprüft werden, ob eine Ausführung der Transaktion eine gültige Zustandsveränderung ist. Der aufwendigste Teil ist die Intra-Block Validierung: Hier müssen die Transaktionen der Reihe nach auf dem aktuellen Zustand ausgeführt und temporär persistiert und danach auf Gültigkeit geprüft werden. Sobald eine Transaktion fehlschlägt oder die Überprüfung erfolgreich beendet ist, muss der Zustand zum Zeitpunkt vor Beginn der Validierung wieder hergestellt werden. Um dies zu vereinfachen, gibt es verschiedene Möglichkeiten, wie Snapshots, Rollbacks oder temporäre Kopien des gesamten Zustandes, die einige Datenbanktechnologien unterstützen.

Persistenz

Die Persistenzkomponente ist die einzige der benötigten Komponenten, die ohne das Zutun anderer Komponenten agieren kann. Ihre Aufgabe ist es lediglich, Blöcke und Transaktionen zu persistieren und diese auf Anfrage wieder herauszugeben. Um dies zu leisten, werden keinerlei Funktionen von anderen Komponenten verwendet. Die Persistenz agiert also rein passiv beziehungsweise reaktiv auf Anfragen von anderen.

Für das eigentliche Speichern, auf beispielsweise der Festplatte, wird keine sonderlich anspruchsvolle Datenbanksoftware benötigt. Viele andere Blockchains nutzen einfache *Key-Value-Stores*. Dies wird vor allem dadurch begünstigt, dass alle Blöcke und Transaktionen über ihren jeweiligen Hash identifizierbar sind und keine Abfrage-Sprache, wie SQL, benötigt wird, um auf Transaktionsinhalte zuzugreifen.

Um die Persistenzkomponente zu verwenden, benötigen einige der anderen Komponenten bestimmte Funktionen, wie nachfolgend dargestellt:

`saveBlock(Block):Future` speichert permanent den Block, inklusive aller Metadaten und Transaktionen, in der Datenbank. Der Rückgabewert informiert innerhalb eines Futures, ob die Operation erfolgreich war.

`getBlock(Hash):Future` sucht einen Block anhand seines Root-Hashes in der Datenbank. Sobald ein Ergebnis vorliegt, wird der Aufrufer darüber benachrichtigt.

`getBlockAtHeight(height):Future` sucht einen Block anhand seiner Höhe in der Datenbank. Sobald ein Ergebnis vorliegt, wird der Aufrufer darüber benachrichtigt.

`getTransaction(Hash):Future` sucht eine Transaktion anhand ihres Hashes in der Datenbank. Diese Funktion dient außerdem als Methode, um zu verifizieren, dass eine neue Transaktion kein Duplikat ist oder eine Hash-Kollision hervorruft, was zu verhindern ist. Sobald ein Ergebnis vorliegt, wird der Aufrufer darüber benachrichtigt.

```
1  private org.iq80.leveldb.DB blocks;
2  private org.iq80.leveldb.DB transactions;
3
4  public IFuture<Boolean> saveBlock(Block block) {
5    // create a block with pointers to transactions
6    BlockShell shell = new BlockShell(block);
7    byte[] blockHeight = longToBytes(block.getHeight());
8    byte[] serializedBlock = JadexByteSerialize.serialize(shell);
9
10   // put whole block at height:blockHeight into db
11   blocks.put(blockHeight, serializedBlock);
12   // reference height
13   blocks.put(block.getHash().getData(), blockHeight);
14
15   saveTransactions(block.getTransactions());
16   return Future.TRUE;
17 }
18
19 private void saveTransactions(List<Transaction> transactionList) {
20   WriteBatch batch = transactions.createWriteBatch();
21   transactionList.forEach(tx -> {
22     byte[] bs = transactions.get(tx.getHash().getData());
23     batch.put(tx.getHash().getData(), tx.getData());
24   });
25   transactions.write(batch);
26 }
```

Algorithmus 7.1: Speichern von Blöcken in LevelDB

Um die Suche und den Zugriff auf einzelne Transaktionen zu beschleunigen ist es sinnvoll, dass der Block nicht komplett serialisiert wird, sondern mit Zeigern (Pointer) auf die echten Daten versehen ist. Demnach würden zwei Datenbanken (oder Tabellen) benötigt, welche jeweils nur Blöcke und Transaktionen enthalten. Die Transaktionszeiger innerhalb der Blöcke zeigen dann nur noch auf den Eintrag innerhalb der Transaktionsdatenbank. Als Zeiger kann hier wieder der Hash der Transaktion verwendet werden, so wird auch der Zugriff auf einzelne Transaktionen über ihren Hash ermöglicht.

Algorithmus 7.1 zeigt einen Ausschnitt aus der Persistenzkomponente, die intern eine LevelDB (Key-Value Store) verwendet um Blöcke und Transaktionen zu speichern. Dabei wird zuerst der Block von seinen Transaktionen über `BlockShell` getrennt, damit diese separat serialisiert und anschließend ge-

speichert werden können. Die Funktionen, um Blöcke aus der Datenbank zu extrahieren, erfolgen analog hierzu rückwärts. In der konkreten Implementation werden Blöcke anhand ihrer Blockhöhe gespeichert. Dazu wird die Blockhöhe als Schlüssel und der serialisierte Block als Wert verwendet. Um zusätzlich anhand des Blockhashes einen Zugriff zu ermöglichen, wird außerdem ein Zeiger in der Form $block_{hash} \rightarrow block_{height}$ angelegt.

Mempool

Der Mempool ist die Schlüsselkomponente, die vom Konsens und der Kommunikation verwendet wird, um alle einkommenden Transaktionen flüchtig zwischenzuspeichern. Wenn aktuellen Knoten ein Block vorgeschlagen wird, aber auch wenn ein anderer Knoten einen Block vorschlägt, können aus dem Mempool die Transaktionen entnommen werden. Dies ermöglicht einerseits dem Knoten eine zentrale Stelle an der potentielle Kandidaten für einen Block sind, andererseits spart dies Bandbreite bei der Übertragung im Netz, wenn nur die Blöcke inklusive Transaktions-Hashes übertragen werden. Abgesehen von reaktiven Aufgaben, kann der Mempool sich periodisch selbst „bereinigen“, in dem er alte Transaktionen automatisch löscht.

`receiveTransaction(Transaction):void` empfängt eine Transaktion und speichert sie in den Mempool. Dabei sollten alle Transaktionen in der Reihenfolge vorliegen, in der sie eingetroffen sind und sie sollen außerdem mit einer *Time-To-Live* (Lebenszeit) versehen werden. Dies erlaubt periodisch die Überprüfung auf veraltete Transaktionen und lässt die Bereinigung des Mempools zu.

`getTransaction(Hash):Future` verifiziert ob eine Transaktion im Mempool vorhanden ist. Sobald ein Ergebnis vorliegt, wird der Aufrufer darüber benachrichtigt.

`removeFromMempool(Hash):void` entfernt eine Transaktion aus dem Mempool. Dies kann aus verschiedenen Gründen erfolgen, wie bei-

spielsweise der periodischen Selbstbereinigung oder von außen, wenn festgestellt wurde, dass eine Transaktion nicht mehr gültig ist.

An dieser Stelle wurde bewusst auf eine direkte Integration mit der Anwendungsschicht verzichtet, um den Mempool möglichst simpel zu belassen. Dies bedeutet wiederum, dass eine Validierung der einzustellenden Transaktionen innerhalb der Komponente erfolgen muss, die sie empfangen hat. So wird implizit erreicht, dass alle Transaktionen, die sich im Mempool befinden, auf jeden Fall zum Einstellungszeitpunkt *atomar* gültig sind.

P2P-Kommunikation

Obwohl innerhalb des Jadex-Frameworks alle Komponenten mit ihren jeweiligen Gegenparts auf den entfernten Knoten über die definierten Dienst-Schnittstellen über die Awareness direkt kommunizieren könnten, wurde sich hier dazu entschieden, eine nachrichtenbasierte Kommunikationskomponente einzuführen. Dies schirmt einerseits alle Komponenten innerhalb eines Knotens von direkten Zugriffen ab und andererseits treffen Nachrichten so zentral an einer Stelle im System ein. Weiterhin lässt sich so ein Overlay-Netzwerk innerhalb der Jadex-Netzstruktur etablieren, um auch andere Peer-to-Peer Protokolle für den jeweiligen konkreten Einsatzzweck zu evaluieren und umzusetzen.

Grundsätzlich benötigt daher die `P2PService`-Dienstschnittstelle somit nur zwei grundlegende Funktionen, nämlich das Senden von Nachrichten und das Reagieren auf einkommende Nachrichten, wie folgt dargestellt:

`broadcastMessage(Object):Future` versendet eine beliebige Nachricht innerhalb des Netzwerkes unter Beachtung der Regeln des drunterliegenden Netzwerk-Protokolls. Die konkrete Ausprägung des zu versendenden Objektes spielt für die Schnittstellendefinition keine Rolle und kann dynamisch zur Laufzeit erfolgen.

`registerForType(Type):SubscriptionFuture` dient als Schnittstelle für die Verwendung des *Observer-Patterns*, damit sich verschiedene Komponenten über eintreffende Nachrichten benachrichtigen lassen können. Das `SubscriptionFuture` fungiert hier als Sonderfall eines Futures, denn es können hier permanent neue Werte asynchron nachgeliefert werden. Über den Type-Parameter lässt sich festlegen, welche Nachrichtentypen eine Komponente abonnieren möchte, um gezielt nur relevante Ereignisse zu bearbeiten.

Während `broadcastMessage` verwendet wird, um von einem Knoten den Nachrichtenversand zu starten, muss auf der anderen Seite eine globale Funktion zum Empfangen bereitstehen. Da dies von der Implementation des Protokolls abhängig ist, ist eine **`receiveMessage(Object...)`** nicht Teil der Schnittstellenbeschreibung, aber in der konkreten Implementation meist erforderlich. Während also `broadcastMessage` und `registerForType` Schnittstellen innerhalb eines Knotens darstellen, muss `receiveMessage` von außen erreichbar sein.

Ein Funktionsaufrufschema, wie eine Nachricht vom Typ X von einem Knoten A an einen Knoten B geschickt wird, ist in Abbildung 7.4 beispielhaft dargestellt.

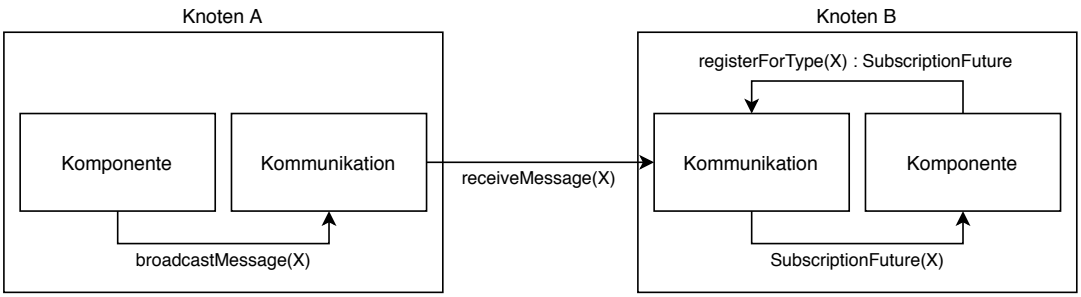

Abbildung 7.4.: Kommunikationsaufrufschema

Konsens

Wie bereits mehrfach dargestellt, ist die Konsens-Komponente das Herzstück jeder Blockchain, da sie darüber entscheidet, in welcher Form und nach welchen Regeln Transaktionen verarbeitet werden. Beim Konsens handelt es sich um eine autonome Komponente, die reaktiv und proaktiv agieren muss. Viele Prozesse laufen hier einerseits eigenständig oder periodisch ab, aber auch reaktiv durch einkommende Nachrichten. Für andere Komponenten innerhalb desselben Knotens besteht keine Notwendigkeit, in irgendeiner Form auf den Konsens einzuwirken, denn der Konsensalgorithmus definiert die nötigen Schritte, wie beispielsweise die Bündelung von Transaktionen zu einem Block. Daraus resultierend bietet die Konsens-Schnittstelle auch keine Funktionen mit der aktiv auf den Konsens eingewirkt werden kann, wie folgt dargestellt:

`beginConsensus()` wird verwendet, um dem Konsensdienst zu signalisieren, dass alle anderen Komponenten einsatzfähig sind und dieser nun mit dem Konsensalgorithmus beginnen kann.

`pauseConsensus()` wird verwendet, um dem Konsensdienst zu signalisieren den aktuellen laufenden Algorithmus zu pausieren.

`registerForBlocks():SubscriptionFuture` ermöglicht die Benachrichtigung über neue fertiggestellte Blöcke. Die Anwendungsschicht kann so über neue Zustandsveränderungen, die bereits durch den Konsens als gültig bestätigt wurden, durch den Konsensdienst benachrichtig werden.

Die Idee hinter dieser Modellierung ist, dass der Konsens einmalig pro Knoten gestartet wird, nachdem alle anderen Komponenten einsatzbereit sind und dann fortlaufend eigenständig seine Arbeit verrichtet. Über die `registerForBlocks` Schnittstelle können andere Komponenten resultierende Zustandsänderungen durch das Hinzufügen neuer Blöcke abonnieren, was nach jetzigem Konzept und Stand des Prototypen nur die Anwendungsschicht betrifft.

7.2.4. Basis-Datenstrukturen

Innerhalb einer Blockchain gibt es grundlegend immer dieselben zwei Datenstrukturen, die Transaktionen als kleinste Elemente und dadrüber geordnet die Blöcke als Sammlung von Transaktionen.

Transaktion

Um für alle möglichen Anwendungsfälle die Transaktionen gleich modellierbar zu machen, wurde hier auf die simpelste Lösung zurückgegriffen, ein Byte-Array. Dies wird vereinfacht im Algorithmus 7.2 dargestellt.

Da es für alle Komponenten des cadeia-Prototypen irrelevant ist, was die Transaktionen konkret bedeuten, spielt auch die Repräsentation innerhalb der Anwendungsschicht keine Rolle. Die einzige Anforderung ist, dass sich ein Hash aus der Transaktion bilden lässt. Die Hashes werden zur Feststellung der Einzigartigkeit verwendet, wodurch doppelte Transaktionen mit identischem Inhalt verhindert werden. Weiterhin werden die Hashes innerhalb des MerkleTrees des Blockes verwendet.

```
1  class Transaction {
2
3      byte[] data;
4
5      public byte[] getHash() {
6          return Hash.of(data);
7      }
8
9  }
```

Algorithmus 7.2: Vereinfachter Auszug aus der Transaktions-Klasse

Für die Validierung einer Transaktion wird über die entsprechende Schnittstelle immer die Anwendungsschicht verwendet. So wird nur in der Anwendungsschicht über die Semantik der Transaktion entschieden. Um in der Anwendungsschicht ein domänenspezifisches Objekt in eine Transaktion zu

konvertieren, kann dies beispielsweise über gängige JSON-Serializer oder über den in Jadex bereits enthaltenen `SBinarySerializer2` erfolgen.

Abbildung 7.5 zeigt beispielhaft den Ablauf vom Einstellen einer Transaktion von der Anwendungsseite aus und den Ablauf, wie eine Transaktion durch die Anwendung validiert wird. Bevor die Anwendung ihr Domänenobjekt in eine Transaktion konvertiert, muss das Objekt in einen Byte-array umgewandelt werden, danach kann dieser in Form einer Transaktion an das Framework geschickt werden. Wenn das Framework eine Anfrage zur Validierung einer Transaktion an die Anwendung stellt, läuft der Prozess quasi rückwärts ab. Über den Serializer wird der Inhalt der Transaktion wieder in ein Domänenobjekt umgewandelt, welches dann mittels der Anwendungslogik validiert werden kann. Anhand des Futures kann dann asynchron der Ausgang der Validierung signalisiert werden.

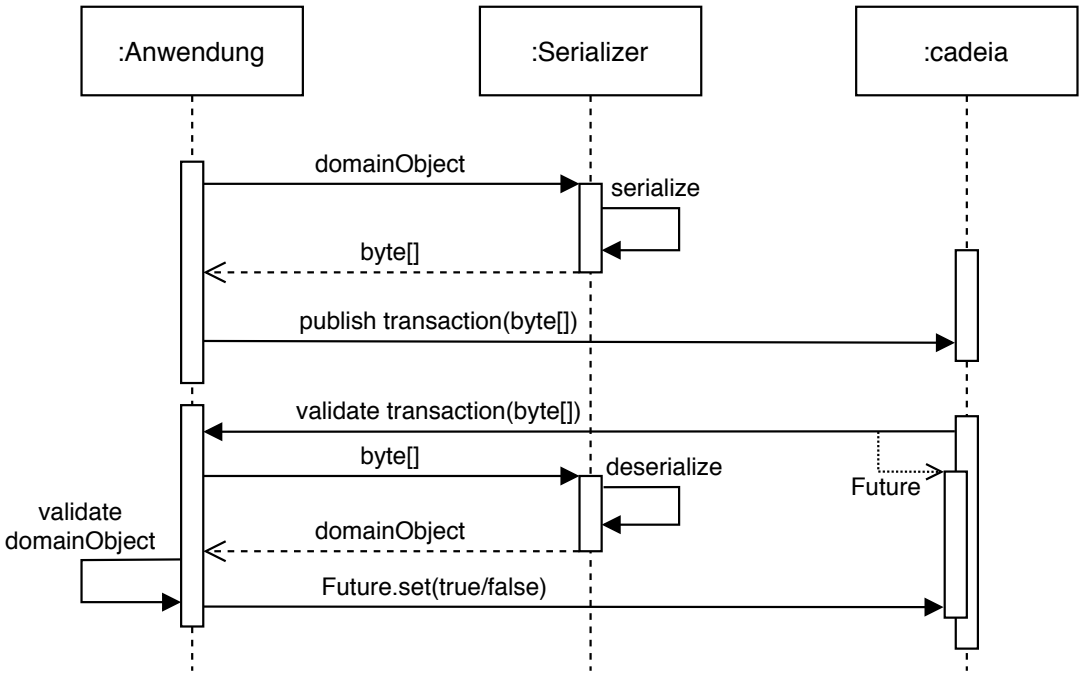

Abbildung 7.5.: Veröffentlichen und Validieren von Transaktionen

Block

Analog zu vielen anderen Blockchains wird auch hier der Block zur Bündelung der Transaktionen verwendet. Zusätzlich zur Liste der Transaktionen werden einige weitere Metainformationen abgelegt.

Höhe Repräsentiert die Höhe des aktuellen Blockes. Eine explizite Deklaration an dieser Stelle erspart so die Berechnung der Höhe durch Traversieren und Zählen der Elternknoten.

MerkleRoot Hash Der resultierende Hash aus der Erzeugung des Merkle-Trees anhand der Liste der Transaktionen.

Vorgänger Blockhash Der Blockhash des vorhergehenden Blockes. Durch Verkettung wird hier die Quasi-Immutabilität der Blockchain erreicht. Änderungen an vorhergehenden Transaktionen propagieren durch alle Hashes bis zum letzten Block und invalidieren diese.

Anwendungshash Ein Hash über den Anwendungszustand, der von der Anwendung gesetzt werden kann, um weitere Zustandsmodifikationen zu verhindern.

Abstimmungen Eine Liste aller Stimmen, die für den aktuellen Block gestimmt haben. Für Proof-of-Work Algorithmen kann dies als Work- und Minerinformaiton verwendent werden und würde dann nur einen Eintrag enthalten, während diese in BFT-PoS eine Liste der Stimmen der Validatoren enthält.

Der Blockhash des Blockes ergibt sich durch die Hashwertberechnung der Werte aus Höhe, MerkleRoot Hash, Vorgänger Blockhash, Anwendungshash und Abstimmungsdetails.

7.2.5. Peer-to-Peer Umsetzung

Innerhalb des cadeia Prototypen wurden für verschiedene Anwendungsfälle zwei unterschiedliche Algorithmen beziehungsweise Netzstrukturen umgesetzt.

Grundsätzlich handelt es sich bei den nachfolgend vorgestellten beiden Varianten um reine Overlay Netzwerke, aufbauend auf dem durch Jadex bereitgestellten Netzwerk zwischen den Knoten. In Jadex können sich Knoten im selben Netzwerk direkt finden und verbinden, während über lokale Netzwerkgrenzen hinaus global bereitgestellte Relay-Server verwendet werden, um dies zu ermöglichen. Abbildung 7.6 verdeutlicht dies an einem Beispiel mit vier Knoten und einem Relay-Server. Applikationen kommunizieren transparent direkt miteinander, während intern die Aufrufe zwischen den Knoten oder über den Relay ablaufen.

Nachfolgend werden die beiden Varianten vorgestellt. Zuerst ein Overlay Netzwerk, welches unstrukturierte Netze für den Einsatz in öffentlichen Netzen abbildet und anschließend ein voll vermaschtes Netz, wie man es beispielsweise in innerbetrieblichen Strukturen finden kann.

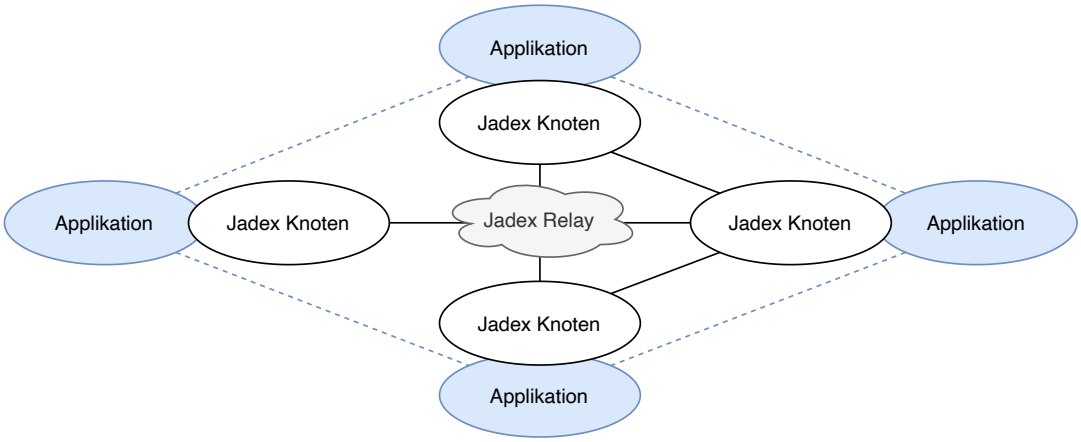

Abbildung 7.6.: Jadex-Netzwerk und Overlay

Cyclon

Heutzutage gibt es eine Vielzahl an effizienten Peer-to-Peer Overlay-Netzwerken. Um den Prototypen bereits mit einem lauffähigen Protokoll für verschiedene Einsatzzwecke auszustatten, wurde der *Cyclon*-Algorithmus ausgewählt [Voulgaris u. a. 2005].

Cyclon ist ein effizienter Algorithmus für ein leichtgewichtiges, unstrukturiertes Overlay-Netzwerk basierend auf *Gossip*-Nachrichten und permanentem *Shuffling* der Teilnehmerlisten. Durch zufälliges Ändern der Nachbarknoten wird hier versucht, gegen mögliche Störfaktoren, beispielsweise durch großflächige Ausfälle von Knoten, und gegen zentrale Verwaltung und globale vordefinierte Informationen vorzugehen.

Grundsätzlich führt jeder Knoten seine eigene Adressliste (Peerlist) von Nachbarknoten unabhängig von den anderen. Es ist nicht notwendig, dass beteiligte Knoten wechselseitig auf deren Adresslisten auftauchen, entsprechend gilt, dass Knoten A auf seiner Liste Knoten B haben kann, Knoten B muss aber nicht A auf seiner haben.

In festgelegten Abständen führt jeder Knoten unabhängig das *Shuffling* durch. Knoten A wählt dazu eine Teilmenge ℓ_A seiner Kontakte und sendet diese an einen zufälligen Knoten B. Knoten B empfängt diese Teilmenge und sendet eine Teilmenge ℓ_B (mit $\ell_B \leq \ell_A$) seiner eigenen Kontakte zurück an A. Knoten A übernimmt darauf die Kontakte aus ℓ_B in freie Positionen seiner Kontaktliste. Sollte diese bereits voll sein, werden Kontakte aus ℓ_A überschrieben. Durch dieses Verfahren werden in regelmäßigen Abständen mit zufälligen Kontakten die Adresslisten verändert und es sorgt für eine fortlaufende Neuvermaschung innerhalb des gesamten Netzes. Die Größe der Adressliste sollte von vornherein einheitlich gewählt werden und an die zu erwartende Netzwerkgröße angepasst werden.

Nachrichten werden in Cyclon per Gossip-Protokoll (oder *Epidemie-Protokoll*) verbreitet. Im Prinzip heißt das, dass jeder Knoten jede eingehende Nachricht an alle seine Nachbarn weiterschickt, worauf diese wiederum das gleiche tun.

Über einen *Hop-Count* kann kontrolliert werden, dass Nachrichten nicht unendlich weitergeleitet werden, indem für jeden Sprung (Hop) einer Nachricht zwischen zwei Knoten der Hop-Count verringert wird und beim Erreichen von 0 nicht mehr weitergeschickt wird. Dies kann zusätzlich verbessert werden, indem Knoten speichern, welche Nachrichten sie schon einmal an welche Kontakte weitergeleitet haben, sollte dieselbe Nachricht erneut bei ihnen eintreffen.

Full Mesh

Während für große und vor allem öffentlich eingesetzte Netzwerke ein unstrukturiertes P2P-Netzwerk mit Gossip-Protokoll von Vorteil sein kann, ist dies für firmeninterne oder beschränkte Netzwerke zwischen wenigen aber bekannten Teilnehmern nicht der Fall.

In kleineren Netzwerken ist es durchaus möglich, ein komplett vermaschtes Netzwerk aufzustellen, da die teilnehmenden Knoten alle bekannt sind. Weiterhin kann hier auf ein Gossip-Protokoll verzichtet werden, da eine Nachricht vom Sender direkt an alle Teilnehmer ohne sonderlichen Overhead verteilt werden kann. Dies führt zu deutlich kürzeren Nachrichtenlaufzeiten und potentiell zur schnelleren Abwicklung von Prozessen.

Durch die Nutzung von *Jadex* muss in der cadeia Blockchain keine weitere eigene Logik zur Herstellung des Netzwerkes und dem Pflegen von Nachbarlisten erfolgen. Ein Knoten kann über die Awareness-Komponente direkt die Kommunikationskomponenten der anderen Knoten auffinden und direkt die öffentlich verfügbare `receiveMessage(Object)` Funktion mit der Nachricht aufrufen. Ein Hop-Count oder ähnliches muss hier nicht verwendet werden, da jeder Knoten nur einen Sprung von jedem anderen Knoten entfernt ist und Nachrichten unmittelbar eintreffen.

7.2.6. Umsetzung des Exonum-Konsensalgorithmus

Für die Umsetzung eines Konsensalgorithmus eignet sich – basierend auf den bisher vorgestellten Eigenschaften der einzelnen Algorithmen (PoW, PoS, DPoS, usw.) und der Auswahl im Konzeptteil in Kapitel 6.2.3 – ein BFT-basierter Proof-of-Stake Algorithmus am besten. Hinzu kommt weiterhin, dass die Anforderungen der Geschäftsprozesse, beispielsweise sofortige Block- und Transaktionsfinalität, von entscheidender Rolle sind, die nur BFT-basierte Algorithmen erfüllen.

Ablauf

Der Exonum-Algorithmus ist ein BFT-basierter PoS Algorithmus, bei dem nur ein Subset an Knoten für den Konsens verantwortlich ist (Validatoren). Die anderen Teilnehmer (Nicht-Validatoren) nehmen nur „passiv“ am Konsens teil, durchlaufen aber dieselben Schritte.

Exonum selbst basiert auf Tendermint und setzt diesen zu großen Teilen identisch um, mit einigen kleinen Anpassungen. Der Ablauf von Tendermint ist bereits in Kapitel 3.3.3 skizziert worden. Zur Wiederholung, die grundlegenden Schritte beziehungsweise Phasen des Algorithmus sind Proposal, Pre-Vote, Pre-Commit und Commit. Dies passiert für jede Höhe h mit einer unbestimmten Anzahl an Runden r. Wird eine Phase nicht abgeschlossen, beginnt der Prozess von vorn mit derselben Höhe h, aber mit Runde $r + 1$. Erst bei erfolgreichem Durchlaufen mit abschließendem Commit wird die Höhe inkrementiert ($h + 1$) und die Runde wieder auf $r = 0$ zurückgesetzt. Die Zustimmung zu einer Phase wird durch das Signieren der betreffenden Nachricht oder des Blockhashes mit dem eigenen Schlüssel vorgenommen, eine Ablehnung erfolgt mit einer `null`-Nachricht.

Um innerhalb einer Runde von einer Phase in die nächste voranzuschreiten muss eine $+\frac{2}{3}$ Mehrheit für die aktuelle Phase erreicht sein. Dies passiert auf allen Knoten asynchron, sodass es sein kann, dass ein Knoten bereits gering-

fügig früher in die nächste Phase überwechselt. Anders als bei Tendermint haben Runden in Exonum kein Timeout.

In Tendermint kann es passieren, dass der Konsens fortlaufend fehlschlägt, weil beispielsweise die Nachrichtenübermittlung zulange dauert und so die Zeit für eine Runde überschritten wird. Beim Überschreiten der Rundenzeit wird wieder von vorn in der Proposal-Phase begonnen und die Runde auf $r + 1$ erhöht. In Exonum hingegen gibt es keine Rundenzeiten, die für einen Abbruch der aktuellen Runde sorgen. Die Rundenzeit terminiert hier nicht die aktuelle Runde, sondern sorgt dafür, dass quasi parallel eine neue zweite Runde mit $(h, r + 1)$ gestartet wird.

Sobald ein Knoten allerdings in einer Runde r eine $+\frac{2}{3}$ Mehrheit an Pre-Votes bekommen hat, sperrt (lock) er sich für diese und stimmt nachfolgend nur noch für diese Runde ab. Beim Empfangen eines neuen Proposals in Runde r' mit $r \neq r'$ sendet dieser Knoten sein Pre-Vote mit dem vorherigen Lock und vorherigen Proposal als sog. „Proof-of-Lock“. Andere Knoten können sich diesem nun nach und nach anschließen, wodurch es nun passieren kann, dass eine Runde $r = 1$ erst jetzt abgeschlossen wird, während schon Runde $r = 7$ gestartet wurde.

Anders als in anderen partiell-synchronen Konsensalgorithmen gibt es in Exonum also nur definierte Startzeiten, aber keine definierten Endzeiten von Runden.

Implementationsdetails

Bei der Implementation des Exonum-Algorithmus wird sich intern ein Automat mit festen Übergängen zu nutze gemacht. Da jede Phase im Prozess feste Bedingungen zum Übergang in die Folgephase oder Abbruchbedingungen zur Startphase hat, kann dies auch softwaretechnisch direkt so umgesetzt werden. Abbildung 7.7 zeigt nochmals die vier Phasen und deren Übergänge.

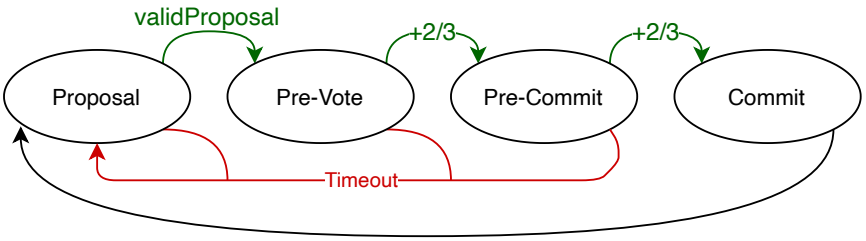

Abbildung 7.7.: Zustandsübergänge in Exonum

Um dies zu ermöglichen, wird der eigentliche Algorithmus (`ExonumConsensusAlgorithm`) mit einem Zustand und einer Nachrichten-Queue ausgestattet, wie in Abbildung 7.8 dargestellt. Eine einkommende Konsens-Nachricht wird in die MessageQueue gegeben, wo diese nach einer zum aktuellen Konsenszustand priorisiert und sortiert abgelegt wird. Der Konsensalgorithmus fragt nacheinander alle Nachrichten aus der Queue ab und verarbeitet diese. Die Sortierung innerhalb der Queue sorgt dafür, dass Nachrichten zur aktuellen Phase zuerst ausgegeben werden. Anschließend werden Nachrichten, die bereits der nächsten Phase angehören, betrachtet und Nachrichten, die nichtmehr verarbeitet werden müssen, werden gänzlich verworfen.

Der Konsenszustand (`ExonumConsensusState`) verwaltet alle Nachrichten innerhalb ihrer Phasen, um eine schnelle Überprüfung für einen Phasenwechsel zu gewährleisten. Er beinhaltet Listen zu aktuellen Proposals, Pre-Votes und Pre-Commits sowie aktuelle Höhe, Runde und gesperrten Proposals. So kann schnell verifiziert werden, ob die aktuelle Phase, beispielsweise Pre-Vote, bereits eine $+\frac{2}{3}$ Mehrheit erreicht hat, um zur nächsten überzugehen.

Der `ConsensusAgent` ist das Herzstück der Komponente. Über ihn wird der Konsens-Service angeboten und die benötigten Services, wie P2P und Persistenz, eingebunden. Er verknüpft also die internen Komponenten mit den benötigten Diensten der anderen Komponenten. Da die Konsens-Schnittstelle (siehe Abschnitt 7.2.3) sehr knapp gehalten ist, lässt sich der konkret implementierte Konsensalgorithmus sehr leicht durch einen anderen austauschen.

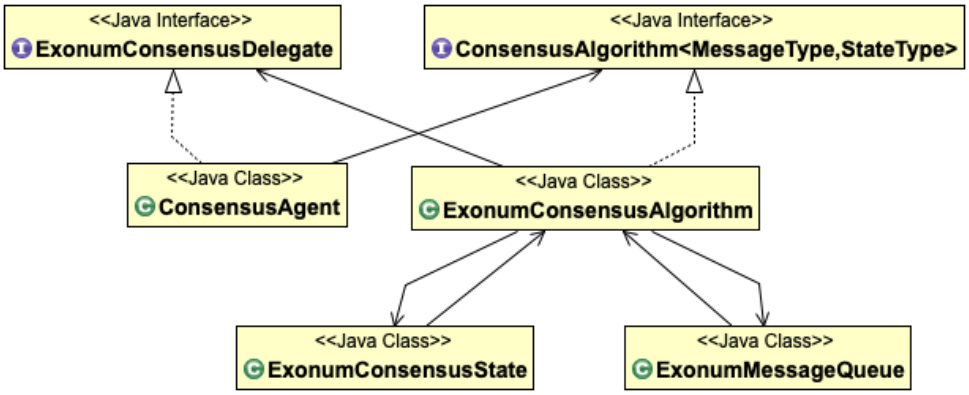

Abbildung 7.8.: Klassendiagramm: Konsenskomponente

7.2.7. Genesis-Block

Der Genesis-Block stellt in den meisten Blockchains den Startpunkt der Blockchain dar. Im cadeia-Prototypen kann dieser für globale Initialzustände verwendet werden, enthält aber auch Einstellungen und Informationen für netzweite Informationen.

Die konkrete Implementation ist stark auf BFT-PoS Algorithmen ausgerichtet, sie kann natürlich aber ersetzt werden. Neben den geerbten Feldern des Blockes (Transaktionen, MerkleRoot, usw.) enthält dieser, die für den Konsens notwendigen Timeout-Regelungen, Runden- und Proposal-Zeit sowie die Liste der öffentlichen Schlüssel, der anfänglich bestimmten Validatoren. Weitere Regeln betreffen die Konstruktionsregeln für Blöcke, wie etwa maximale Größe von Blöcken und Transaktionen in Bytes oder maximale Anzahl von Transaktionen pro Block. Diese Informationen sind, zusätzlich zu den bekannten des normalen Blockes, innerhalb des Blockhashes verankert, um sie gegen Manipulation abzusichern.

Eine Eigenheit im Genesis-Block dieser Implementation ist der `chainname`, welcher verwendet wird, um ein eigenes Netzwerk innerhalb von Jadex zu definieren. Während dies nicht per se eine Abschottung von anderen Jadex-Applikationen darstellt, minimiert es immerhin so die auffindbaren Dienste

und sorgt auch dafür, dass Dienste nur innerhalb dieses Netzes angeboten werden.

Besonders Validierungs- und Abstimmungsregeln im Genesis-Block sind für den Konsens wichtig, damit sich später alle Knoten nach den gleichen Regeln verhalten. Über die Liste der Validatorknoten kann eindeutig bestimmt werden, welcher Validator in welcher Höhe und Runde das Proposal vorschlagen muss, welcher der nächste Proposer sein wird und wann eine Runde terminiert und mit einer neuen begonnen wird.

7.2.8. Monitoringwerkzeuge im Prototyp

Aufgrund der Nutzung von Jadex als Middelware lassen sich noch weitere Funktionen, die über den normalen Anwendungsbereich des cadeia-Prototypen hinausgehen, umsetzen. So kann beispielsweise der Nachrichtenfluss und die Vermaschung der Knoten analysiert und visualiert werden, indem weitere Dienstschnittstellen in die Komponenten integriert wurden, wie in Abbildung 7.9 dargestellt. Hierbei werden die Knoten direkt angesprochen, also nicht über einen Nachrichtenaustausch mittels des Overlay-Netzes, um Informationen abzufragen. Mit diesem Werkzeug lässt sich über spezielle Peer-Nachrichten nachvollziehen, nach wie vielen Hops und über welche Knoten eine Nachricht bei einem Knoten angekommen ist. Ebenfalls lassen sich die auch ein- und ausgehenden Kanten (Nachbarn) eines Knoten darstellen.

Auf eine ähnliche Weise lässt sich auch ein „Blockchain-Explorer“ realisieren. Diese Funktionalität gibt es für viele Blockchains als Webinterface und wird meist dafür verwendet, um sich die Rohinformationen einer Blockchain anzuschauen, also Blöcke und Transaktionen. Da bei dem cadeia-Prototypen immer eine Anwendung zur Interpretation der Transaktionssemantik erforderlich ist, muss dies hier auch mithilfe der Anwendung geschehen. Blöcke und Rohtransaktionen lassen sich von beliebigen Knoten extrahieren und di-

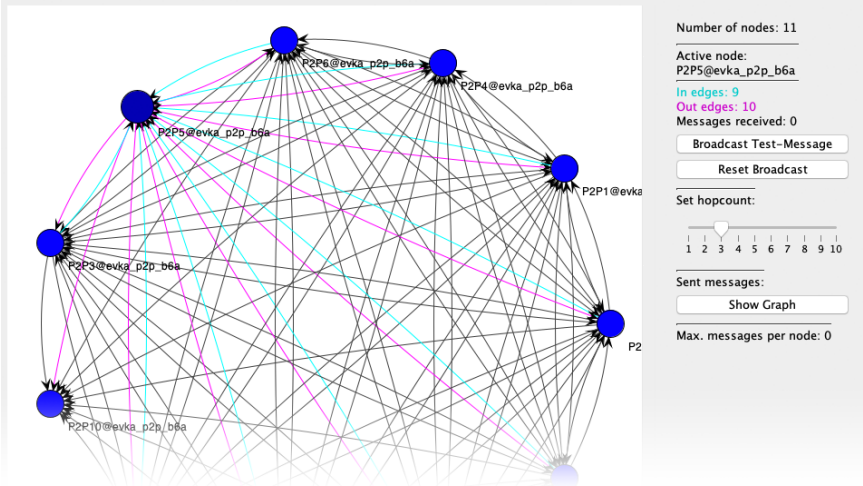

Abbildung 7.9.: Ausschnitt des Konnektivitäts User Interfaces

rekt darstellen, aber für eine semantikbasierte Darstellung wird ein optionaler Dienst der Anwendungsschicht eingebunden.

7.3. Blockchain-Erweiterungen

Um bei der bisherigen Vorstellung des cadeia-Prototypen noch weitere offene Punkte zu adressieren, werden nachfolgend die weiteren erforderlichen Zusatzfunktionalitäten aus dem Konzeptteil vorgestellt, um die Verwendung von Blockchains innerhalb von geschäftsprozessorientierten Anwendungen zu verbessern.

7.3.1. Anwendungsbedingte Transaktionsreihenfolge

Wie schon im Konzept in Abschnitt 6.2.4 erwähnt, ist es für viele innerbetriebliche Prozesse von essentieller Bedeutung, dass eine bestimmte Reihenfolge von Abläufen eingehalten wird und diese auch so innerhalb der Blöcke repräsentiert ist.

In anderen Blockchains, beispielsweise Bitcoin oder Tendermint, werden Transaktionen nach dem *First-In-First-Out* Prinzip in die Blöcke sortiert. Erstens ist die Reihenfolge absolut abhängig von der Eingangsreihenfolge der Nachrichten beim aktuellen Proposer (oder Miner) und zweitens besteht auch keine Eingriffsmöglichkeit in diesen Prozess durch die Anwendungsschicht.

Dies sorgt in vielen Fällen für abgelehnte oder invalide Transaktionen durch die vom Blockchain-Framework bereitgestellte Reihenfolge. Im cadeia-Prototypen gibt es für diesen Zweck eine explizite Rücksprache mit der Anwendungsschicht, bevor es zu einem Proposal kommt. Über die Funktion `validateProposal(List<Tx>):Future` wird die Anwendungsschicht beauftragt dieses zu validieren, bevor das Proposal verschickt wurde. Zusätzlich lassen sich die Transaktionen an dieser Stelle auch nach Anwendungssemantik durch die Anwendung selbst in eine bessere Reihenfolge bringen und über das Future zurückgeben.

Da es in einem BFT-PoS Algorithmus keine Transaktionsgebühr, wie in PoW das Trinkgeld, gibt, entstehen in kooperativen Szenarien auch keine Anreize für eine Applikation die Transaktionen in eine nur für sie vorteilhafte Reihenfolge zu bringen. Wenn dies dennoch stattfindet, besteht für die anderen Validatoren die Möglichkeit im normalen Konsensablauf dieses Proposal abzulehnen und so zum nächsten Proposer überzugehen.

7.3.2. Datenkorrekturverfahren

Die konkrete Umsetzung des Datenkorrekturverfahrens basiert auf dem skizzierten Konzept in Abschnitt 6.3.1. Da es im gewählten BFT-Konsensalgorithmus nicht zu Forks kommen kann, durch die gleichzeitig gültige Zustände erzeugt werden, muss dies hier absichtlich geschehen und in den implementierten Konsens und in die Anwendungsschnittstelle eingearbeitet werden.

Zuallererst muss eine Widerspruchsnachricht mit der für den Widerspruch nötigen Information angelegt werden. Dazu gehören vor allem der Block in dem die entsprechende Transaktion ist, die „schlechte“ Transaktion selbst und die Korrektur-Transaktion, die die schlechte ersetzt, wie in Algorithmus 7.3 dargestellt.

```
1   class DisputeMessage {
2
3       Hash disputeBlock; //block identifier
4
5       Hash badTransaction; // the transaction to replace
6
7       Transaction replacingTransaction; // new transaction
8
9       // hash adjustments to subsequent blocks
10      // old-hash -> new-hash
11      BlockChanges[] blockChanges;
12
13  }
```

Algorithmus 7.3: Auszug aus der Widerspruchsnachricht

Die Widerspruchsnachricht wird von der Anwendung angefertigt und über den normalen Weg an die anderen Knoten verteilt. Worauf diese nun die jeweiligen Transaktionen prüfen müssen. Die Überprüfung, ob die alte Transaktion und die neue nach der Kostenfunktion $Tx_{bad} < Tx_{replace}$ besser ist, kann leicht umgesetzt werden, was aber natürlich stark vom Anwendungskontext abhängt.

Die Überprüfung, ob dadurch keine Konflikte mit den nachfolgenden Transaktionen entstehen, ist komplexer. Dazu muss von der Anwendung an der Stelle, wo der originale Block angewendet wurde, ein zweiter Zustand Z_B abgezweigt werden. Im Zustand Z_B müssen nun alle Transaktionen ab der Stelle der Ersetzten neu validiert und sukzessiv auf diesen angewendet werden. Sollte der neue Zustand Z_B mit allen nachfolgenden Transaktionen valide sein, ist es empfehlenswert, diesen präventiv zwischenzuspeichern, da es bei erfolgreichem Widerspruchsverfahren zur Anwendung kommt und so ei-

ner neuen sukzessiven Transaktionsanwendung auf den Ursprungszustand nicht bedarf.

Die Liste der Hash-Wert-Änderungen muss nach erfolgreicher Validierung zusätzlich noch einmal sukzessiv durchgegangen werden. Dadurch, dass eine Transaktion in Block B_h geändert wurde, verändert sich der Root-Hash dieses Blockes, wodurch sich auch der gesamte Hash des Blockes B_{h+1} und alle weiteren darauffolgenden ändern.

Das Abzweigen eines Zustandes an einer bestimmten Blockhöhe h kann die Anwendung unter größere Probleme stellen, wenn dies nicht von vornherein bedacht wurde. Die einfachste Lösung ist, einen Snapshot des gesamten Zustandes nach jedem Block zu erstellen und diesen zu speichern. Eine andere zeitaufwendigere Methode ist die Rückabwicklung jeder Transaktion, um den Zustand zur Höhe h wiederherzustellen. Dazu kann jede Transaktion mit einer Funktion f und dazugehöriger Umkehrfunktion u ausgestattet werden, beispielsweise $f(k) : Konto_k + 500$ und $u(k) : Konto_k - 500$.

Während die Erstellung von Snapshots deutlich mehr Speicher erfordert, ist bei der Rückabwicklung der Rechenaufwand deutlich höher. Es sollte also bei der Anwendungsentwicklung dringend abgewogen werden, wie oft Widersprüche passieren und welche Regeln für diese gelten, z.B. nur die letzten x Blöcke.

Nachdem alle notwendigen Validierungen in den Knoten passiert sind und ein Proposer den Widerspruch im Konsens vorgeschlagen hat, muss diesem nach den Konsensregeln $+\frac{2}{3}$ der Validatoren zustimmen, bevor er als gültig anerkannt wird. Die Zustimmung erfolgt nicht wie im normalen Prozess durch Signieren des Blockes, sondern umfasst das Signieren aller neuen Blockhashes, da diese mit gültigen Signaturen in der Block-Datenbank des Frameworks abgelegt werden müssen.

7.3.3. Abhängigkeitsmodell für Squashing

Ein explizites Abhängigkeitsmodell der Transaktionen stellt den Grundstein für die als „Squash“ vorgestellte Operation zur Minimierung der gesamten Blockchain-Historie dar. Hierzu wird die Transaktionsklasse (aus 7.2) um ein Feld für eine Liste der Abhängigkeiten erweitert. Für Abhängigkeiten können die Hashwerte der Elterntransaktionen verwendet werden. Eine leere Liste bedeutet schlicht, dass keine Abhängigkeiten vorliegen. Weiterhin wurde der Kontext einer Transaktion hinzugefügt und die beiden neuen Felder in den Hashwert inkludiert. Die Erweiterung der Transaktionsklasse ist in Algorithmus 7.4 dargestellt.

```
1   class Transaction {
2
3       byte[] data;
4       Hash[] dependencies;
5       Context context; // ∅, sequence, sequence_end, ...
6
7       public byte[] getHash() {
8           return Hash.of(data, dependencies, context);
9       }
10  }
```

Algorithmus 7.4: Erweiterung der Transaktions-Klasse um Abhängigkeiten

Zusätzlich zur Möglichkeit Squashes durchzuführen, bieten explizite Abhängigkeiten auch Vorteile für den Widerspruchsmechanismus. Anstelle von Snapshots, die fortlaufend von der Anwendung erstellt werden müssen, können mit expliziten Abhängigkeiten nun vom Framework gezielt die Transaktionen ausgewählt werden, die zur Erstellung eines Abbildes des Gesamtzustandes benötigt werden. So können quasi Snapshots nebenbei erzeugt werden, sobald sie erforderlich sind und die Anwendung so etwas entlasten.

Für das Squashing selbst werden mittels Tiefen- und Breitensuche die untereinander abhängigen Transaktionen zu Graphen zusammengefasst. Für die Ausführung innerhalb der Anwendungsschicht ist es nützlich diese in eine geordnete Listenstruktur zu überführen, nämlich konkret nach aufsteigender

Blockhöhe und aufsteigender Transaktionsnummer innerhalb der Blöcke bei gleicher Blockhöhe. Dies erlaubt es später der Anwendung die Transaktionen auf einem leeren Zustandsobjekt sukzessiv anzuwenden, um den Zustand dieses Graphens zu erzeugen. Abschließend können von der Anwendung nun neue Transaktionen erstellt werden, die direkt aus neuen *Erzeugungstransaktionen* mit möglichst wenig Transaktionen den aktuellen Endzustand erzeugen.

Mithilfe des Widerspruchsverfahrens können nun die neu erstellten Transaktionsgraphen, konkret durch die Ersetzung von gesamten Blöcken, innerhalb der Blockchain-Historie ausgetauscht werden. Über eine Kostenfunktion der Transaktionen, die hier über die Anzahl der Transaktionen ($|G|$) definiert ist, kann hier schnell festgestellt werden, dass $|G'| < |G|$ gilt. Über die Anwendungshashes innerhalb der Blöcke kann weiterhin validiert werden, dass der höchste neue Block B'_{max} denselben Anwendungshash wie der ursprüngliche höchste Block B_{max} hat und somit für die Zustände $S_{G'} = S_G$ gilt.

Zusammenfassung

In diesem Kapitel wurde die exemplarische Realisierung eines generalisierten Blockchain-Frameworks auf Basis des zuvor in Kapitel 6 entwickelten Konzepts vorgestellt. Zuerst wurden mehrere unterschiedliche Softwareentwicklungsparadigmen vorgestellt und miteinander verglichen.

Als am besten geeignet stellte sich das Paradigma der Aktiven Komponenten heraus. Dieses vereint serviceorientierte mit agentenorientierten Architekturen, wodurch es sich für stark nebenläufige Anwendungen, die sowohl feste Strukturen als auch klar definierte Schnittstellen benötigen, hervortut.

Für die Ausführung einer Software, die nach dem Aktive Komponenten Paradigma entwickelt wird, musste weitergehend eine Ausführungsumgebung ausgewählt werden. Dabei eignete sich die Jadex-Middleware für die Realisierung, da sie alle erforderlichen Funktionen unterstützt und darüber hinaus

noch weitere agentenspezifische Funktionen und Modellierungs-Notationen aufweist, die hierfür nicht verwendet wurden, aber für darauf aufbauende Geschäftsprozesse durchaus nützlich sind.

Das entwickelte generalisierte Blockchain-Framework unterteilt sich, gemäß dem Konzept, in mehrere Kernkomponenten. Diese sind durch klare Schnittstellen voneinander getrennt und lassen sich so beliebig durch Komponenten mit anderen Funktionen austauschen. Durch die serviceorientierte Architektur konnte erreicht werden, dass es möglich wird, andere Konsensalgorithmen, Datenbanken oder Kommunikationswege zu verwenden, sollte es der damit umzusetzende Geschäftsprozess erfordern. Eine Aufteilung der verwendeten Komponenten auf mehrere Computer ist so auch denkbar, um Last zu reduzieren. Die lose Kopplung durch den Service-orientierten Ansatz ermöglichte so auch, bis auf wenige Ausnahmen, eine von der Anwendungsschicht unabhängige Funktionsweise des Frameworks. Lediglich zur Validierung der Transaktionen muss diese hinzugezogen werden.

Abschließend wurden die aus dem Konzept entwickelten erweiterten Blockchain-Funktionen zum Datenkorrekturverfahren, dem Ersetzen von Transaktionen und dem Reduzieren der Historie auf Grundlage des absichtlichen Forkens ebenfalls exemplarisch innerhalb des Frameworks umgesetzt.

Im nachfolgenden Kapitel 8 soll das vorgestellte Framework hinsichtlich seiner konkreten Funktionalität evaluiert werden. Dazu werden beispielhaft einige der Geschäftsprozesse aus vorgestellten Anwendungsszenarien exemplarisch für eine Blockchain-basierte Nutzung mit dem entwickelten Framework angepasst und umgesetzt, um die Erfüllung der Anforderungen und Herausforderungen aus Kapitel 5 aufzuzeigen.

8. Evaluation und Bewertung

Im Folgenden soll das vorgestellte Konzept und der darauf aufbauend entwickelte Prototyp eines generalisierten Blockchain-Frameworks *cadeia* qualitativ anhand einiger ausgewählter Anwendungsszenarien untersucht werden. Dabei wird der Fokus hauptsächlich auf die softwaretechnischen Aspekte der Umsetzung und der daraus resultierenden Anwendungsgebiete gelegt. Konkret werden mehrere Beispielszenarien aus dem Konzept- und Anforderungsteil dieser Arbeit erneut aufgegriffen und mithilfe des entwickelten Prototyps praktisch umgesetzt.

Die ersten zwei Beispiele widmen sich dabei den Prozessen der vorgestellten Anwendungsszenarien aus dem Konzeptteil. Hierzu wird die exemplarische Umsetzung der Prozesse aus dem Beispiel der Mitversicherungen (Kapitel 5.3) vorgestellt. Hierbei soll gezeigt werden, dass die aufgezeigten Herausforderungen bezüglich Sichtbarkeit der Daten, Validierung der Daten und Anonymität der Dateninhalte gleichzeitig gewährleistet werden können.

Als zweites Beispiel folgt das kurz vorgestellte Beispiel von virtuellen Kraftwerken (Virtual Power Plant) aus Kapitel 6.2.4. Hierbei soll aufgezeigt werden, wie kooperativ eine Lastverteilung und Fahrplan-Konsolidierung mittels Blockchain umgesetzt werden kann, deren Transaktionsreihenfolge auf Anwendungssemantik basiert und durch das Widerspruchsverfahren Änderungen für „bessere“ Transaktionen zulässt.

Im zweiten Abschnitt werden Zusatzkonzepte des cadeia-Frameworks, wie das Squashing der Blockchain-Historie, gezeigt und evaluiert. Dazu dienen eine minimalistische Kryptowährung und eine fiktive Lieferkette als Beispiel.

An diesen wird das Squashing (Kapitel 6.5.2) demonstriert, also die Möglichkeit der Reduzierung der Blockchain-Historie unter gleichbleibendem Anwendungszustand. Um das Squashing überhaupt zu gewährleisten, werden hier auch die expliziten Transaktionsabhängigkeiten aktiv genutzt.

Abschließend folgen zwei weitere Projekte an denen weitere Möglichkeiten eines generalisierten Blockchain-Frameworks aufgezeigt werden sollen. In vielen Industrieprojekten kommt früher oder später immer die Frage nach Smart Contracts auf, da diese vermeintlich als einzige Möglichkeit für eigene Programmlogik innerhalb von Blockchains verstanden werden. In cadeia werden Transaktionsinhalte immer durch eine Anwendungsschicht bestimmt, sodass eine Smart Contract Lösung hier nicht notwendig ist. Im Beispielprojekt soll aber gezeigt werden, wie auch hier eine Smart Contract Möglichkeit innerhalb cadeias geschaffen werden kann.

Letztlich werden die erzielten Ergebnisse in Hinblick auf die Anforderungen eines generalisierten Blockchain-Frameworks zusammengefasst und bewertet.

8.1. Umsetzung ausgewählter Anwendungsszenarien

Dieser Abschnitt widmet sich der Umsetzung zweier im Konzept skizzierter Anwendungsszenarien, an denen die notwendigen Funktionalitäten des Frameworks für eine prozessspezifische Umsetzung gezeigt werden können.

Zum einen demonstrieren die beiden entwickelten Prototypen die generelle Umsetzbarkeit der bereits in der Analyse vorgestellten Prozesse mit Hilfe des cadeia Prototypen. Die konzeptionelle Entscheidung bewusst auf festgelegte Transaktionsmuster und somit semantische Informationen im Framework zu verzichten, erlaubt hier die Umsetzung jeglicher auch komplexer Prozesse. Zum anderen aber zeigen die Prototypen auch die funktionalen Vortei-

le, die bei der Verwendung gegenüber anderen Frameworks entstehen, wie beispielsweise die Möglichkeit der semantischen Sortierung von Transaktionen im Blockbildungsprozess, was für viele Geschäftsprozesse erforderlich ist.

8.1.1. Prozessumsetzung "Mitversicherung"

Die Umsetzung des skizzierten Prozesses bei der Mitversicherung bei größeren Produkten erfordert es, dass Verträge, die darin beinhalteten Teilsummen innerhalb von Transaktionen in der Blockchain speichern. Aufgrund von Datenschutz und Geschäftsgeheimnissen sollen aber nicht alle Teilnehmer alle Inhalte eines Vertrages lesen können, geschweige denn, über die konkreten Summen der anderen Mitversicherer Bescheid wissen.

Wie bereits in der Prozessadaption in Abschnitt 5.3.5 beschrieben, muss für „Leserechte" auf ein hybrides Verschlüsselungsverfahren, bestehend aus asymmetrischer Kryptografie mit Public/Private-Keys und symmetrischer Kryptografie, gesetzt werden. Um die Geldsummen innerhalb eines Vertrages aufsummieren zu können und diese somit validierbar zu machen, aber diese trotzdem vor den Augen anderer Teilnehmer zu verbergen, kann hier auf Pedersen Commitments gesetzt werden.

Neben der hybriden Verschlüsselung DHIES hat sich ein weiteres Verfahren auf Basis elliptischer Kurven durchgesetzt, welches mit kürzeren Schlüssellängen und geringerem Rechenbedarf auskommt, das *ECIES (Elliptic Curve Integrated Encryption Scheme)* [Katz und Lindell 2014, S. 409]. Dieses kam in der Beispielumsetzung für die Mitversicherung zum Einsatz.

Der Konsortialführer kann die Inhalte des Vertrages sehen und ist damit für die Erstellung dessen verantwortlich. Damit entspricht die Vertragserstellung in diesem Szenario einer Erzeugungstransaktion im Blockchain-Sinn.

Ein für dieses Beispiel vereinfachter Vertrag beinhaltet in unverschlüsselter Form die Felder: Name, Gesamtsumme und die Liste der Teilnehmer und

deren Teilsummen. Die dazugehörige Transaktion muss hingegen über einige verschlüsselte Bereiche und Commitments verfügen, damit die Vertragsdetails verborgen bleiben. Der Vorteil der Nutzung von ECIES, abgesehen von den kleineren Schlüssellängen und der Geschwindigkeit, ist, dass dieselbe Kryptografiebibliothek auch für die Pedersen-Commitments verwendet werden kann, weil elliptische Kurven homomorphe Eigenschaften bezüglich der Addition besitzen.

Eine Vertragstransaktion muss also folgende Attribute mit bestimmten Inhalten besitzen:

Proofs Sind eine Liste von verschlüsselten Werten. Jeder Teilnehmer muss hier versuchen, alle Werte der Liste mit seinem privaten Schlüssel zu entschlüsseln, denn die Werte sind nicht adressiert oder in einer bestimmten Reihenfolge, um Anonymität zu gewährleisten. Sollte die Entschlüsselung eines Wertes gelingen, befindet der Teilnehmer nun im Besitz der Informationen über: seine Summe im Vertrag, den verwendeten Blinding-Faktor und den Vertragsnamen.

Commitments Stellen eine Liste von Commitments dar. Ein Commitment besteht aus der Summe S und dem Blinding-Faktor B und wird konkret mittels der Generatoren G und H wie folgt gebildet: $PC(B, S) = H * B + G * S$. Alle vier Werte und das Ergebnis PC selbst sind wiederum Punkte auf der gewählten elliptischen Kurve (*Secp256k1*).

Commitment Summe Die Summe der Commitments muss noch einmal separat als Wert innerhalb der Transaktion auftauchen, damit bei der Validierung auf einen bestimmten Wert geprüft werden kann.

Die Validierung eines Vertrages erfolgt anhand der Liste der Commitments und der ebenfalls in der Transaktion enthaltenen Commitment-Summe als Vergleichswert. Anhand der homomorphen Eigenschaft bezüglich Addition, lassen sich die einzelnen Commitments einfach addieren (hier muss die Addition auf elliptischen Kurven angewendet werden). Die Validierung der Commitment-Summe ist in Algorithmus 8.1 dargestellt.

```
1  public IFuture<Boolean>
2                validateTransaction(ContractTransaction contract) {
3    ECPoint sum = null;
4    for (ECPoint commit : contract.getCommitments()) {
5        if (sum == null) {
6            sum = commit;
7        } else {
8            sum = sum.add(commit); // Addition auf EC
9        }
10   }
11
12   return new Future<>(
13       Objects.equal(sum, contract.getCommitmentSum())
14       );
15 }
```

Algorithmus 8.1: Validierung von Vertragstransaktionen

Weiterhin kann von bestimmten Knoten noch zusätzlich jeweils ein Commitment selbst validiert werden, nämlich von Teilnehmern des Vertrages. Jeder Teilnehmer kann anhand seines „Proofs“ ein Commitment auf Vollständigkeit prüfen, denn in diesem ist die Summe und der verwendete Blinding-Faktor enthalten. Der Teilnehmer kann sein Commitment also selbst erzeugen und verifizieren, dass es sich in der Liste der Commitments innerhalb der Transaktion befindet.

Allerdings muss hier stark abgewogen werden zwischen dem direkten Widerspruch im Konsensverfahren (durch Enthalten oder Gegenstimmen) oder einem nachträglichen off-chain-Kontakt mit dem Konsortialführer. Ein Widerspruch im Konsensverfahren kann nämlich unter Umständen die eigene Identität verraten.

Eine weitere Notwendigkeit für viele Geschäftsprozesse ist die Verbindung von Transaktionsinhalten zwischen mehreren Transaktionen bei gleichzeitiger Geheimhaltung der Inhalte vor unbeteiligten Dritten. Werden Transaktionen Ende-zu-Ende verschlüsselt, sprich von Klient zu Klient, können nur diese selbst die Inhalte lesen und validieren. Auch eine etwaige Beziehung zwischen den Transaktionen ist nur den Beteiligten erkennbar und lässt sich nicht ohne Weiteres auf andere übertragen.

Für den Anwendungsfall der Mitversicherung bedeutet dies, dass die Vertragsbeziehungen aus mehreren unterschiedlichen Transaktionen nicht erkennbar sind. Was wiederum bedeutet, dass eine Minimierung der Überweisungen durch direktes Verrechnen aus mehreren Verträgen nicht möglich ist. Ohne eine dritte Instanz ist dies also erstmal nur zwischen zwei Teilnehmern direkt möglich.

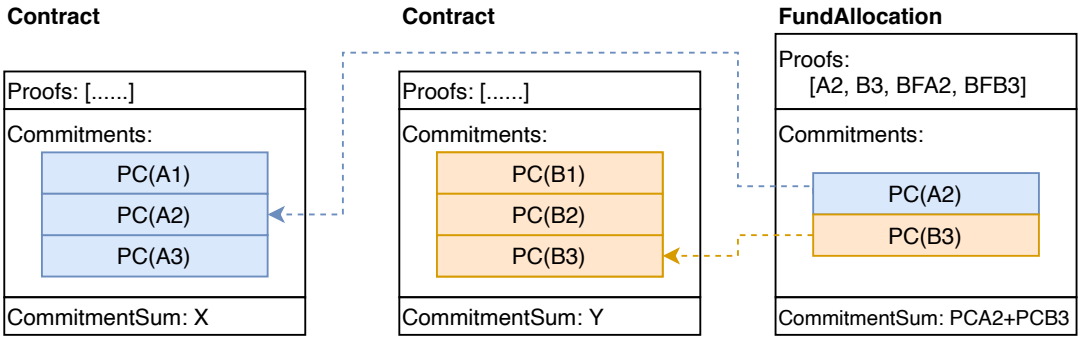

Abbildung 8.1.: Aufteilung der Summen aus zwei Verträgen

Für eine Aufteilung der Summen wird eine Transaktion erstellt, die die zu verrechnenden Commitments aus ausgewählten Verträgen beinhaltet. Dies erfolgt durch Referenzierung des Transaktions-Hashes und der Commitment-Nummer innerhalb der Transaktion. Um Kenntnis über den Inhalt der Commitments und Vertragswerte zu beweisen, müssen innerhalb des Proof Bereiches der Transaktion für beide Teilnehmer sämtliche Werte und zugehörige Blinding-Faktoren offengelegt werden (verschlüsselt für die beiden Parteien). Die Verrechnung erfolgt dann off-chain anhand der entschlüsselten Summen zwischen den Teilnehmern, die Commitments dienen hier nur der Verifikation. Analog zu UTXO-Transaktionen, aus beispielsweise Bitcoin, sind Commitments, die in einer Aufteilungstransaktion vorkommen, nun nicht mehr für andere Transaktionen verwendbar. Die Aufteilungstransaktion und die Referenzierung von Commitments sind in Abbildung 8.1 noch einmal beispielhaft für zwei Teilnehmer mit zwei Verträgen dargestellt.

Für am Vertrag und der Aufteilung unbeteiligte Knoten lassen sich analog zur Vertragstransaktion sämtliche Commitment-Werte in allen drei Transak-

tionen verifizieren, in dem Summen gebildet und verglichen werden, wie die folgende, basierend auf dem Beispiel aus Abbildung 8.1:

$$PC(A1) + PC(A3) + PC(B1) + PC(B2) = X + Y - PC(A2) - PC(B3)$$

Durch diese zwei Transaktionen mit unterschiedlichen „Sichtbarkeits“-Bereichen innerhalb der Transaktionen lassen sich also die Anforderungen aus dem Mitversicherungsszenario umsetzen.

Nun stellt sich abschließend die Frage, ob nicht auch ein bestehendes Blockchain-Framework, wie beispielsweise Ethereum für eine Umsetzung dieses Prozesses in Frage kommt.

Eine Smart Contract basierte Umsetzung erscheint auf den ersten Blick plausibel, da Ver- und Entschlüsselungen sowieso auf Anwendungsebene erfolgen und nicht Teil der direkten Transaktionsvalidierung sind. Auch die Validierung der Commitments erscheint trivial, da diese nur aus Summierung und Gleichheitsüberprüfungen besteht.

Dennoch scheitert eine Umsetzung aber sofort an den Laufzeitbeschränkungen von Smart Contracts. Für die Validierung der Commitments muss eine Addition auf elliptischen Kurven durchgeführt werden, was eben nicht einer Addition von Zahlen entspricht. Zusätzlich können nicht alle Punkte auf elliptischen Kurven überhaupt mit den primitiven Datentypen von Solidity dargestellt werden. Bei einer großen Menge von Commitments führt dies schnell zu exorbitanten Gaskosten. Die Smart Contracts könnten somit nur noch als Datenhalter fungieren, was deren Einsatz äußerst fragwürdig macht und die Validierung gänzlich aushebelt.

8.1.2. Prozessumsetzung “virtuelle Kraftwerke”

Die Möglichkeit Transaktionen basierend auf der eigenen Anwendungslogik zu sortieren und auszuwählen, statt beispielsweise rein chronologisch,

erlaubt es, weitere Geschäftsprozesse dezentral, mithilfe einer generalisierten Blockchain, umzusetzen. Anhand des folgenden Beispiels soll diese Möglichkeit dargestellt werden.

Das Szenario einer kooperativen Nutzung einer Blockchain zur Planung von Fahrplänen in virtuellen Kraftwerken (VPP) erfordert es beispielsweise, dass die teilnehmenden Knoten Einfluss auf die Transaktionsauswahl innerhalb des Blockbildungs- und Konsensverfahrens haben.

Der Prozess des Szenarios erfordert es, dass nur die beste Transaktion, bezogen auf eine gewählte Kostenfunktion, innerhalb des Blockes persistiert wird. Während die Berechnung der Erzeugungspläne (Optimierungsproblem) deutlich zeitaufwendiger ist, ist der Vergleich der Transaktionen anhand der Kostenfunktion trivial.

Ein Fahr-, Last- oder Erzeugungsplan für ein virtuelles Kraftwerk muss als Transaktion prinzipiell nur die VPP-Kennung, den anvisierten Zeitslot, beispielsweise im 15-Minuten Takt, und die Erzeugungsplan-Vektoren enthalten:

Kennung Eine eindeutige Kennung des virtuellen Kraftwerks.

Zeitslot Eine eindeutige Identifizierung des Zeitslots für die Lieferung.

Load-Vector Eine Liste V von Erzeugungsplänen (Vektoren) für jeden einzelnen Teilnehmer innerhalb eines VPP.

Gesamtkosten Die Gesamtkosten für die Ausführung dieses Erzeugungsplanes. Je nach Typ des Kraftwerkes können unterschiedliche Kosten für die Erzeugung entstehen. Die Gesamtkosten ergeben sich aus der Summe der Einzelkosten: $\sum_{n=0}^{|V|} = cost(V_n)$.

Um nun, nach den Anforderungen des Szenarios, nur eine Transaktion pro Zeitslot in der Blockchain-Historie zu speichern, kann innerhalb der Anwendungsschicht auf den Proposal-Teil des Konsensmechanismus eingegriffen werden.

Über die Funktion `validateProposal` wird die Anwendungsschicht bereits vor jeder Abstimmung vom aktuellen Proposer über den Vorschlag informiert und über dessen Gültigkeit befragt. Hier besteht direkt die Möglichkeit anhand der Kostenfunktion die beste Transaktion auszuwählen (Zeile #7 und #10) und alle anderen permanent für ungültig zu erklären (Zeile #11), wie in Algorithmus 8.2 vereinfacht dargestellt. Zusätzlich zur reinen Sortierung anhand der Vector-Kosten wurde im Beispiel zur Verdeutlichung des Konzeptes auf weitere Validierungen und Sortierungen, wie Überprüfung des aktuellen Zeitslots, verzichtet. Eine atomare Validierung der einzelnen Transaktionen muss an dieser Stelle nicht erfolgen, da sie bereits beim Hinzufügen zum Mempool und für die Erstellung des Proposals atomar validiert wurden.

```
1  public IFuture<ValidationResult>
2      validateProposalTransactions(List<LoadVector> transactions) {
3
4      transactions.sort((vect1, vect2)
5                  -> vect1.cost.compareTo(vect2.cost));
6
7      LoadVector optimalVector = transactions.remove(0);
8
9      ValidationResult validationResult = new ValidationResult();
10     validationResult.validTransactions = List.of(optimalVector);
11     validationResult.invalidTransactions = transactions;
12
13     return new Future<>(validationResult);
14 }
```

Algorithmus 8.2: Beispiel der semantischen Blockbildung

Wie bereits im Konzept in Abschnitt 6.3.1 beschrieben, kann es versehentlich dazu kommen, dass eine suboptimale Transaktion persistiert wurde, zum Beispiel durch Nachrichtenverlust auf dem Transportweg. Dies würde im VPP-Szenario dazu führen, dass ein Lastvektor mit einem schlechten Kostenwert persistiert wird.

Über die Widerspruchsnachricht kann im Konsens über die Ersetzung dieser entschieden werden. Da in diesem Szenario keine Abhängigkeiten zwischen den Transaktionen bestehen, kann die Verifikation an dieser Stelle trivial ausgeführt werden ohne auf weitere Zustände einzugehen. Es genügt nur die Gesamtkosten der beiden Transaktionen zu vergleichen, was sich leicht umsetzen lässt:

```
transactionNew.cost < transactionOld.cost
```

Ist die neue Transaktion basierend auf den Kosten besser, wird sie über den skizzierten Widerspruchsmechanismus im Konsens ersetzt. Ist sie hingegen schlechter, wird der Widerspruch ignoriert und verworfen.

Der Vergleich mit anderen Blockchain-Technologien ist an dieser Stelle schnell erledigt, denn es gibt derzeit schlicht keine anderen Frameworks, die eine anwendungsspezifische, beziehungsweise semantische Auswahl oder Beeinflussung des Proposals ermöglichen. Alle anderen Frameworks agieren entweder nach dem FIFO-Prinzip oder im Falle von Kryptowährungen nach egoistischen Prinzipien der Gewinnmaximierung durch Miner.

8.2. Evaluation der neuartigen Blockchain-Funktionalitäten

Innerhalb des Konzeptes in Kapitel 6 wurden zusätzlich zu den grundlegenden Funktionen, Komponenten und Prozessen einer Blockchain einige weitere Funktionen entwickelt, die besonders in geschäftsspezifischen Prozessen für bestimmte Anforderungen notwendig sind. Nachfolgend sollen deshalb an zwei auch konkret prototypisch umgesetzten Beispielen, die Funktionen zu Datenkorrekturverfahren (Abschnitt 6.3), die expliziten Abhängigkeiten von Transaktionen (Abschnitt 6.4) und der darauf aufbauenden wiederkehrenden Blockminimierung (Abschnitt 6.5) vorgestellt und aufgezeigt werden.

8.2.1. Squashing in Kryptowährungen

Anhand einer beispielhaften Kryptowährung soll hier gezeigt werden, wie die expliziten Transaktionsabhängigkeiten genutzt werden, um im cadeia-Framework die Squashing-Funktionalität zu nutzen. In diesem Beispiel werden also zu bestimmten Zeitpunkten Transaktionen zu Graphen zusammengefasst, damit diese in einen neuen Transaktionsgraphen zusammengeführt werden können, der weniger Transaktionen beinhaltet, aber denselben Anwendungszustand widerspiegelt.

In Anlehnung an Ethereum wurde ein vereinfachtes Account-Modell zur Repräsentation der Währung gewählt. In diesem Beispiel gibt es zwei verschiedene Arten von Transaktionen. Über eine Erzeugungstransaktion (`Create`) wird ein neuer Account mit einer beliebigen Menge der Währung angelegt und über eine Transfer-Transaktion (`Transfer`) können zwischen zwei Accounts Werte transferiert werden. Beide Transaktionen sind in Algorithmus 8.3 vereinfacht dargestellt.

Die Programmlogik innerhalb der Anwendungsschicht ist relativ simpel. Für jeden Account, der anhand einer Adresse identifiziert wird, muss ein aktueller Kontostand als Zustand vorgehalten werden. Jede Erzeugungstransaktion enthält eine Adresse des Accounts, beispielsweise den Hash des Public-Keys, und einen initialen Kontostand.

Innerhalb der Transfer-Transaktionen werden die Adressen des Start- und Zielkontos angegeben, zwischen denen die Währung transferiert werden soll. Eine Validierung kann hier auf die Deckung des Kontos reduziert werden. Die Anwendung kann nun bei jeder Transfer-Transaktion direkt den Zustand der beiden Accounts modifizieren. Sollten hier keine historischen Daten aufbewahrt werden, enthält die Datenbank nur die Accountinformationen und den Kontostand, ist also nicht besonders speicherintensiv. In der Blockchain-Historie hingegen, ist jeder Werttransfer eine eigene Transaktion und wird persistiert.

```
1   class CreateTransaction {
2       byte[] account;
3       long currency;
4       Hash[] dependencies  =  null;
5       Context context = SEQUENCE_START;
6   }
7
8   class TransferTransaction {
9       byte[] sourceAccount;
10      byte[] targetAccount;
11      long amount;
12      Hash[] dependencies  =  { sourceAccount.create.hash ,
13                                targetAccount.create.hash };
14      Context context = SEQUENCE;
15  }
```

Algorithmus 8.3: Auszug der Kryptowährungstransaktionen

Um nun dem Framework die Möglichkeit zu geben und zu erkennen, welche Transaktionen zusammengehören, werden für jede Transfer-Transaktion die beiden betreffenden Erzeugungstransaktionen als Abhängigkeit angegeben, denn die beiden Adressen (Start und Ziel) sind semantische Informationen aus der Anwendungsschicht, mit denen das Framework nichts anzufangen weiß.

Über die Abhängigkeiten der Transaktionen, identifiziert über deren Hash-Werte, kann das Framework nun einen Abhängigkeitsgraphen erstellen und diesen zur Anwendungsschicht weitergeben. Durch die Wahl des Modells und den Abhängigkeiten in diesem konkreten Beispiel werden ein oder mehrere bipartite Graphen erzeugt, wie in Abbildung 8.2 dargestellt. Wenn nur Transaktionen zwischen zwei Accounts stattgefunden haben, gibt es nur einen kleinen Graphen mit $Create_D$ und $Create_E$, sobald aber mehrere Transaktionen zwischen mehreren Accounts durchgeführt werden, hängen diese Accounts fortan in einem Graphen zusammen, auch wenn nicht direkt Werte zwischen diesen transferiert wurden. Im Beispiel hängen $Create_A$ und $Create_C$ durch gemeinsame Transaktionen mit $Create_B$ zusammen.

Für die Zusammenführung der Transaktionen und Vereinfachung der Graphen spielt die Art der Beschaffenheit aber eine größere Rolle. Die Anwen-

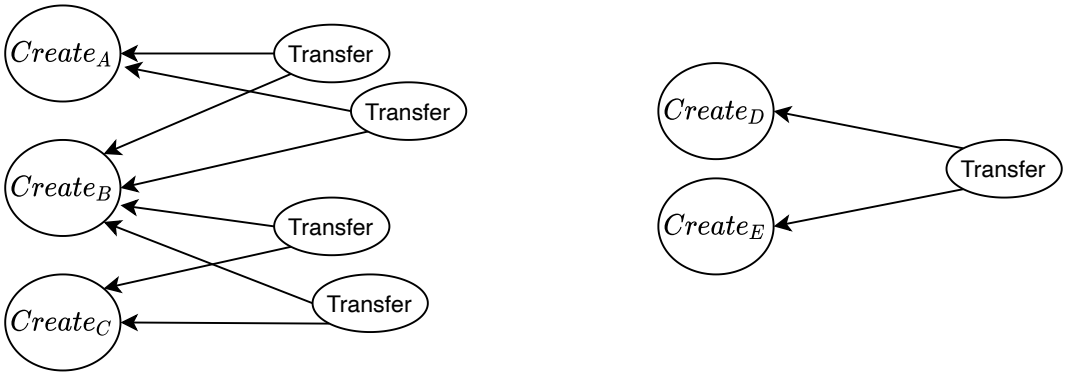

Abbildung 8.2.: Beispiel der erzeugten bipartiten Graphen in Währungen

dung sollte die Transaktionen im Graphen topologisch sortieren, sodass alle Erzeugungstransaktionen vor den Transfertransaktionen und alle Transfertransaktionen, in der Reihenfolge, in der sie auch in der Blockchain-Historie vorkommen, ausgeführt werden. In einem temporären Zustand werden nun die Kontostände der betroffenen Accounts neu berechnet und ein neuer Graph G' konstruiert. Dieser enthält nur noch Erzeugungstransaktionen der betroffenen Accounts mit den aktuellen Kontoständen, alle Werttransfers sind bereits im Kontostand reflektiert und werden verworfen.

Prinzipiell kann das Squashing in diesem Beispiel auch komplett ohne temporäre Zustände und das sukzessive Anwenden der Transaktionen im Graphen durchgeführt werden. Die Anwendung muss aus dem Graphen nur die betroffenen Accounts extrahieren, in seiner eigenen Datenbank deren Kontostand abrufen, hieraus neue Erzeugungstransaktionen erstellen und dem Framework zurückführen.

8.2.2. Squashing in Lieferketten

Das zweite Beispiel an dem der Squashing-Mechanismus gezeigt werden soll, ist ein weiteres in der Praxis beliebtes Anwendungsszenario für Blockchains: Lieferketten. In diesem Beispiel wird anhand von Start-, Transfer- und Endpunkten eine Lieferkette simuliert. Über Erzeugungs- und Transfertrans-

aktionen werden im System die einzelnen Stationen der Lieferkette repräsentiert und durchlaufen.

Die Erzeugungstransaktionen dienen der Repräsentation des Beginns eines Produktes, der erste Schritt in der Lieferkette. Alle weiteren Stationen werden durch Transfertransaktionen durchlaufen. Zur Simulation und Evaluation wurde sich auf eine maximale Lieferkettenlänge von 10 beschränkt. Das heißt wiederum, dass bei der 10. Transaktion (1 Erzeugung, 9 Transfers) die Lieferkette abgeschlossen ist und der Context der letzten Transaktion somit mit `sequence_end` festgelegt werden kann.

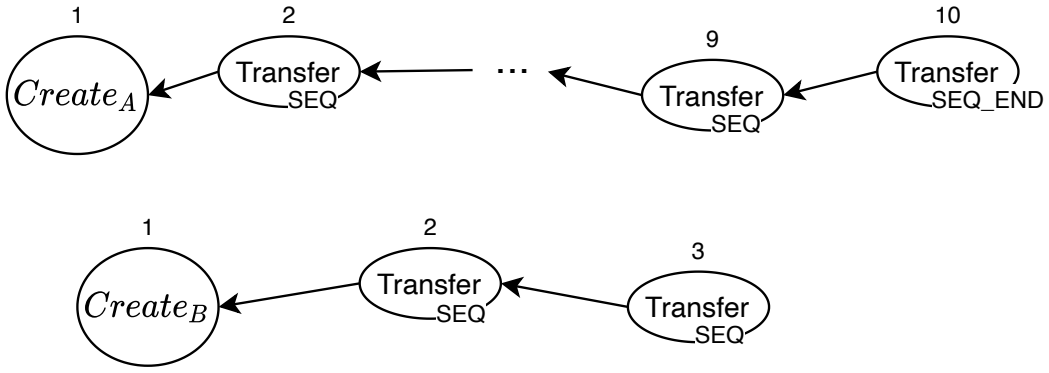

Abbildung 8.3.: Beispiel der erzeugten Graphen in Lieferketten

Anders als im Kryptowährungsbeispiel haben wir uns in diesem Szenario dazu entschieden, dass ein Squash nur durchgeführt werden kann, wenn eine Lieferkette komplett durchlaufen ist, also wenn eine Transaktion den entsprechenden Context enthält. Dies entspricht einer Lieferkette, bei der der finale Schritt das physische Produkt „konsumiert“, also nicht mehr existiert, oder dessen Repräsentation in der Lieferkette nicht mehr von Relevanz ist.

Für einen kompletten Squash der gesamten Blockchain-Historie muss das Framework also nur Transaktionen betrachten, die den Context `sequence_end` enthalten und die dazugehörigen Abhängigkeiten in Graphen zusammenzufassen. Alle Ketten, die noch nicht abgeschlossen sind, werden nicht weiter betrachtet. Im Beispiel in Abbildung 8.3 betrifft dies nur die erste Transaktionskette ausgelöst von $CREATE_A$, da diese von der 10. Transfer-

transaktion mit dem Context `SEQ_END` abgeschlossen wurde. In diesem Szenario entstehen auch keine zusammenhängenden bipartiten Graphen wie im Kryptowährungsbeispiel. Jeder Graph kann einzeln betrachtet werden. Die Anwendungsschicht entscheidet, wie weiter damit verfahren wird, was in diesem Szenario die vollständige Löschung des Graphens ist.

8.2.3. Evaluation der Blockchain-Minimierung durch Squashing

Um das Verhalten beider Szenarien zu evaluieren, wurden jeweils 100 pseudozufällige Durchläufe mit bestimmten Regeln durchgeführt. Bei beiden Szenarien wurde jeder Durchlauf auf jeweils 1000 Transaktionen beschränkt. Die Anzahl der Erzeugungs- und Transfertransaktionen variierte hierbei, je nach Szenario.

Im Kryptowährungsszenario wurden nur sieben Erzeugungstransaktionen verwendet. Die restlichen 993 Transaktionen sind zufällige Werttransfers zwischen den Accounts. Bei der Generierung der Transaktionen wurde nur darauf geachtet, dass die Accounts bereits erstellt wurden, sonstige Gültigkeitsregeln wurden nicht beachtet (beispielsweise ausreichend Guthaben).

In allen 100 Durchläufen wurde die gesamte Transaktionsmenge der Blockchain-Historie von 1000 auf 7 reduziert, was der Anzahl der erstellten Accounts entspricht. Dies entspricht auch dem erwarteten Verhalten, da nicht mehr als die initialen sieben Accounts entstehen können. Ebenso wurden alle Transaktionen sukzessiv auf die Kontostände addiert bzw. subtrahiert, sodass alle Wertetransfers abgearbeitet sind und keine mehr verbleiben. Weniger als 7 Transaktionen wurden nicht erwartet, weil keine Logik in der Anwendung enthalten war, die Accounts mit einem Kontostand von 0 löscht. Die maximale Reduktion $\Re$ der Blockchain-Historie, beziehungsweise der Transaktionen, ergibt sich also aus $\Re = \frac{|Transaktionen|}{|Accounts|}$.

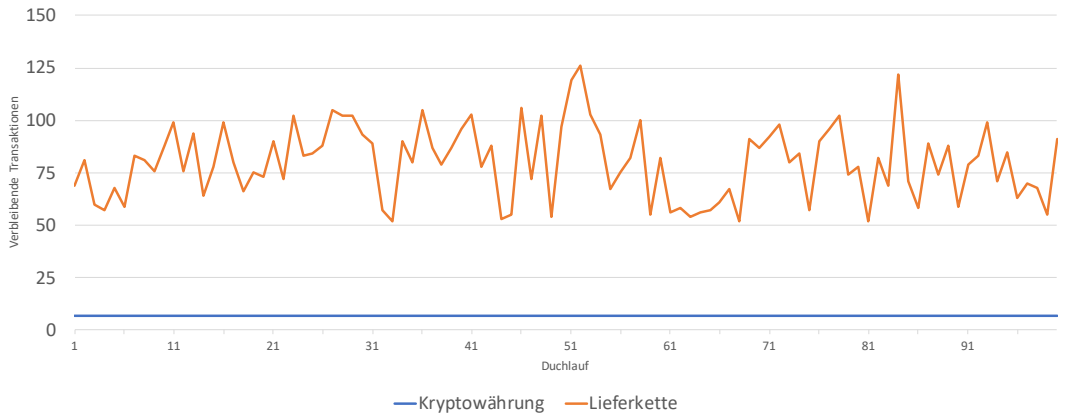

Abbildung 8.4.: Verbleibende Transaktionen nach Squash

Für das Lieferkettenszenario gibt es nur die Limitierung, dass sich nicht mehr als 15 unabgeschlossene Ketten gleichzeitig im System befinden und dass die 10. Transaktion einer jeden Kette immer den Context `sequence_end` enthält, um sie so abzuschließen. Transaktionen wurden nach diesen Regeln fortlaufend zufällig generiert.

Abbildung 8.4 zeigt den Verlauf der verbleibenden Transaktionen für beide Szenarien. Wie zu erkennen ist, fluktuiert die Reduktion im Lieferkettenszenario deutlich zwischen den Durchläufen. Da eine Transaktionskette nur entfernt wird, wenn sie 10 Transaktionen beinhaltet, bleiben diese am Ende des Squashingprozesses unverändert übrig. Die Reduktion auf durchschnittlich 73 Transaktionen entspricht einer Einsparung von 92,7% bei 1000 Transaktionen.

Die Einsparung in realen Lieferketten hängt natürlich von den jeweiligen Faktoren und Regeln bezüglich des Squashings ab. Beispielsweise könnten statt dem Löschen von abgeschlossenen Ketten unvollständige Ketten zusammengeführt werden, um Platz zu sparen. Die Evaluation zeigt aber, dass das Squashing Konzept generell nützlich ist, aber dessen Effektivität stark von den Szenarien abhängt.

8.3. Evaluation einer Smart Contract-Integration

Smart Contracts versprechen Programme innerhalb der Blockchain zu persistieren und diese dort ausführen zu lassen. Dadurch profitieren sie von den Blockchain-Eigenschaften wie Transparenz, Konsens und Unveränderbarkeit. Ziel dieser Evaluation ist die Möglichkeit, aufzuzeigen, auch in einem generalisierten Blockchain-Framework, wie cadeia, Smart Contracts innerhalb dieser bereitzustellen und auszuführen. Im nachfolgenden Abschnitt werden knapp nochmals die Eigenschaften von Smart Contracts wiederholt und ein Implementationsentwurf für eine Integration innerhalb des cadeia-Frameworks. Der Entwurf und die anschließende Evaluation soll zeigen, dass das generalisierte Framework nicht nur auf reine Geschäftsprozesse abzielt, sondern auch die Flexibilität besitzt, zusätzlich nachträglich durch Nutzer generierte Programme darin auszuführen.

Viele Startups und auch neue Umsetzungen bereits etablierter Geschäftsprozesse, die während der Hype-Zeit von Blockchains entstanden sind, fußten häufig auf der Implementierung von Smart Contracts (SC) für ihre Geschäftsprozesse. Die vermehrte Berichterstattung über Ethereum in dieser Zeit führte dazu, dass in vielen Projekten und auch Blockchain-Foren immer nach SCs gefragt wurde. Wie bereits in Kapitel 4.1.4 beschrieben eignen sich SC aber nur bedingt für die meisten komplexeren Geschäftsprozesse. Hinzu kommen noch weitere Einschränkungen bei der Entwicklung und Gestaltung des Quelltextes der SCs sowie bei der Interoperabilität mit anderen Systemen.

Beispielsweise gibt es in Ethereum-basierten SCs deutliche Einschränkungen zwischen der Abbildung von Domänenobjekten in SCs und dem Rest des Softwaresystems. In Java werden Domänenobjekte durch Klassen (bzw. *POJOs*) modelliert, was in SCs in der Solidity Programmiersprache mittels `structs` gewährleistet wird. Hierbei gibt es aber die Einschränkung, dass Funktionen des SC keine `structs` als Parameter entgegen nehmen können

oder als Ausgabe zurückgeben können [1]. Weitere Einschränkungen finden sich in der Anzahl der Felder in structs, bei zu vielen reicht der Platz im Stack nicht aus oder der Anzahl an Parametern von Funktionen. Um aus Java oder JavaScript mit SCs auf Ethereum-Blockchains zu interagieren, wird die Bibliothek *Web3j* bzw. *Web3js* verwendet, um zwischen SC und Anwendung zu übersetzen. Hier gibt es weitere Einschränkungen auf Grund der nicht ausgereiften Bibliothek. Während in Java Domänenobjekte beliebig tief geschachtelt sein können, ist die Tiefe in Web3j auf maximal 3 beschränkt.

Aus den in Kapitel 4.1.4 und oben genannten Einschränkungen wurde der cadeia-Prototyp explizit nicht auf Grundlage zur Ausführung von SCs entworfen. Es ist jedoch möglich, eine SC-Ausführungsumgebung als angeschlossene Applikation mit cadeia zu verwenden, was im folgenden ausgeführt wird.

8.3.1. Ausführungsumgebung

Um generell eine Möglichkeit zu schaffen, Smart Contracts, egal in welcher Form, zu integrieren wird eine Laufzeitumgebung für den von Nutzern produzierten Code benötigt. Am besten eignen sich hierfür Laufzeitumgebungen für Skript-Sprachen, wie JavaScript, da diese einfach zur Laufzeit interpretiert und ausgeführt werden, ohne dass der Nutzer sie kompilieren muss. Generell bietet sich hier JavaScript aufgrund seiner großen Verbreitung in der Webentwicklung und der Menge an Entwicklungstools an.

Aufgrund der Funktions- und Laufzeiteinschränkungen in Ethereum wurde bewusst auch hier darauf verzichtet, die Ethereum-VM (EVM) zur Ausführung von SC in cadeia zu integrieren. Stattdessen wurde die *GraalVM* zur Ausführung von JavaScript als Applikation in der Anwendungsschicht integriert, wie in Abbildung 8.5 dargestellt. Zusätzlich zur reinen Integration

[1] Seit Solidity-Compiler Version 0.4.17 gibt es eine experimentelle Funktion, die `structs` als Parameter ermöglicht. Ab Version 0.8 gilt diese nicht mehr als experimentell, muss aber explizit aktiviert werden.

der GraalVM gehören natürlich auch die Integration von Transaktionen, aus denen die SC geladen werden sowie einer Zustandsdatenbank für das Zwischenspeichern von aktuellen Zuständen, damit diese nicht bei jedem Aufruf aus der Blockchain-Historie erneut erzeugt werden müssen.

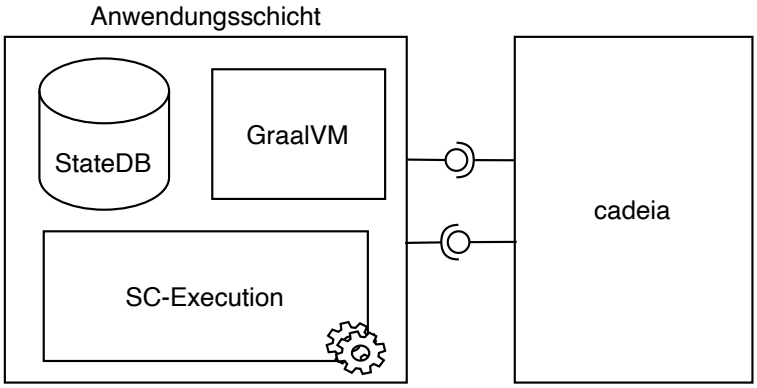

Abbildung 8.5.: Smart Contract-Ausführung als Anwendungsschicht

8.3.2. Konzept

Für die Ausführung eines JavaScript Smart Contracts müssen vorweg einige Funktionalitäten, Richtlinien beziehungsweise Regeln definiert werden, damit bei der Entwicklung und besonders bei der Ausführung von SCs keine negativen Nebeneffekte auftreten:

Turing-Vollständigkeit SCs sollen in der Lage sein beliebige Programme ausführen zu können. Endlosschleifen müssen allerdings unterbrochen werden.

Zustand Jeder SC hat seinen eigenen Zustand und kann diesen beliebig modifizieren. Dieser sollte möglichst effizient für mehrfach Abrufe vorgehalten sein.

Isolation Ein SC wird isoliert von seiner Umgebung ausgeführt. Er kann nicht die Zustände von anderen SCs oder der ausführenden Umgebung selbst verändern oder auslesen. Spezielle Werte wie die Blockhöhe oder

der Blockhash werden analog zu anderen SC-Laufzeitumgebungen bereitgestellt.

Im Speziellen müssen für die Ausführung innerhalb der GraalVM auch einige weitere JavaScript-Funktionen „deaktiviert“ werden. Beispielsweise das Erzeugen von Java-Objekten innerhalb der Ausführungsumgebung und Funktionen über die Zufallswerte generiert werden können, wie `Math.random` oder `Date`- und `Time`-Funktionen. Durch das Erzeugen von Java-Objekten entstehen Möglichkeiten aus der GraalVM auszubrechen und Zufallswerte zerstören die zwingend notwendige deterministische Ausführung der SCs.

Um Endlosschleifen zu vermeiden, verwendet Ethereum als Laufzeitbeschränkung *Gas* (dt.: Benzin). Jede Aktion verbraucht Gas und sobald dieses aufgebraucht ist, terminiert der SC. Für die Evaluation wurde sich für eine simple zeitliche Restriktion der Ausführung entschieden und auf Ausführungskosten verzichtet. Ohne Gas stehen somit den SCs in dieser Implementation auch theoretisch ein unbegrenzter flüchtiger und persistenter Speicher zur Verfügung.

Für die Zustandsmigration und Persistierung nach der Ausführung wird der globale Variablenspeicher des SCs aus dem JavaScript-Quelltext serialisiert und in die *StateDB* geschrieben. Vor der Ausführung wird dieser eingelesen und wieder in den Quelltext injiziert. Für eine performantere und referenzbasierte Serialisierung, die auch zyklische Referenzen in JavaScript unterstützt, wird auf den Algorithmus von Oh u. a. 2015 zur Zustandsmigration in Web-Anwendungen zurückgegriffen.

8.3.3. Funktionale Parität

Für die Feststellung der Parität der Funktionalitäten zwischen weit verbreiteten Smart Contracts und dieser Beispiel-Implementation wurde der in Ethereum standardisierte SC *ERC20* ausgewählt. Der *ERC20* Contract modelliert

einen Token, also eine Art Währung, der fast ausschließlich für *Initial Coin Offerings* verwendet wird.

Grundliegende Funktionen des ERC20-Tokens sind das Speichern des Kontostandes und das Transferieren von Werten zwischen zwei Accounts innerhalb des Contracts. Mit diesen relativ simplen Funktionen lässt sich besonders gut ein erster Vergleich zwischen den Solidity- und den GraalVM-basierten SCs zeigen. Im Anhang sind in den Algorithmen A.1 und A.2 Ausschnitte der beiden ERC20-Implementationen zu sehen. Abgesehen von einigen kleineren sprachspezifischen Anpassungen, kann man hier sehen, dass sich die SCs nicht unterscheiden. Lediglich Typenüberprüfungen sollten in der JavaScript-Variante vorgenommen werden, da es eine typenlose Programmiersprache ist, wie in Algorithmus A.3 Zeile # 9-10 dargestellt. Grundsätzlich lässt sich mithilfe des ERC20-Tokens bereits eine Parität bei den Funktionalitäten zwischen Solidity- und den hier entworfenen JavaScript SCs zeigen.

Weiterhin profitieren die JavaScript SC in diesem Falle von den unzähligen weiteren Frameworks und Bibliotheken aus dem Webbereich. So können spezielle Funktionen einfach in den Contract importiert werden. Da es keine Instruktions- und Speicherbeschränkungen gibt, lassen sich beispielsweise auch Sortieralgorithmen innerhalb der SCs praktisch einsetzen, anders als es in Ethereum beispielsweise der Fall ist (siehe Kapitel 6.1). Auch der Umgang mit Objekten innerhalb von JavaScript ist deutlich komfortabler und mit weniger Einschränkungen behaftet als es die structs in Solidity sind. Dies erlaubt eine deutlich bessere Übertragung der Domänenobjekte in Repräsentationen innerhalb der SmartContracts. Ebenso unterliegen Funktions-Schnittstellen keinerlei Einschränkungen und können beliebige und vor allem beliebig viele Parameter beinhalten.

8.3.4. Laufzeitvergleich

Um einen abschließenden Vergleich der beiden SC-Varianten zu ermöglichen, müsste zusätzlich abschließend noch ein Laufzeitvergleich durchgeführt werden. Hierauf wurde aber verzichtet, da sich schon aus der Konzeption der JavaScript SCs in cadeia einige Nachteile ergeben.

Zum einen werden Ethereum-Contracts zu Bytecode kompiliert, was es der EVM erlaubt, diese direkt auszuführen. Zum anderen werden die Zustände der SCs von der EVM in die Ausführungsumgebung abgebildet, sodass die Registerzugriffe des kompilierten Bytecodes in die dahinterliegende Datenstrukturen passen und in-situ verändert werden können.

Im JavaScript SC liegt hier kein Bytecode vor, da es sich um eine Skriptsprache handelt, die nicht vorkompiliert wird. Das führt dazu, dass bei jedem Zugriff auf einen Smart Contract der Quelltext neu eingelesen und jedesmal zur Laufzeit von der GraalVM neu interpretiert werden muss. Der andere Nachteil ist der Speicherzugriff. Die Zustände müssen aus der Datenbank gelesen, dann deserialisiert werden und anschließend in den SC injiziert werden, was zusätzliche Zeit in Anspruch nimmt. Nach der Ausführung muss der Zustand wieder serialisiert und in die Datenbank zurückgeschrieben werden.

Aus diesem Grund wurde auf einen Laufzeitvergleich verzichtet. Bei gleicher Ausführungsgeschwindigkeit der eigentlichen Funktionen wären JavaScript SCs allein durch die Vor- und Nachbearbeitung der Daten langsamer. Generell dient dieses Konzept und der Prototyp nur der Verdeutlichung der Möglichkeit einer Smart Contract Schnittstelle innerhalb von cadeia. Die Ausführung von nativem Code als angeschlossene Applikation ist im Allgemeinen die bevorzugte Variante, da sie deutlich komplexere Konzepte unterstützt und wesentlich performanter ist als beispielsweise Ethereum SCs.

8.4. Zusammenfassung und Bewertung

In diesem Kapitel wurde die Evaluation des entwickelten Konzeptes und der prototypischen Umsetzung, wie sie im Rahmen des Projektes cadeia anhand mehrerer beispielhafter Anwendungen durchgeführt wurde, beschrieben. Hierbei sollte zuerst aufgezeigt werden, dass die aufgestellten Anforderungen für Geschäftsprozesse aus Kapitel 5 an ein Blockchain-Framework erfüllt wurden. Dazu wurden exemplarisch die Prozesse von zwei der vorgestellten Beispiele umgesetzt. Danach wurden die zusätzlich konzipierten Funktionalitäten für ein Framework anhand zweier weiterer Use Cases vorgestellt, beispielhaft umgesetzt und abschließend evaluiert. Darüber hinaus wurde abschließend gezeigt, dass sich der entwickelte Prototyp auch prinzipiell für den Einsatz als Smart Contract Framework eignet, wenn auch mit gewissen Abstrichen.

Der erste Teil der Evaluation sollte aufzeigen, dass das vorgeschlagene Konzept und der darauf basierende Prototyp für eine Blockchain-basierte Umsetzung der vorgestellten Geschäftsprozesse geeignet ist. Die beiden Beispielszenarien zeigen, wie einerseits die Generalisierbarkeit durch frei wählbare Transaktionsinhalte möglich ist und deren Validierbarkeit weiterhin gewährleistet werden kann. Konkret wäre die Umsetzung auf SC-basierten Blockchains, wie Ethereum, aufgrund der hier gewählten Transaktionen mit enthaltenen Commitments technisch nicht umsetzbar. Andererseits zeigen sie aber auch, dass durch eine Kopplung mit der Anwendung direkte Auswirkungen auf innere Blockchain-Prozesse, wie die Blockbildung und Sortierreihenfolge, möglich sind, was eine Integration in weitere Anwendungsszenarien erlaubt.

Der zweite Teil der Evaluation beschäftigte sich mit den notwendigen Prozesserweiterungen innerhalb der Blockchain, den expliziten Transaktionsabhängigkeiten und dem Squashing. Beide sind neuartige Konzepte, um den Umgang mit einem Blockchain-Framework aus Sicht der Geschäftsprozesse zu verbessern. Hierbei wurde gezeigt, dass sowohl Kryptowährungen als

auch klassische Prozesse, wie Lieferketten, von diesen Konzepten profitieren können. Hierzu wurden zwei Prozesse modelliert und exemplarisch umgesetzt. Zur qualitativen Evaluation wurden mehrere Testdurchläufe vollzogen, um die generelle Funktionalität und auch den Nutzen der Reduktion der Blockchain-Historie aufzuzeigen. Während die Reduktion innerhalb der Kryptowährung deutlicher ausfällt (99,3%), erreicht die Lieferkette nicht dasselbe Potential, was den Prozessrestriktionen geschuldet ist. Dennoch wird hierbei gezeigt, welche möglichen Einsparungen hier und in anderen Geschäftsprozessen zu erzielen sind. Im Allgemeinen kann man sagen, dass in vielen Prozessen eine Reduktion der Größe möglich ist, wenn es zwischen Start und Endzustand Übergänge gibt, die nicht mehr zur Wiederherstellung des Gesamtzustandes oder zwecks Protokollierung benötigt werden. In einigen speziellen Fällen kann sogar eine massive Reduktion der Größe erfolgen, wie beispielsweise bei Kontoständen.

Abschließend wurde zur Vergleichbarkeit mit Smart Contract-basierten Frameworks, wie Ethereum, eine JavaScript-basierte Skriptumgebung konzipiert, und mittels der GraalVM in das Framework integriert und anschließend evaluiert. Mit dieser können nutzergenerierte Programme zur Laufzeit der Blockchain darin gespeichert und ausgeführt werden.

Hierbei konnte gezeigt werden, dass sich das im Rahmen der Arbeit vorgeschlagene Konzept generell hierzu eignet, was auch exemplarisch mittels des cadeia-Prototyps validiert wurde. Die Ethereum-Funktionalitäten können nahezu 1-zu-1 umgesetzt werden, aber zusätzlich um viele weitere aus dem JavaScript- und Webentwicklungsbereich erweitert werden. Die Ausführungsgeschwindigkeit der SCs ist in dieser exemplarischen Umsetzung nicht vergleichbar mit der von SCs in Ethereum.

Einerseits liegt das an der Ausführungsumgebung (GraalVM), da hier Zustände zwischen der Anwendung und der VM hin und her transferiert werden müssen, damit darauf zugegriffen werden kann. Andererseits liegt es auch an den Eigenschaften von Skriptsprachen, da diese unmittelbar vor der Ausführung übersetzt werden müssen, was zusätzliche Zeit benötigt, und

nicht direkt als Maschinencode vorliegen. Hier gibt es durchaus Potential zur Verbesserung, wenn dabei auf vorkompilierten Programmiersprachen mit spezifischen Ausführungsumgebungen zurückgegriffen wird. Eine auf SCs optimierte Ausführungsumgebung, wie es die EthereumVM ist, mit einer minimalen Instruktionsmenge und direkten Speicherzugriffen auf die relevanten Blockchain-Daten bietet hier deutliches Optimierungspotential. Hierdurch entsteht allerdings wieder ein Trade-off zwischen Ausführungszeit und Mächtigkeit der Programmiersprache.

9. Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Konzept für die softwarebasierte Unterstützung dezentraler Geschäftsprozesse in B2B-Anwendungen mithilfe der Blockchain-Technologie entwickelt. Abschließend sollen die Inhalte der Arbeit noch einmal kurz zusammengefasst sowie die erzielten Ergebnisse diskutiert und reflektiert und ein Ausblick für weiterführende Arbeiten gegeben werden.

Für die Konzeption und Entwicklung verteilter Prozesse bedarf es sowohl einer fachlichen geschäftsorientierten wie auch einer technischen Betrachtung dieser. Technische Workflows und fachliche Geschäftsprozesse lassen sich mit einer Vielzahl von Modellierungssprachen beschreiben und mit dazugehörigen Ausführungsumgebungen auch direkt (teilweise mit Zwischenschritten) technisch umsetzen. Mit Geschäftsprozess- und Workflow-Management-Systemen lässt sich deren Ausführung von zentraler Stelle aus beobachten und überwachen. In der heutigen Zeit, die stark durch Globalisierung räumlich getrennten Liefer- und Wertschöpfungsketten geprägt ist, entsteht so eine zentralisierte Abhängigkeit von einigen wenigen Akteuren im Prozess, was durch die Zentralisierung der Server- und Cloud-Infrastrukturen durch wenige Anbieter, wie beispielsweise Amazon oder Google, noch weiter vorangetrieben wird. Das ist aber in vielen Fällen äußerst problematisch, sodass hier auch geeignete dezentrale Lösungen zu bevorzugen sind - zumindest soweit sie auch die notwendigen Anforderungen an Koordination, Vertrauen, Sicherheit und Geheimnisschutz erfüllen.

Generell sollten in verteilten Systemen, bei denen für einen fehlerfreien Ablauf mit mehreren Teilnehmern garantiert werden muss, geeignete Konsens-

algorithmen verwendet werden, die bei bestimmten Ausfall- oder Angriffsszenarien einen gültigen systemweiten Zustand unter den fehlerfrei agierenden Teilnehmern garantieren. Bei den klassischen verteilten Konsensalgorithmen zur Ausfallsicherheit (Crash-Fault-Tolerance) und byzantinischen Fehlertoleranz (Byzantine-Fault-Tolerance) nehmen jedoch auch einzelne (ggfs. dezentrale) Komponenten eine wichtige Rolle im Gesamtsystem ein.

Sowohl bei der Durchführung von verteilten Geschäftsprozessen als auch in der Ausführung von verteilten Workflows kann meist (i.d.R. kostenpflichtig) auf externe Dritte zurückgegriffen werden, um beispielsweise im Falle von konkurrierenden Teilnehmern für Vertrauen zu sorgen. Sowohl technisch als auch fachlich entsteht so der Bedarf für einen sicheren und vertrauensvollen Mechanismus, der Konsens garantiert und unabhängig von Dritten ist. Nur so kann eine Dezentralisierung der Prozesse erreicht werden und es können weiterhin auch die entstehenden Abhängigkeiten (inklusive Kosten) durch externe Intermediäre effektiv reduziert werden.

Aus einer solchen Motivation heraus, zentrale Intermediäre aus eigentlich verteilten Systemen auszuschließen, entstand die Idee der Verwendung der Blockchain-Technologie zur dezentralen Realisierung der notwendigen Koordination. Die Blockchain-Technologie ermöglicht es erstmals, in verteilten Systemen unabhängig von Dritten mit untereinander unbekannten Teilnehmern für gemeinsame gültige und von allen akzeptierte Systemzustände zu sorgen. Um dies technisch umzusetzen, wird dabei auf einen verteilten Konsensalgorithmus gesetzt, der Glaubhaftigkeit durch das Lösen eines kryptografischen „Rätsels“ erzeugt. Die Blockchain-Technologie schafft durch ihre verkettete Datenstruktur zusätzliche Sicherheit in Form von nachträglicher Unveränderbarkeit. Prüfsummen sorgen dafür, dass eine Veränderung auf Transaktions- oder Blockebene alle nachfolgenden Werte invalidiert und bis in den aktuellsten Block kaskadiert und somit auch kleinste Veränderungen der in der Blockchain gespeicherten Informationen niemals unentdeckt bleiben.

Viele der frühen Blockchain-Technologien haben als Hauptanwendungszweck den Werttransfer in Form von sogenannten Kryptowährungen und sind somit auch technisch darauf optimiert. Ebenso fokussieren darauf bauende Weiterentwicklungen mit neuen Funktionen oder besseren und schnelleren Konsensalgorithmen meist digitale Währungen als einzigen Anwendungszweck. Mit Ethereum entstand die zweite Generation der Blockchain-Technologien, die es erstmals auch ermöglichte, verteilte Codeausführung mit Hilfe von Smart Contracts zu gewährleisten. Smart Contracts erweitern den Funktionsumfang und die möglichen Anwendungsgebiete enorm. Sie sind aber beispielsweise durch Laufzeitbeschränkungen in ihren Möglichkeiten weiter eingeschränkt, sodass sie sich, vor allem aufgrund ihrer Komplexität, im Allgemeinen nicht für Geschäftsprozesse eignen.

Betrachtet man die Blockchain-Technologien aus softwaretechnischer Sicht, wird schnell deutlich, dass diese Technologien sich nicht wesentlich in ihren Komponenten und Abläufen unterscheiden. Auf Architekturebene sind sie häufig in Komponenten, wie etwa Persistenz, Kommunikation, Konsens und Mempool aufgeteilt. Wobei je nach Technologie die einzelnen Komponenten etwas mehr oder weniger stark voneinander getrennt sind, was es teilweise unmöglich macht sie zu entzerren und Teile für den flexiblen Einsatz in Geschäftsprozessen umzuwidmen. Eine lose Kopplung erlaubt hier potentiell ein flexibleres Austauschen von Komponenten für andere Anwendungszwecke und sollte deshalb für ein generalisiertes Blockchain-Framework bevorzugt werden.

Weiterhin unterliegen verteilte Geschäftsprozesse speziellen Herausforderungen in Bezug auf deren technische Umsetzung. Bei der Umwandlung auf eine Blockchain-basierte Lösung entstehen neue Herausforderungen dadurch, dass ein Intermediär als vertrauensvolle Instanz fehlt. Bei der technischen Umsetzung von Geschäftsprozessen mittels offener Blockchain-Systeme müssen entsprechend notwendigerweise Teile des Prozesses adaptiert werden, damit sie den besonderen Ansprüchen von Geschäftsprozessen genügen, beispielsweise hinsichtlich der Zugangsbeschränkungen, Sichtbarkeit und Sicherheit. Zusätzlich zu den generellen Anforderungen an Geschäftsprozessen

kommen je nach Anwendungsszenario noch weitere prozessspezifische Anforderungen, wie etwa Blockzeiten oder Netzwerkstrukturen, hinzu.

9.1. Beitrag zur Forschung

Zur Umsetzung verteilter Geschäftsprozesse mithilfe der Blockchain-Technologie bedarf es eines softwaretechnischen Entwurfs, welcher die besonderen Anforderungen von Geschäftsprozessen von vornherein berücksichtigt. Basierend auf diesen Herausforderungen wurde im Rahmen der vorliegenden Arbeit ein Konzept für ein generalisiertes Blockchain-Framework entwickelt, um verschiedenartige, vor allem verteilte, Geschäftsprozesse damit umsetzen zu können. Wohldefinierte Schnittstellen zwischen den Komponenten innerhalb des Frameworks sorgen für eine Austauschbarkeit dieser, um beispielsweise ggfs. auch andere Konsensalgorithmen dabei verwenden zu können. Klar definierte Transaktionsphasen sorgen weiterhin für eine ständige Validität und Konformität, sowohl syntaktisch innerhalb des Frameworks als auch semantisch innerhalb des Geschäftsprozesses.

Weitere Funktionalitäten erweitern das Konzept für eine noch bessere Integration von Geschäftsprozessen. So erlaubt beispielsweise die Einführung der Longest-Chain-Rule in einen BFT-PoS Algorithmus das konsensbasierte Neuschreiben der Historie für verschiedenartige Anwendungszwecke. Neben der reinen Datenkorrektur auf Transaktionsebene entsteht so auch eine weitere Nutzungsmöglichkeit durch ein derartiges „absichtliches Forken“. Unter Zuhilfenahme von expliziten Transaktionsabhängigkeiten innerhalb der Transaktionen können damit, losgelöst von der Semantik, zusammengehörige Transaktionen identifiziert und anschließend so modifiziert werden, dass die Blockchain-Historie minimiert, aber dennoch der gesamte Systemzustand beibehalten werden kann. Für den Geschäftsprozess nicht mehr benötigte oder irrelevante Transaktionen können so aus der Historie der Blockchain gestrichen werden, was ihren Speicherbedarf verringert und gleichzeitig das Onboarding neuer Knoten beschleunigt.

Für eine exemplarische Realisierung eines generalisierten Blockchain-Frameworks wurden zuerst eine Reihe von Software-Entwicklungsparadigmen diskutiert und es wurde anschließend ein für die anvisierten Anwendungsszenarien besonders geeignetes ausgewählt. Das ausgewählte Paradigma der „Aktiven Komponenten“ vereint serviceorientierte mit agentenorientierten Architekturen, wodurch es gerade für stark nebenläufige Anwendungen, die sowohl feste Strukturen als auch klar definierte Schnittstellen benötigen, besonders geeignet ist. Für die Ausführung einer Software, die nach dem entsprechenden Paradigma der Aktiven Komponenten entworfen ist, musste zudem eine entsprechende Ausführungsumgebung identifiziert werden. Hier eignete sich insbesondere die Jadex-Middleware für die Realisierung, da sie alle für die Umsetzung erforderlichen Funktionen unterstützt.

Das entwickelte generalisierte Blockchain-Framework unterteilt sich, gemäß dem im Rahmen dieser Arbeit entwickelten Konzept, in mehrere Kernkomponenten. Diese sind durch klare Schnittstellen voneinander getrennt und lassen sich so relativ einfach durch Komponenten mit anderen Funktionen austauschen. Durch die Verwendung einer serviceorientierten Architektur konnte zudem ermöglicht werden, dass sehr flexibel auch andere Konsensalgorithmen, Datenbanken oder Kommunikationswege eingesetzt werden können, sollte es der damit umzusetzende Geschäftsprozess erfordern. Die lose Kopplung der Komponenten durch den serviceorientierten Ansatz ermöglichte damit so auch (bis auf wenige Ausnahmen) eine von der Anwendungsschicht unabhängige Funktionsweise des Frameworks. Zudem wurden im Rahmen der Arbeit erweiterte Blockchain-Funktionen zu Datenkorrekturverfahren, dem Ersetzen von Transaktionen und dem Reduzieren der Historie auf Grundlage des absichtlichen Forkens konzipiert und ebenfalls exemplarisch innerhalb des Frameworks prototypisch realisiert.

Abschließend wurde das vorgestellte Konzept und der darauf aufbauend entwickelte Prototyp eines generalisierten Blockchain-Frameworks qualitativ anhand einiger ausgewählter Anwendungsszenarien untersucht und bewertet. Dabei wurde der Fokus hauptsächlich auf die softwaretechnischen Aspekte der Umsetzung und der daraus resultierenden Anwendungsgebie-

te gelegt. Dafür wurden mehrere Beispielszenarien aus dem Konzept- und aus dem Anforderungsteil dieser Arbeit erneut aufgegriffen, um diese mithilfe des exemplarisch entwickelten Prototyps praktisch umzusetzen. Anhand der so realisierten Beispielszenarien wurde damit gezeigt, dass die aufgestellten Herausforderungen der Geschäftsprozesse bewältigt wurden, und dass besonders die Anforderungen bezüglich der Sichtbarkeit, der Validierung und der Anonymität der Daten bzw. der Dateninhalte erfüllt werden konnten.

Auch konnte innerhalb einer derartigen Evaluation praktisch gezeigt werden, wie die weiterführenden Konzepte eingesetzt werden können und welche Szenarien sie zu realisieren ermöglichen. Sowohl die anwendungsorientierte Sortierung der Transaktion im MemPool bzw. in den resultierenden Blöcken als auch die komplexere Funktion des Squashings zur Minimierung der Blockchain-Historie wurden so exemplarisch umgesetzt und praktisch evaluiert. Zuletzt wurde in Anlehnung an die nutzergenerierte Programmlogik, wie sie beispielsweise Smart Contracts in Ethereum darstellen, ein Konzept für die Realisierung derartiger Smart Contracts innerhalb des hier entwickelten Prototyps entworfen und praktisch evaluiert.

Zusammenfassend leistet diese Arbeit damit sowohl eine umfassende technische als auch eine Anforderungsanalyse der Blockchain-Technologie unter besonderer Berücksichtigung der Herausforderungen für den Einsatz in verteilten Geschäftsprozessen. Das dafür im Rahmen dieser Arbeit entwickelte generalisierte Konzept einer verallgemeinerten Blockchain-Technologie, das sowohl technisch als auch semantisch von der Anwendungslogik getrennt ist, ebnet den Weg für einen qualitativ besseren Einsatz dieser Technologien zur Dezentralisierung von Geschäftsprozessen und Workflows verschiedenster Art.

9.2. Grenzen des Ansatzes

Das vorgestellte Konzept zur Dezentralisierung von Geschäftsprozessen mittels einer Blockchain-Technologie hat einige Einschränkungen bezüglich seiner Nutzbarkeit. Der zugrundeliegende Geschäftsprozess muss erst einmal überhaupt für eine Dezentralisierung in Frage kommen bzw. seine Blockchain-basierte Umsetzung sollte einen nennenswerten Vorteil bieten. Beispielsweise wenn in einem Prozess überhaupt nur zwei Akteure beteiligt sind, ist eine Blockchain-basierte Lösung nicht zwingend besser als eine klassische Umsetzung. Der Vorteil wäre hier ein nicht veränderbares verteiltes Log aller Vorgänge, um etwa in nachgelagerten Rechtsstreiten unabstreitbare Beweise vorzulegen. Allerdings wird in einem solchen Anwendungsszenario ein Knotenverbund von mindesten vier Validatoren benötigt, um die $3f + 1$ Anforderung (Kapitel 2.8.2) für byzantinische Fehlertoleranz zu erfüllen.

Ebenso wie eine minimale Teilnehmerzahl bzw. Knotenzahl besteht, existiert auch eine vertretbare maximale Anzahl für die Blockchain-basierte Umsetzung von Geschäftsprozessen. Der vorgeschlagene und prototypisch umgesetzte *Exonum* BFT-PoS Algorithmus erfordert einen hohen Nachrichtenaustausch, wie faktisch alle Algorithmen, die auf PBFT basieren. Bei steigender Anzahl an Validatoren, steigt die Anzahl an Nachrichten mit jedem neuen Knoten quadratisch. Die Menge an Nicht-Validatorknoten, also Knoten, die nicht am Entscheidungsprozess des Konsens beteiligt sind, haben keinen großen Einfluss auf die Nachrichtenmenge, da sie nicht im 3-Phasen Protokoll eingebunden werden müssen. Bei Geschäftsprozessen mit mehreren Hunderten Teilnehmern, muss also zwischen einer vertretbaren Nachrichtenmenge und damit auch der erhöhten Dauer für den Konsens oder einer „Zentralisierung“ der Validatoren auf eine Teilmenge der beteiligten Akteure gewählt werden.

Die vorgeschlagene maximale semantische Trennung der Anwendung vom Blockchain-Framework sorgt softwaretechnisch für eine lose Kopplung und bessere Wartbarkeit der Einzelkomponenten, erhöht aber gleichzeitig die Dau-

er und Komplexität der Funktionsaufrufe zwischen diesen beiden Teilen. Anfragen zwischen Blockchain und Anwendung laufen stets asynchron, weshalb schon bei der Entwicklung hierauf geachtet werden muss.

Ebenso sorgt die Trennung der beiden Teile für eine doppelte Datenhaltung. Während beispielsweise in Bitcoin die abgelegten Transaktionen auch semantisch ausgewertet werden können, muss im vorgestellten Konzept anders mit den Daten verfahren werden. Beispielsweise kann die Anwendung separat die Hashwerte der Transaktionen der betroffenen Daten speichern, um diese dann jedesmal aus der Blockchain-Historie abzufragen und zu deserialisieren. Dies erfordert nur eine kleine Datenbank auf der Anwendungsseite, aber eine erhöhte Dauer für die immer wiederkehrenden Abfragen der Transaktionen. Eine andere Möglichkeit ist, dass die Anwendung eine semantisch umgeformte Variante der Transaktionen in einer eigenen Datenbank hält. Dies ermöglicht es der Anwendung direkt mit den Daten zu arbeiten, es erfordert aber eine komplette Duplizierung der Daten.

Auf Anwendungsseite kann die Datenbank regelmäßig bereinigt werden, für das Bereinigen der Blockchain-Historie hingegen werden aber spezielle Verfahren benötigt, wie etwa das in dieser Arbeit vorgestellte Squashing-Konzept kombiniert mit dem ebenfalls vorgestellten absichtlichem Forken. Auch hier gilt, dass der Geschäftsprozess dieses Konzept sinnvoll unterstützen muss, um davon zu profitieren. Prozesse, bei denen schrittweise bestehende Daten durch neue Transaktionen verändert werden, profitieren von diesem Konzept, da ganze Graphen von Transaktionen zusammengeführt werden können. In Prozessen, in denen alle Transaktionen unabhängig voneinander sind und fortwährend Gültigkeit behalten, kann auch keine Minimierung der Historie erfolgen. Zusätzlich erfordert der Squashing Prozess und das Reintegrieren der Blöcke durch absichtliches Forken auch eine gewisse Zeit. Zeitkritische Geschäftsprozesse oder solche, in denen keine Zeiten ohne Transaktionsaufkommen gefunden werden können, eignen sich beispielsweise nicht für dieses Konzept.

9.3. Ausblick

Das im Rahmen dieser Arbeit entwickelte Konzept für ein generalisiertes Blockchain-Framework bietet eine Reihe von Anknüpfungspunkten für die Forschung, als auch technische Weiterentwicklungsmöglichkeiten.

Die dabei für die Umsetzung des Prototyps verwendete Middleware Jadex bietet, neben den hierbei verwendeten Software-Agenten und Aktiven Komponenten, auch die Möglichkeit zur direkten Modellierung, Integration und Ausführung von BPMN-Workflows [Hotz u. a. 2012]. Während in dieser Dissertation der Fokus auf dem Konzept einer generalisierten Blockchain-Technologie für jegliche verteilte Geschäftsprozesse lag, könnte in zukünftigen Arbeiten unter anderem auch die Verknüpfung von Blockchain und BPMN durch Jadex erfolgen. Dies würde es ermöglichen, dass viele Prozesse, die bereits mit BPMN technisch modelliert sind, mithilfe der hier entwickelten Technologie zusammen integriert werden können. Die Erweiterung von der BPMN um die Goal-oriented Processes (GPMN) bietet zudem noch weitere Möglichkeiten der Integration von Prozessen, da durch diese Modellierungssprache zusätzliche Ziele und auch Ziel-Relationen zu den bereits bestehenden Aktivitäten hinzukommen [Jander u. a. 2016, 2011].

Weitere Anknüpfungspunkte bietet die in dieser Arbeit eingeführte Squashing-Funktionalität für das Verkleinern beziehungsweise Bereinigen der Blockchain-Historie. Zur Erprobung des Konzeptes wurde in dieser Arbeit lediglich auf die generelle Funktionalität geachtet, nicht jedoch auf die weiteren Implikationen, die bei der Ausführung entstehen. In der Evaluierung wurde der Squash immer nach einem festen Intervall durchgeführt, was sich für einige Geschäftsprozesse mitunter nicht eignet. Hier müssen, möglicherweise adaptiv, andere Zeitpunkte für die Durchführung gefunden werden. Für eine automatische Detektion von geeigneten Zeitpunkten lassen sich gegebenenfalls auch Algorithmen zur Mustererkennung einsetzen, um beispielsweise Zeitfenster mit wenigen bis gar keinen stattfindenden Transaktionen zu identifizieren. Zusätzlich zum Erkennen von derartigen geeigneten Zeitfenstern

muss dabei auch die Dauer des Squash-Prozesses selbst mit einberechnet werden, was wiederum auf einer effektiven Prognose der geschätzten Dauer basiert [Orsini 2017].

Während die Integration von Smart Contract in das hier erstellte Blockchain-Konzept nur der Vorstellung der generellen Machbarkeit diente, setzen viele andere Blockchain-basierte Projekte und Prozesse hauptsächlich auf diese Form der Integration auf. Die technische Funktionalität der hier vorgestellten Lösung ist zwar durch die Nutzung von JavaScript deutlich einfacher zu realisieren, wenn es um die Unterstützung durch fertige Bibliotheken und Entwicklungsumgebungen geht. Allerdings ist deren Ausführungsgeschwindigkeit gegenüber Implementierungen wie Solidity deutlich geringer, was einen praktischen Einsatz derzeit verhindert. Für eine Verbesserung der Ausführungsgeschwindigkeit von nutzergenerierter Logik könnte hier etwa eine besser geeignete Ausführungsumgebung mit geringerem Funktionsumfang als beispielsweise die GraalVM verwendet werden, was wiederum die möglichen Anwendungsfälle reduzieren würde. Eine tiefere Integration innerhalb des Blockchain-Frameworks könnte ebenfalls für weitere Optimierungen sorgen, um beispielsweise das Laden und Speichern von Zuständen der Smart Contracts zu beschleunigen.

Ein weiteres, in dieser Arbeit nicht weiter betrachtetes Konzept ist die Verwendung von Multichain [Hwang u. a. 2018] oder Sidechain bzw. Channel-Konzepten [Singh u. a. 2020]. Diese verwenden mehrere Blockchain(-artige) Systeme mit miteinander verknüpften, aber verteilten Daten auf einer oder mehreren Blockchains (oder DLTs). Häufig werden dabei auf Seitenketten Daten und Zustände verwaltet, die nur einen begrenzten Teilnehmerkreis betreffen und mithilfe von sog. „Ankern“, meist in Form von Prüfsummen der betreffenden Daten, in der Hauptkette kryptografisch abgesichert sind. Auch die Verknüpfung von mehreren unabhängigen Blockchains untereinander bringt neue Herausforderungen mit sich, denn die Daten müssen gegenseitig jederzeit deterministisch sein und verifizierbar bleiben. Gleiches gilt für die Einbindung von externen Daten, wie beispielsweise von extern bereitgestellten Daten zum aktuellen Wetter. In der Praxis werden dafür aktuell

sog. „Orakel“ mit Smart Contracts eingesetzt. Diese sind Diensteanbieter für das deterministische Abrufen von externen Daten, welche über Prüfsummen und Protokolle beglaubigt werden. Jedoch sind diese immer externe Dienste, die wiederum ungewünschte Dritte bzw. Intermediäre darstellen. Hier bedarf es für eine vollständige Dezentralisierung also ebenfalls noch einer anderen weitergehenden Lösung.

A. Anhang

```
contract ERC20 {

mapping (address => uint256) private _balances;

function balanceOf(address account) public view virtual override
    returns (uint256) {
     return _balances[account];
}

function transfer(address recipient, uint256 amount) public virtual
     override returns (bool) {
     _balances[_msgSender()] = _balances[_msgSender()].sub(amount, "
        ERC20: transfer amount exceeds balance");
     _balances[recipient] = _balances[recipient].add(amount);
     emit Transfer(_msgSender(), recipient, amount);
     return true;
}
}
```

Algorithmus A.1: ERC20-Token in Solidity

```
// contract ERC20

var _balances;

function balanceOf(tokenOwner) {
     return _balances[tokenOwner] === undefined) ? 0.0 : _balances[
        tokenOwner];
}

function transfer(recipient, amount) {
```

```
10      var msgSender = __callInfos.getCaller();
11      _balances[msgSender] = safeSubtract(balanceOf(msgSender),
            amount, "ERC20: transfer amount exceeds balance");
12      _balances[recipient] = safeAdd(balanceOf(recipient), amount);
13      Event.emit("Transfer", JSON.stringify([msgSender, recipient,
            amount]));
14      return true;
15  }
```

Algorithmus A.2: ERC20-Token in JavaScript

```
1   // solidity
2   function sub(uint256 a, uint256 b) internal pure returns (uint256)
        {
3       require(b <= a, "SafeMath: subtraction overflow");
4       return a - b;
5   }
6
7   // javascript
8   function safeSub(a, b) {
9       __require(__isNumber(a));
10      __require(__isNumber(b));
11      __require(b <= a);
12
13      return a-b;
14  }
```

Algorithmus A.3: Sichere Math-Funktionen

Veröffentlichungen

Folgende Veröffentlichungen sind aus den Ergebnissen im Umfeld dieser Dissertation hervorgegangen:

Posdorfer, W., Kalinowski, J., Bornholdt, H., and Lamersdorf, W. (2018). Decentralized billing and subcontracting of application services for cloud environment providers. In *Proceedings of the ESOCC 2018 Workshops*, Lecture Notes in Computer Science, pages 78–89. Springer

Posdorfer, W. and Kalinowski, J. (2019). Contesting the truth - intentional forking in bft-pos blockchains. In *Highlights of Practical Applications of Survivable Agents and Multi-Agent Systems*, pages 112–120. Springer

Stübs, M., Posdorfer, W., and Kalinowski, J. (2020a). Business-driven blockchain-mempool model for cooperative optimization in smart grids. In *Smart Trends in Computing and Communications: Proceedings of SmartCom 2019*, pages 31–39. Springer

Posdorfer, W., Bornholdt, H., and Lamersdorf, W. (2020). Transaction dependency model for block minimization in arbitrary blockchains. In *2nd International Electronics Communication Conference (IECC 2020)*. ACM

Stübs, M., Posdorfer, W., and Momeni, S. (2020b). Blockchain-based multitier double-auctions for smart energy distribution grids. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE

Posdorfer, W., Kalinowski, J., and Bornholdt, H. (2021). Toward eu-gdpr compliant blockchains with intentional forking. In *Advances in Computer, Communication and Computational Sciences*, pages 649–658. Springer

Gao, G., Song, C., Bandara, A., Shen, M., Zhang, H., Huang, Y., Yang, F., Posdorfer, W., Tao, D., and Wen, Y. (2021). Fogchain: a blockchain-based peer-to-peer solarpower trading system powered by fog ai. In *IEEE internet of things journal*. IEEE. Accepted for publication

Literaturverzeichnis

[Abadi und Brunnermeier 2018] ABADI, Joseph ; BRUNNERMEIER, Markus: Blockchain economics / National Bureau of Economic Research. 2018. – Forschungsbericht

[Abeyratne und Monfared 2016] ABEYRATNE, Saveen A. ; MONFARED, Radmehr P.: Blockchain ready manufacturing supply chain using distributed ledger. In: *International Journal of Research in Engineering and Technology* 5 (2016), Nr. 9, S. 1–10

[Allweyer 2005] ALLWEYER, Thomas: *Geschäftsprozessmanagement: Strategie, Entwurf, Implementierung, Controlling*. W3l GmbH, 2005

[Alsberg und Day 1976] ALSBERG, Peter A. ; DAY, John D.: A principle for resilient sharing of distributed resources. In: *Proceedings of the 2nd international conference on Software engineering* IEEE Computer Society Press (Veranst.), 1976, S. 562–570

[Anceaume u. a. 2020] ANCEAUME, Emmanuelle ; POZZO, Antonella ; RIEUTORD, Thibault ; TUCCI-PIERGIOVANNI, Sara: On finality in blockchains. In: *arXiv preprint arXiv:2012.10172* (2020)

[Androulaki u. a. 2018] ANDROULAKI, Elli ; BARGER, Artem ; BORTNIKOV, Vita ; CACHIN, Christian ; CHRISTIDIS, Konstantinos ; DE CARO, Angelo ; ENYEART, David ; FERRIS, Christopher ; LAVENTMAN, Gennady ; MANEVICH, Yacov u. a.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: *Proceedings of the Thirteenth EuroSys Conference* ACM (Veranst.), 2018, S. 30

[Antonopoulos 2014] ANTONOPOULOS, Andreas M.: *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, Inc., 2014

[Atzei u. a. 2017] ATZEI, Nicola ; BARTOLETTI, Massimo ; CIMOLI, Tiziana: A survey of attacks on ethereum smart contracts (sok). In: *International conference on principles of security and trust* Springer (Veranst.), 2017, S. 164–186

[Back 2002] BACK, Adam: *Hashcash-amortizable publicly auditable cost-functions*. 2002

[Baird 2016] BAIRD, Leemon: Hashgraph consensus: fair, fast, byzantine fault tolerance. In: *White Paper* (2016). – URL http://www.swirlds.com/wp-content/uploads/2016/06/2016-05-31-Swirlds-Consensus-Algorithm-TR-2016-01.pdf. – Zugriff 13.01.2020

[Baudet u. a. 2019] BAUDET, Mathieu ; CHING, Avery ; CHURSIN, Andrey ; DANEZIS, George ; GARILLOT, François ; LI, Zekun ; MALKHI, Dahlia ; NAOR, Oded ; PERELMAN, Dmitri ; SONNINO, Alberto: *State machine replication in the Libra Blockchain*. 2019. – URL https://www.libraplus.org/docs/assets/papers/libra-consensus-state-machine-replication-in-the-libra-blockchain.pdf. – Zugriff amf 24.03.2020

[Benedetti u. a. 2014] BENEDETTI, Miriam ; BELLMAN, Atillio ; ROTUNNO, Rafaelle ; INTRONA, Vito ; CESAROTTI, Vittorio: Impact of track and trace integration on pharmaceutical production systems. In: *International Journal of Engineering Business Management* 6 (2014), Nr. Godište 2014, S. 6–25

[Bitcoin Community 2018] BITCOIN COMMUNITY: *Bitcoin Developer Reference*. 2018. – URL https://bitcoin.org/en/developer-reference. – Zugriff 21.01.2020

[Blockchain Luxembourg S.A. 2020] BLOCKCHAIN LUXEMBOURG S.A.: *Blockchain Size*. 2020. – URL https://www.blockchain.com/charts/blocks-size. – Zugriff am 13.01.2020

[Bonanni 2011] BONANNI, Leo: Sourcemap: eco-design, sustainable supply chains, and radical transparency. In: *XRDS: Crossroads, The ACM Magazine for Students* 17 (2011), Nr. 4, S. 22–26

[Bondi 2000] BONDI, André B: Characteristics of scalability and their impact on performance. In: *Proceedings of the 2nd international workshop on Software and performance*, 2000, S. 195–203

[Brassard u. a. 1988] BRASSARD, Gilles ; CHAUM, David ; CRÉPEAU, Claude: Minimum disclosure proofs of knowledge. In: *Journal of Computer and System Sciences* 37 (1988), Nr. 2, S. 156–189

[Braubach und Pokahr 2011] BRAUBACH, Lars ; POKAHR, Alexander: Addressing challenges of distributed systems using active components. In: *Intelligent Distributed Computing V*. Springer, 2011, S. 141–151

[Brewer 2000] BREWER, Eric A.: Towards robust distributed systems. In: *PODC* Bd. 7 Portland, OR (Veranst.), 2000

[Bruce 2014] BRUCE, JD: The mini-blockchain scheme. In: *White Paper* (2014). – URL http://bitpaper.info/paper/5659313586569216. – Zugriff 13.01.2020

[Buchman 2016] BUCHMAN, Ethan: *Tendermint: Byzantine fault tolerance in the age of blockchains*, Dissertation, 2016. – URL https://atrium.lib.uoguelph.ca/xmlui/bitstream/handle/10214/9769/Buchman_Ethan_201606_MAsc.pdf. – Zugriff 21.04.2020

[Budhiraja u. a. 1993] BUDHIRAJA, Navin ; MARZULLO, Keith ; SCHNEIDER, Fred B. ; TOUEG, Sam: The primary-backup approach. In: *Distributed systems* 2 (1993), S. 199–216

[Bundeswahlleiter 2015] BUNDESWAHLLEITER: *Online Wahlen*. 2015. – URL https://www.bundeswahlleiter.de/service/glossar/o/online-wahlen.html. – Zugriff 13.01.2020

[Buterin 2014] BUTERIN, Vitalik: A next-generation smart contract and decentralized application platform. In: *White Paper* (2014). – URL https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf. – Zugriff 13.01.2020

[Canetti und Rabin 1993] CANETTI, Ran ; RABIN, Tal: Fast asynchronous Byzantine agreement with optimal resilience. In: *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, S. 42–51

[Castro und Liskov 1999] CASTRO, Miguel ; LISKOV, Barbara: Practical Byzantine fault tolerance. In: *OSDI* Bd. 99, 1999, S. 173–186

[Chandra u. a. 2007] CHANDRA, Tushar D. ; GRIESEMER, Robert ; REDSTONE, Joshua: Paxos made live: an engineering perspective. In: *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, 2007, S. 398–407

[Chandra und Toueg 1996] CHANDRA, Tushar D. ; TOUEG, Sam: Unreliable failure detectors for reliable distributed systems. In: *Journal of the ACM (JACM)* 43 (1996), Nr. 2, S. 225–267

[Chaum 1985] CHAUM, David: Security without identification: Transaction systems to make big brother obsolete. In: *Communications of the ACM* 28 (1985), Nr. 10, S. 1030–1044

[Chaum u. a. 1988] CHAUM, David ; FIAT, Amos ; NAOR, Moni: Untraceable electronic cash. In: *Conference on the Theory and Application of Cryptography* Springer (Veranst.), 1988, S. 319–327

[Chen u. a. 2017] CHEN, Lin ; XU, Lei ; SHAH, Nolan ; GAO, Zhimin ; LU, Yang ; SHI, Weidong: On security analysis of proof-of-elapsed-time (poet). In: *International Symposium on Stabilization, Safety, and Security of Distributed Systems* Springer (Veranst.), 2017, S. 282–297

[Chervinski u. a. 2019] CHERVINSKI, Joao Otávio M. ; KREUTZ, Diego ; YU, Jiangshan: FloodXMR: Low-cost transaction flooding attack with Monero's bulletproof protocol. In: *IACR Cryptology ePrint Archive* 2019 (2019), S. 455

[Christidis und Devetsikiotis 2016] CHRISTIDIS, Konstantinos ; DEVETSIKIOTIS, Michael: Blockchains and smart contracts for the internet of things. In: *IEEE Access* 4 (2016), S. 2292–2303

[Christopher 2005] CHRISTOPHER, Martin: *Logistics and supply chain management: creating value-adding networks*. Pearson education, 2005

[Costa u. a. 2013] COSTA, Corrado ; ANTONUCCI, Francesca ; PALLOTTINO, Federico ; AGUZZI, Jacopo ; SARRIÁ, David ; MENESATTI, Paolo: A review on agri-food supply chain traceability by means of RFID technology. In: *Food and bioprocess technology* 6 (2013), Nr. 2, S. 353–366

[De Angelis u. a. 2018] DE ANGELIS, Stefano ; ANIELLO, Leonardo ; BALDONI, Roberto ; LOMBARDI, Federico ; MARGHERI, Andrea ; SASSONE, Vladimiro: PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain. In: *Cryptology ePrint Archive* (2018). – URL https://eprint.iacr.org/2020/041.pdf. – Zugriff am 20.04.2020

[Delfs und Knebl 2015] DELFS, Hans ; KNEBL, Helmut: *Introduction to Cryptography: Principles and Applications, Third Edition*. Springer, 2015

[Delmolino u. a. 2016] DELMOLINO, Kevin ; ARNETT, Mitchell ; KOSBA, Ahmed ; MILLER, Andrew ; SHI, Elaine: Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In: *International conference on financial cryptography and data security* Springer (Veranst.), 2016, S. 79–94

[Dickerson u. a. 2019] DICKERSON, Thomas ; GAZZILLO, Paul ; HERLIHY, Maurice ; KOSKINEN, Eric: Adding concurrency to smart contracts. In: *Distributed Computing* (2019), S. 1–17

[Digiconomist 2020] DIGICONOMIST: *Bitcoin Energy Consumption Index*. 2020. – URL https://digiconomist.net/bitcoin-energy-consumption. – Zugriff 01.04.2020

[Dolev u. a. 1987] DOLEV, Danny ; DWORK, Cynthia ; STOCKMEYER, Larry: On the minimal synchronism needed for distributed consensus. In: *Journal of the ACM (JACM)* 34 (1987), Nr. 1, S. 77–97

[Douceur 2002] DOUCEUR, John R.: The sybil attack. In: *International workshop on peer-to-peer systems* Springer (Veranst.), 2002, S. 251–260

[Dwork u. a. 1988] DWORK, Cynthia ; LYNCH, Nancy ; STOCKMEYER, Larry: Consensus in the presence of partial synchrony. In: *Journal of the ACM (JACM)* 35 (1988), Nr. 2, S. 288–323

[Dwork und Naor 1992] DWORK, Cynthia ; NAOR, Moni: Pricing via processing or combatting junk mail. In: *Annual International Cryptology Conference* Springer (Veranst.), 1992, S. 139–147

[EESPA 2020] EESPA, European E-invoicing Service Providers Association: *Workflow - EESPA Glossary*. 2020. – URL https://eespa.eu/glossary/workflow/. – Zugriff 12.04.2021

[Erkens u. a. 2012] ERKENS, David H. ; HUNG, Mingyi ; MATOS, Pedro: Corporate governance in the 2007–2008 financial crisis: Evidence from financial institutions worldwide. In: *Journal of corporate finance* 18 (2012), Nr. 2, S. 389–411

[Europäisches Parlament und Rat 2011] EUROPÄISCHES PARLAMENT UND RAT: *EU Verodnung Nr. 1227/2011 - Integrität und Transparenz des Energiegroßhandelsmarkts*. 2011. – URL http://data.europa.eu/eli/reg/2011/1227/oj. – Zugriff 28.07.2020

[Europäisches Parlament und Rat 2014] EUROPÄISCHES PARLAMENT UND RAT: *Richtlinie 2014/65/EU - Märkte für Finanzinstrumente*. 2014. – URL http://data.europa.eu/eli/dir/2014/65/oj. – Zugriff 28.07.2020

[Fischer 1983] FISCHER, Michael J.: The consensus problem in unreliable distributed systems (a brief survey). In: *International conference on fundamentals of computation theory* Springer (Veranst.), 1983, S. 127–140

[Fischer u. a. 1985] FISCHER, Michael J. ; LYNCH, Nancy A. ; PATERSON, Michael S.: Impossibility of distributed consensus with one faulty process. In: *Journal of the ACM (JACM)* 32 (1985), Nr. 2, S. 374–382

[Garay und Moses 1998] GARAY, Juan A. ; MOSES, Yoram: Fully polynomial Byzantine agreement for n> 3 t processors in t+ 1 rounds. In: *SIAM Journal on Computing* 27 (1998), Nr. 1, S. 247–290

[Gentry 2009] GENTRY, Craig: Fully homomorphic encryption using ideal lattices. In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, S. 169–178

[Georgeff u. a. 1998] GEORGEFF, Michael ; PELL, Barney ; POLLACK, Martha ; TAMBE, Milind ; WOOLDRIDGE, Michael: The belief-desire-intention model of agency. In: *International Workshop on Agent Theories, Architectures, and Languages* Springer (Veranst.), 1998, S. 1–10

[Gilbreth und Gilbreth 1922] GILBRETH, Frank B. ; GILBRETH, LM: Process charts and their place in management. In: *Mechanical engineering* 70 (1922), Nr. 1, S. 38–41

[Gomez 2017] GOMEZ, Miguel: *Ethereum Co-Founder Vitalik Buterin Weighs in on Blockchain Improvement & Scaling Issues*. 2017. – URL https://cryptovest.com/news/ethereum-co-founder-vitalik-buterin-weighs-in-on-blockchain-improvement--scaling-issues/

[González u. a. 1997] GONZÁLEZ, Oscar ; SHRIKUMAR, H ; STANKOVIC, John A. ; RAMAMRITHAM, Krithi: Adaptive fault tolerance and graceful degradation under dynamic hard real-time scheduling. In: *Proceedings Real-Time Systems Symposium* IEEE (Veranst.), 1997, S. 79–89

[Gray u. a. 1981] GRAY, Jim u. a.: The transaction concept: Virtues and limitations. In: *VLDB* Bd. 81, 1981, S. 144–154

[Gray und Lamport 2006] GRAY, Jim ; LAMPORT, Leslie: Consensus on transaction commit. In: *ACM Transactions on Database Systems (TODS)* 31 (2006), Nr. 1, S. 133–160

[Haber und Stornetta 1990] HABER, Stuart ; STORNETTA, W S.: How to time-stamp a digital document. In: *Conference on the Theory and Application of Cryptography* Springer (Veranst.), 1990, S. 437–455

[Haq u. a. 2010] HAQ, Izhar ; MONFARED, Radmehr ; HARRISON, Robert ; LEE, Les ; WEST, A: A new vision for the automation systems engineering for automotive powertrain assembly. In: *International Journal of Computer Integrated Manufacturing* 23 (2010), Nr. 4, S. 308–324

[Hewitt u. a. 1973] HEWITT, Carl ; BISHOP, Peter ; STEIGER, Richard: Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence. In: *Advance Papers of the Conference* Bd. 3 Stanford Research Institute (Veranst.), 1973, S. 235

[Higgins 2017] HIGGINS, Stan: *From $900 to $20,000: Bitcoin's Historic 2017 Price Run Revisited*. 2017. – URL https://www.coindesk.com/900-20000-bitcoins-historic-2017-price-run-revisited. – Zugriff 30.03.2020

[Hileman und Rauchs 2017] HILEMAN, Garrick ; RAUCHS, Michel: Global cryptocurrency benchmarking study. In: *Cambridge Centre for Alternative Finance* 33 (2017)

[Hjálmarsson u. a. 2018] HJÁLMARSSON, Friðrik Þ ; HREIÐARSSON, Gunnlaugur K. ; HAMDAQA, Mohammad ; HJÁLMTÝSSON, Gísli: Blockchain-based e-voting system. In: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* IEEE (Veranst.), 2018, S. 983–986

[Hotz u. a. 2012] HOTZ, Lothar ; VON RIEGEN, Stephanie ; POKAHR, Alexander ; BRAUBACH, Lars ; SCHWINGHAMMER, Torsten: Monitoring BPMN-Processes with Rules in a Distributed Environment. In: *RuleML (2)*, 2012

[Huhns und Singh 2005] HUHNS, Michael N. ; SINGH, Munindar P.: Service-oriented computing: Key concepts and principles. In: *IEEE Internet computing* 9 (2005), Nr. 1, S. 75–81

[Hwang u. a. 2018] HWANG, Gwan-Hwan ; CHEN, Po-Han ; LU, Chun-Hao ; CHIU, Chun ; LIN, Hsuan-Cheng ; JHENG, An-Jie: InfiniteChain: A multi-chain architecture with distributed auditing of sidechains for public blockchains. In: *International Conference on Blockchain* Springer (Veranst.), 2018, S. 47–60

[Jander u. a. 2016] JANDER, Kai ; BRAUBACH, Lars ; LAMERSDORF, Winfried: Distributed monitoring and workflow management for goal-oriented workflows. In: *Concurrency and Computation: Practice and Experience* 28 (2016), Nr. 4, S. 1324–1335

[Jander u. a. 2011] JANDER, Kai ; BRAUBACH, Lars ; POKAHR, Alexander ; LAMERSDORF, Winfried ; WACK, Karl-Josef: Goal-oriented processes with GPMN. In: *International Journal on Artificial Intelligence Tools* 20 (2011), Nr. 06, S. 1021–1041

[Jennings u. a. 1998] JENNINGS, Nicholas R. ; SYCARA, Katia ; WOOLDRIDGE, Michael: A roadmap of agent research and development. In: *Autonomous agents and multi-agent systems* 1 (1998), Nr. 1, S. 7–38

[Johnson 1984] JOHNSON, Barry: Fault-tolerant microprocessor-based systems. In: *IEEE Micro* (1984), Nr. 6, S. 6–21

[Johnson 2017] JOHNSON, Nick: *Why I find Iota deeply alarming*. 2017. – URL https://hackernoon.com/why-i-find-iota-deeply-alarming-934f1908194b. – Zugriff am 13.01.2020

[Kaiser u. a. 2018] KAISER, Ben ; JURADO, Mireya ; LEDGER, Alex: The looming threat of china: An analysis of chinese influence on bitcoin. In: *arXiv preprint arXiv:1810.02466* (2018)

[Karantias u. a. 2020] KARANTIAS, Kostis ; KIAYIAS, Aggelos ; ZINDROS, Dionysis: Proof-of-burn. In: *International Conference on Financial Cryptography and Data Security* Springer (Veranst.), 2020, S. 523–540

[Katz und Lindell 2014] KATZ, Jonathan ; LINDELL, Yehuda: *Introduction to modern cryptography*. CRC press, 2014

[Kauflin 2018] KAUFLIN, Jeff: *IOTA Rose 464% In 2017, But Buyer Beware: Experts Have Major Security Concerns*. 2018. – URL https://www.forbes.com/sites/jeffkauflin/2018/01/03/iota-rose-464-in-2017-but-buyer-beware-experts-have-major-security-concerns/#28f1c3b45faa. – Zugriff am 13.01.2020

[Keidar und Rajsbaum 2001] KEIDAR, Idit ; RAJSBAUM, Sergio: On the cost of fault-tolerant consensus when there are no faults: preliminary version. In: *ACM SIGACT News* 32 (2001), Nr. 2, S. 45–63

[Kiayias u. a. 2017] KIAYIAS, Aggelos ; RUSSELL, Alexander ; DAVID, Bernardo ; OLIYNYKOV, Roman: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: *Annual International Cryptology Conference* Springer (Veranst.), 2017, S. 357–388

[Kim und Laskowski 2018] KIM, Henry M. ; LASKOWSKI, Marek: Toward an ontology-driven blockchain design for supply-chain provenance. In: *Intelligent Systems in Accounting, Finance and Management* 25 (2018), Nr. 1, S. 18–27

[King 2013] KING, Sunny: Primecoin: Cryptocurrency with prime number proof-of-work. In: *July 7th* 1 (2013), Nr. 6

[King und Nadal 2012] KING, Sunny ; NADAL, Scott: Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. In: *White Paper* 19 (2012).

– URL https://bitcoin.peryaudo.org/vendor/peercoin-paper.pdf. – Zugriff 13.01.2020

[Kirsch und Amir 2008] KIRSCH, Jonathan ; AMIR, Yair: Paxos for system builders: An overview. In: *Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*, 2008, S. 1–6

[Klebsch u. a. 2019] KLEBSCH, Wolfgang ; HALLENSLEBEN, Sebastian ; KOSSLERS, Sebastian: Roter Faden durch das Thema Blockchain. (2019)

[Klen u. a. 1999] KLEN, Alexandra Augusta P. ; RABELO, Ricardo J. ; SPINOSA, Luiz M. ; FERREIRA, Aureo C.: Distributed business process management. In: *Working Conference on Virtual Enterprises* Springer (Veranst.), 1999, S. 241–258

[Kollmann 2019] KOLLMANN, Tobias: Die Grundlagen des E-Marketplace. In: *E-Business*. Springer, 2019, S. 495–670

[Koshy u. a. 2014] KOSHY, Philip ; KOSHY, Diana ; MCDANIEL, Patrick: An analysis of anonymity in bitcoin using p2p network traffic. In: *International Conference on Financial Cryptography and Data Security* Springer (Veranst.), 2014, S. 469–485

[Krug und Peterson 2015] KRUG, Joseph ; PETERSON, Jack: Sidecoin: a snapshot mechanism for bootstrapping a blockchain. In: *arXiv preprint arXiv:1501.01039* (2015)

[Kshetri 2017] KSHETRI, Nir: *Potential roles of blockchain in fighting poverty and reducing financial exclusion in the global south*. 2017

[Kusmierz 2017] KUSMIERZ, B: The first glance at the simulation of the Tangle: discrete model. In: *White Paper* (2017), S. 1–10. – URL https://assets.ctfassets.net/r1dr6vzfxhev/2ZO5XxwehymSMsgusUE6YG/f15f4571500a64b7741963df5312c7e7/The_First_Glance_of_the_Simulation_Tangle_-_Discrete_Model_v0.1.pdf. – Zugriff 13.01.2020

[Kwon 2014] KWON, Jae: Tendermint: Consensus without mining. In: *White Paper* (2014). – URL https://cdn.relayto.com/media/files/LPgoWO18TCeMIggJVakt_tendermint.pdf. – Zugriff 13.01.2020

[Lamport 1979] LAMPORT, Leslie: Time, Clocks, and the Ordering of Events in a Distributed System. In: *Communications of the ACM* 21 (1979), Nr. 7, S. 558

[Lamport 1984] LAMPORT, Leslie: Using time instead of timeout for fault-tolerant distributed systems. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 6 (1984), Nr. 2, S. 254–280

[Lamport 1998] LAMPORT, Leslie: The part-time parliament. In: *ACM Transactions on Computer Systems (TOCS)* 16 (1998), Nr. 2, S. 133–169

[Lamport u. a. 2001] LAMPORT, Leslie u. a.: Paxos made simple. In: *ACM Sigact News* 32 (2001), Nr. 4, S. 18–25

[Lamport u. a. 1982] LAMPORT, Leslie ; SHOSTAK, Robert ; PEASE, Marshall: The Byzantine generals problem. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4 (1982), Nr. 3, S. 382–401

[Lampson und Sturgis 1979] LAMPSON, Butler ; STURGIS, Howard E.: Crash Recovery in a Distributed Data Storage System. (1979), January. – URL https://www.microsoft.com/en-us/research/publication/crash-recovery-in-a-distributed-data-storage-system/

[Larimer 2017] LARIMER, Dan: *DPOS Consensus Algorithm - The Missing White Paper*. 2017. – URL https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper. – Zugriff 15.04.2020

[Larimer 2018] LARIMER, Dan: Delegated proof-of-stake consensus. In: *White Paper* (2018). – URL https://bitshares.org/technology/delegated-proof-of-stake-consensus/. – Zugriff 15.04.2020

[Levine u. a. 1989] LEVINE, J ; WEISS, M ; DAVIS, DD ; ALLAN, DW ; SULLIVAN, DB: The NIST automated computer time service. In: *Journal of research of the National Institute of Standards and Technology* 94 (1989), Nr. 5, S. 311

[Lin und Krogstie 2010] LIN, Yun ; KROGSTIE, John: Semantic annotation of process models for facilitating process knowledge management. In: *International Journal of Information System Modeling and Design (IJISMD)* 1 (2010), Nr. 3, S. 45–67

[Lindsey u. a. 1985] LINDSEY, William C. ; GHAZVINIAN, Farzad ; HAGMANN, Walter C. ; DESSOUKY, Khaled: Network synchronization. In: *Proceedings of the IEEE* 73 (1985), Nr. 10, S. 1445–1467

[Lombrozo u. a. 2015] LOMBROZO, Eric ; LAU, Johnson ; WUILLE, Pieter: *Segregated Witness (Consensus layer)*. 2015. – URL https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki. – Zugriff 23.04.2020

[Luu u. a. 2016] LUU, Loi ; CHU, Duc-Hiep ; OLICKEL, Hrishi ; SAXENA, Prateek ; HOBOR, Aquinas: Making smart contracts smarter. In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, S. 254–269

[Lynch 1996] LYNCH, Nancy A.: *Distributed Algorithms*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1996. – ISBN 9780080504704

[Manohar und Briggs 2018] MANOHAR, Arthi ; BRIGGS, Jo: Identity Management in the Age of Blockchain 3.0. (2018)

[Marino und Rowley 2009] MARINO, Jim ; ROWLEY, Michael: *Understanding sca (service component architecture)*. Pearson Education, 2009

[Masternak und Pobiega 2020] MASTERNAK, Tomek ; POBIEGA, Szymon: *Distributed business processes*. 2020. – URL https://exactly-once.github.io/posts/distributed-business-processes/. – Zugriff 10.05.2021

[Mazieres 2007] MAZIERES, David: *Paxos made practical*. 2007. – URL http://read.seas.harvard.edu/~kohler/class/08w-dsi/mazieres07paxos.pdf. – Zugriff 24.03.2020

[Mazieres 2015] MAZIERES, David: The stellar consensus protocol: A federated model for internet-level consensus. In: *Stellar Development Foundation* (2015)

[McKibbin und Stoeckel 2010] MCKIBBIN, Warwick J. ; STOECKEL, Andrew: The global financial crisis: Causes and consequences. In: *Asian Economic Papers* 9 (2010), Nr. 1, S. 54–86

[Mehar u. a. 2019] MEHAR, Muhammad I. ; SHIER, Charles L. ; GIAMBATTISTA, Alana ; GONG, Elgar ; FLETCHER, Gabrielle ; SANAYHIE, Ryan ; KIM, Henry M. ; LASKOWSKI, Marek: Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack. In: *Journal of Cases on Information Technology (JCIT)* 21 (2019), Nr. 1, S. 19–32

[Meier und Cahill 2005] MEIER, René ; CAHILL, Vinny: Taxonomy of distributed event-based programming systems. In: *The Computer Journal* 48 (2005), Nr. 5, S. 602–626

[Merkle 1987] MERKLE, Ralph C.: A digital signature based on a conventional encryption function. In: *Conference on the theory and application of cryptographic techniques* Springer (Veranst.), 1987, S. 369–378

[Merz 2019] MERZ, Michael: *Blockchain im B2B-Einsatz*. MM Publishing, Hamburg, 2019

[Morrison 1968] MORRISON, Donald R.: PATRICIA—practical algorithm to retrieve information coded in alphanumeric. In: *Journal of the ACM (JACM)* 15 (1968), Nr. 4, S. 514–534

[Nakamoto 2008] NAKAMOTO, Satoshi: *Bitcoin: A peer-to-peer electronic cash system*. 2008. – URL https://bitcoin.org/bitcoin.pdf. – Zugriff 13.01.2020

[NEM Foundation 2018] NEM FOUNDATION: NEM Technical Reference. (2018). – URL https://nem.io/wp-content/themes/nem/files/NEM_techRef.pdf. – Zugriff 15.04.2020

[O'Dwyer und Malone 2014] O'DWYER, Karl J. ; MALONE, David: Bitcoin mining and its energy footprint. (2014)

[Oh u. a. 2015] OH, JinSeok ; KWON, Jin-woo ; PARK, Hyukwoo ; MOON, Soo-Mook: Migration of web applications with seamless execution. In: *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2015, S. 173–185

[Ongaro 2014] ONGARO, Diego: *Consensus: Bridging theory and practice*, Stanford University, Dissertation, 2014

[Ongaro und Ousterhout 2013] ONGARO, Diego ; OUSTERHOUT, John: *In search of an understandable consensus algorithm (extended version)*. 2013. – URL https://raft.github.io/raft.pdf. – Zugriff 24.03.2020

[Ongaro und Ousterhout 2014] ONGARO, Diego ; OUSTERHOUT, John: In search of an understandable consensus algorithm. In: *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, 2014, S. 305–319

[Orsini 2017] ORSINI, Gabriel: *Kontextadaptive Anwendungsarchitekturen für das mobile Cloud Computing*. Vogt-Kölln-Str. 30, 22527 Hamburg, Germany, Universität Hamburg, Fachbereich Informatik, Dissertation, 10 2017

[Panurach 1996] PANURACH, Patiwat: Money in electronic commerce: Digital cash, electronic fund transfer, and ecash. In: *Communications of the ACM* 39 (1996), Nr. 6, S. 45–50

[Papazoglou 2003] PAPAZOGLOU, Mike P.: Service-oriented computing: Concepts, characteristics and directions. In: *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.* IEEE (Veranst.), 2003, S. 3–12

[Pease u. a. 1980] PEASE, Marshall ; SHOSTAK, Robert ; LAMPORT, Leslie: Reaching agreement in the presence of faults. In: *Journal of the ACM (JACM)* 27 (1980), Nr. 2, S. 228–234

[Pedersen 1991] PEDERSEN, Torben P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: *Crypto* Bd. 91 Springer (Veranst.), 1991, S. 129–140

[Pilkington 2016] PILKINGTON, Marc: Blockchain technology: principles and applications. In: *Research handbook on digital transformations*. Edward Elgar Publishing, 2016

[Pokahr und Braubach 2011] POKAHR, Alexander ; BRAUBACH, Lars: Active components: A software paradigm for distributed systems. In: *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 02* IEEE Computer Society (Veranst.), 2011, S. 141–144

[Pokahr und Braubach 2013] POKAHR, Alexander ; BRAUBACH, Lars: The active components approach for distributed systems development. In: *International Journal of Parallel, Emergent and Distributed Systems* 28 (2013), Nr. 4, S. 321–369

[Popov 2016] POPOV, Serguei: The tangle. In: *White Paper* (2016). – URL http://tanglereport.com/wp-content/uploads/2018/01/IOTA_Whitepaper.pdf. – Zugriff 13.01.2020

[Reisig 2012] REISIG, Wolfgang: *Petri nets: an introduction*. Bd. 4. Springer Science & Business Media, 2012

[Reiter 1996] REITER, Michael K.: A secure group membership protocol. In: *IEEE Transactions on Software Engineering* 22 (1996), Nr. 1, S. 31–42

[Reyna u. a. 2018] REYNA, Ana ; MARTÍN, Cristian ; CHEN, Jaime ; SOLER, Enrique ; DÍAZ, Manuel: On blockchain and its integration with IoT. Challenges and opportunities. In: *Future Generation Computer Systems* 88 (2018), S. 173–190

[Rivera u. a. 2017] RIVERA, Rogelio ; ROBLEDO, José G ; LARIOS, Víctor M ; AVALOS, Juan M.: How digital identity on blockchain can contribute in a smart city environment. In: *2017 International smart cities conference (ISC2)* IEEE (Veranst.), 2017, S. 1–4

[Rodarmor 2018] RODARMOR, Casey: *IOTA: The Brave Little Toaster That Couldn't*. 2018. – URL https://casey.github.io/iota/. – Zugriff am 13.01.2020

[Rose u. a. 2019] ROSE, David ; MACHERY, Edouard ; STICH, Stephen ; ALAI, Mario ; ANGELUCCI, Adriano ; BERNIŪNAS, Renatas ; BUCHTEL, Emma E. ; CHATTERJEE, Amita ; CHEON, Hyundeuk ; CHO, In-Rae u. a.: Nothing at stake in knowledge. In: *Noûs* 53 (2019), Nr. 1, S. 224–247

[Saad u. a. 2019] SAAD, Muhammad ; NJILLA, Laurent ; KAMHOUA, Charles ; KIM, Joongheon ; NYANG, DaeHun ; MOHAISEN, Aziz: Mempool Optimization for Defending Against DDoS Attacks in PoW-based Blockchain Systems. In: *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* IEEE (Veranst.), 2019, S. 285–292

[Saad u. a. 2020] SAAD, Muhammad ; SPAULDING, Jeffrey ; NJILLA, Laurent ; KAMHOUA, Charles ; SHETTY, Sachin ; NYANG, DaeHun ; MOHAISEN, David: Exploring the attack surface of blockchain: A comprehensive survey. In: *IEEE Communications Surveys & Tutorials* 22 (2020), Nr. 3, S. 1977–2008

[Sasson u. a. 2014] SASSON, Eli B. ; CHIESA, Alessandro ; GARMAN, Christina ; GREEN, Matthew ; MIERS, Ian ; TROMER, Eran ; VIRZA, Madars: Zerocash: Decentralized anonymous payments from bitcoin. In: *2014 IEEE Symposium on Security and Privacy* IEEE (Veranst.), 2014, S. 459–474

[Schmidt 1995] SCHMIDT, Douglas C.: Reactor: An object behavioral pattern for concurrent event demultiplexing and dispatching. (1995)

[Schneider 1990] SCHNEIDER, Fred B.: Implementing fault-tolerant services using the state machine approach: A tutorial. In: *ACM Computing Surveys (CSUR)* 22 (1990), Nr. 4, S. 299–319

[Schuh und Larimer 2017] SCHUH, Fabian ; LARIMER, Daniel: Bitshares 2.0: General overview. In: *White Paper* (2017). – URL https://cryptorating.eu/whitepapers/BitShares/bitshares-general.pdf. – Zugriff 13.01.2020

[Serban u. a. 2008] SERBAN, Constantin ; CHEN, Yingying ; ZHANG, Wenxuan ; MINSKY, Naftaly: The concept of decentralized and secure electronic marketplace. In: *Electronic Commerce Research* 8 (2008), Nr. 1-2, S. 79–101

[Shoker 2017] SHOKER, Ali: Sustainable blockchain through proof of exercise. In: *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)* IEEE (Veranst.), 2017, S. 1–9

[Singh u. a. 2020] SINGH, Amritraj ; CLICK, Kelly ; PARIZI, Reza M. ; ZHANG, Qi ; DEHGHANTANHA, Ali ; CHOO, Kim-Kwang R.: Sidechain technologies in blockchain networks: An examination and state-of-the-art review. In: *Journal of Network and Computer Applications* 149 (2020), S. 102471

[Staud 2006] STAUD, Josef L.: *Geschäftsprozessanalyse: ereignisgesteuerte Prozessketten und objektorientierte Geschäftsprozessmodellierung für betriebswirtschaftliche Standardsoftware*. Springer Science & Business Media, 2006

[Steemit Inc. 2017] STEEMIT INC.: Steem - An incentivized, blockchain-based, public content platform. (2017). – URL https://steem.com/SteemWhitePaper.pdf. – Zugriff 15.04.2020

[Stübs u. a. 2020] STÜBS, Marius ; POSDORFER, Wolf ; MOMENI, Sadaf: Blockchain-Based Multi-Tier Double Auctions for Smart Energy Distribution Grids. In: *2020 IEEE International Conference on Communications Workshops (ICC Workshops)* IEEE (Veranst.), 2020, S. 1–6

[Stübs u. a. 2019] STÜBS, Marius ; POSDORFER, Wolf ; KALINOWSKI, Julian: Business-driven Blockchain-Mempool Model for Cooperative Optimization in Smart Grids. In: *Smart Innovation, Systems and Technologies*, Springer International Publishing, 2019

[Sycara 1998] SYCARA, Katia P.: Multiagent systems. In: *AI magazine* 19 (1998), Nr. 2, S. 79–79

[Szabo 1997] SZABO, Nick: The Idea of Smart Contracts. (1997). – URL http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html. – Zugriff 13.01.2020

[Szyperski u. a. 2002] SZYPERSKI, Clemens ; GRUNTZ, Dominik ; MURER, Stephan: *Component software: beyond object-oriented programming*. Pearson Education, 2002

[TeleTrusT 2017] TELETRUST, Bundesverband IT-Sicherheit e.V.: *TeleTrusT-Positionspaper „Blockchain“*. 2017. – URL https://www.teletrust.de/fileadmin/docs/publikationen/broschueren/Blockchain/2017_TeleTrusT-Positionspapier_Blockchain__.pdf. – Zugriff am 13.01.2020

[Van Der Aalst u. a. 2004] VAN DER AALST, Wil ; VAN HEE, Kees M. ; HEE, Kees van: *Workflow management: models, methods, and systems*. MIT press, 2004

[Van Renesse und Altinbuken 2015] VAN RENESSE, Robbert ; ALTINBUKEN, Deniz: Paxos made moderately complex. In: *ACM Computing Surveys (CSUR)* 47 (2015), Nr. 3, S. 1–36

[Voulgaris u. a. 2005] VOULGARIS, Spyros ; GAVIDIA, Daniela ; VAN STEEN, Maarten: Cyclon: Inexpensive membership management for unstructured p2p overlays. In: *Journal of Network and systems Management* 13 (2005), Nr. 2, S. 197–217

[Vukolić 2015] VUKOLIĆ, Marko: The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In: *International Workshop on Open Problems in Network Security* Springer (Veranst.), 2015, S. 112–125

[Weske 2007] WESKE, Mathias: *Business process management architectures*. Springer, 2007

[White 2004] WHITE, Stephen A.: Introduction to BPMN. In: *Ibm Cooperation* 2 (2004), Nr. 0, S. 0

[Windley 2005] WINDLEY, Phillip J.: *Digital Identity: Unmasking identity management architecture (IMA)*. O'Reilly Media, Inc., 2005

[Wood 2018] WOOD, Gavin: Ethereum: A secure decentralised generalised transaction ledger. In: *White Paper* (2018). – URL https://gavwood.com/paper.pdf. – Zugriff 13.01.2020

[Wooldridge 2001] WOOLDRIDGE, Michael: Intelligent agents: The key concepts. In: *ECCAI Advanced Course on Artificial Intelligence* Springer (Veranst.), 2001, S. 3–43

[Wooldridgey und Ciancarini 2000] WOOLDRIDGEY, Michael ; CIANCARINI, Paolo: Agent-oriented software engineering: The state of the art. In: *International workshop on agent-oriented software engineering* Springer (Veranst.), 2000, S. 1–28

[Wyman 2016] WYMAN, Oliver: Blockchain in capital markets—the prize and the journey. (2016). – URL https://www.euroclear.com/dam/Brochures/BlockchainInCapitalMarkets-ThePrizeAndTheJourney.pdf. – Zugriff 28.07.2020

[Yu u. a. 2017] YU, Lian ; TSAI, Wei-Tek ; LI, Guannan ; YAO, Yafe ; HU, Chenjian ; DENG, Enyan: Smart-contract execution with concurrent block building. In: *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)* IEEE (Veranst.), 2017, S. 160–167

Hinweis

Bei einigen der verwendeten Quellen handelt es sich um sog. Whitepaper oder Yellowpaper, welche nicht zwingend dem wissenschaftlichen Standard entsprechen oder einem Peer-Review-Verfahren unterzogen und zumeist eigenständig veröffentlicht wurden. Die entsprechenden Quellen sind mit *White Paper* gekennzeichnet sowie einer, zum Zeitpunkt der Arbeit verfügbaren, abrufbaren Online-Quelle versehen.

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmen

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe.

Hamburg, den 08.09.2021

Wolf Dietmar Posdorfer