Frankfurt School Blockchain Center

JULI 2018

FSBC Working Paper

R3 Corda: Implementierung eines Prototyps für Schuld-scheindarlehen und Vergleich verschiedener DLT-Frameworks

Philipp Sandner, Christoph Impekoven, Jan-Philipp Wiemann

Mit diesem Working Paper stellen wir eine kurze Analyse von R3's Corda-Plattform zur Verfügung und teilen unsere Erfahrungen bei der Implementierung eines Prototyps. Zu Demonstrationszwecken wurde dieses Forschungsexperiment anhand des Finanzanwendungsfalles eines Schuldschein-darlehens durchgeführt – jeder andere Use Case ist aufgrund des flexiblen Frameworks von Corda denkbar. Ebenfalls enthalten ist eine kurze Übersicht und Abwägung hinsichtlich der Auswahl von Corda als eine Open-Source Plattform im Vergleich zu anderen existierenden Enterprise Distributed Ledger Frameworks.

Frankfurt School Blockchain Center
www.fs-blockchain.de
contact@fs-blockchain.de

Follow us
www.twitter.com/fsblockchain
www.facebook.de/fsblockchain

Frankfurt School of Finance & Management GmbH
Adickesallee 32-34
60322 Frankfurt am Main
Germany

Konzeption und Vorteile von R3 Corda

„R3 is an enterprise software firm working with over 100 banks, financial institutions, regulators, trade associations, professional services firms and technology companies to develop Corda, our distributed ledger platform designed specifically for financial services."[1] R3 als Konsortium aus über 70 Financial Service Providern wird zu jeweils einem Drittel finanziell aus

Asien, Nordamerika und Europa unterstützt und hat heute etwa 125 Mitarbeiter angestellt

Corda von R3 ist heute neben Hyperledger Fabric eines der führenden Software Frameworks für die Etablierung einer *permissioned Blockchain* Infrastruktur. Im Gegensatz zur *unpermissioned Blockchain*, wie beispielsweise der öffentlichen Bitcoin oder die Ethereum Blockchain, werden die Transaktionen in einer „privaten“ Blockchain Infrastruktur nicht im gesamten Netzwerk geteilt. Ursprünglich ist Corda angetreten, um das Problem des Datenschutzes einer Blockchain-Lösung zu lösen. Die Idee dahinter ist, die Transaktionen nur mit dem eigenen Netzwerk zu teilen. Dabei muss nicht zwingend eine „Blockchain“ im engeren Sinne implementiert sein, sondern relevant ist der Distributed Ledger. „Privacy“ und Interoperabilität sind dabei die beiden fundamentalen Designprinzipien von Corda.

Definition von Corda

"Corda is a distributed ledger platform designed and built from the ground up to record, manage and synchronize agreements (legal contracts), designed for use by regulated financial institutions but applicable to any industry."[2]

Entscheidend ist dabei, dass unterschiedliche Algorithmen für Konsens im Netzwerk sorgen. Dies findet allerdings nicht im gesamten Netzwerk, sondern auf der Logikebene einer Handelstransaktion statt. Eines der Grundprinzipien von Corda – und damit diametral divergent zu dem „Code is Law“ Prinzip von Ethereum – ist, dass die Einbettung von Vertragstexten vorgesehen ist und durch „Attachements“ ermöglicht wird – „[there are] […] explicit links between human-language, legal prose documents and smart contract code.“[3]

Der Konsens zwischen den im Netzwerk bei einer Transaktion beteiligten Nodes wird dementsprechend über einen Mittelsmann, der *Notary Node*[4] hergestellt. Das System erlaubt sogar spezielle *Supervisor*- beziehungsweise *Regulator Observer Nodes*[5].

Bevor wir in den Vergleich der Plattformen einsteigen, ist eine wichtige Differenzierung vorzunehmen. Grundsätzlich bieten alle Blockchain-basierten Systeme wie Bitcoin und Ethereum nur einen globalen Konsens auf der Ledger-Ebene an. Dementsprechend gibt es theoretisch nur gesamtheitlichen Konsens durch das sogenannte *Mining* oder es kommt zum logischen Fork. Dies ist bei Distributed Ledger Technology (DLT)-Systemen anders. Hier wird in der Regel ein Konsens (beispielsweise durch Notary Nodes) auf Transaktionsebene vorgenommen. Somit gibt es in den DLT-Systemen – je nach Ausgestaltung – und insbesondere bei Corda nicht eine einheitliche Wahrheit aller Transaktionen, sondern nur eine diskrete Wahrheit zwischen zwei oder mehreren an der Transaktion beteiligten Parteien. Neben R3 Corda finden sich noch weitere Distributed Ledger Plattformen, die für den Einsatz in dedizierten Communities, wie beispielsweise Firmennetzwerken, gebaut wurden. Nachfolgend wird eine Auflistung der bekanntesten Blockchain- beziehungsweise DLT-Plattformen aufgeführt:

Tabelle 1

Überblick über bekannte Blockchain/DLT-Plattformen

Organisation	**Plattform/ Framework**
Ethereum Foundation	– Ethereum as a Platform (ethereum.org)
JPMorgan Chase & Co	– Quorum™ (Ethereum-based)
Linux Foundation	– Hyperledger Fabric (with IBM and Digital Assets)
Chain Inc. (Chain.com)	– Chain Core / Sequence (seq.com)
R3 (R3CEV LLC)	– Corda Enterprise / Open Source
Monax.io	– Monax (former “Eris”)[6]
Clearmatics[7]	– "Distributed Virtual Machine" (DVM)
Blockstream	– Framework “Sidechain Elements”[8]

Diese Plattformen sind zu großen Teilen der Open Source Community zuzurechnen, auch wenn es sich dabei um Enterprise-Software handelt. Dass dies schon lange kein Widerspruch mehr ist, dürfte bereits im Rahmen der Linux Foundation bekannt sein. Selbst der Ethereum-Klon von J.P. Morgan ist komplett Open-Source-basiert[9]. Dieser Paradigmenwechsel lässt sich im Bankensektor insbesondere dadurch erklären, dass sich die Vorteile von DLT an der Schnittstelle zu anderen Instituten zeigen, sodass nur ein gemeinsamer Ansatz erfolgsversprechend ist.

Das bekannteste Framework aus der obenstehenden Auflistung ist sicherlich **Ethereum** mit der zugehörigen Digital- und Kryptowährung *Ether*. Das *UTXO Modell*[10] ist Standard für diese Transaktionen. Hier sind im wesentlichen Inputs, Outputs und Signaturen zu finden. Dieses Modell wird im Übrigen auch von Corda genutzt. Ethereum hat die besondere Eigenschaft, alle Transaktionen mit allen Nodes im Netzwerk zu teilen und via *Proof of Work (PoW)* einen Konsens herzustellen. Dies ist Vor- und Nachteil zugleich, bedeutet allerdings auch, dass keinerlei Privatsphäre existiert und dass die jeweiligen Transaktionsstatus aller Nodes eingesehen werden können (müssen). Corda löst dieses Problem, indem gezielt nicht auf Proof of Work sondern auf Notary Nodes gesetzt wird.

Der Anbieter **Monax.io** liefert für die drei Marktsegmente Finance, Insurance und Logistics ein Software Development Kit (SDK) an, mit dem einzelne Anwendungsfälle abgedeckt werden sollen.

Chain Core und die zugehörige *Sequence-Plattform* haben den Vorteil, dass bestehende Anwendungsfälle bereits implementiert und verfügbar sind.

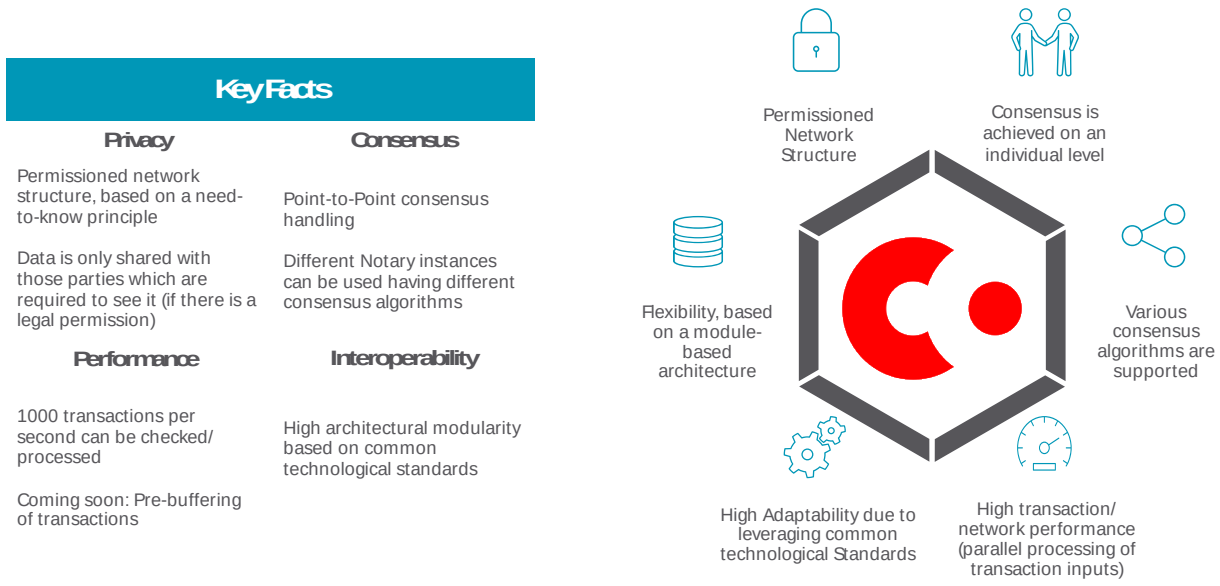
Der Provider **Clearmatics** entwickelt mit einigen Partnern eine eigene DLT-Plattform. Ein wesentlicher Anwendungsfall ist dabei der Utility Settlement Coin.

Vergleichen wir Corda und die weiteren Plattformen, fallen deutliche Unterschiede auf. Der entscheidende Vorteil für Corda liegt in der gut steuerbaren Node-Architektur und in dem Monitoring dieser.

Hyperledger Fabric ist recht flexibel in der Ausgestaltung und Implementierung der Smart-Contracts – hier werden insbesondere die drei Programmiersprachen Java, Java Script und Go unterstützt. Corda dagegen bietet Kompatibilität zu einer ganzen Bandbreite an Programmiersprachen an. Grundsätzlich lassen sich Kontrakte mit jeder Programmiersprache definieren, die kompatibel zur Java Virtual Machine (JVM) ist. “Smart contracts in Corda are defined using JVM bytecode [...]”[11]. Dies sind insbesondere Java und Kotlin, allerdings auch weitere wie beispielsweise Apache Groovy, JRuby, Jython, Clojure (Lisp Dialekt), Scala, Fantom und viele mehr[12]. “[...] [C]orda focuses on deterministic execution of any JVM bytecode, formally verifiable languages that target this instruction set are usable for the expression of smart contracts.”[13]

Abbildung 1

Zentrale Merkmale des Corda Frameworks



Framework Merkmale und Überblick

„Corda is a platform for the writing of CorDapps: applications that extend the global database with new capabilities. Such apps define new data types,

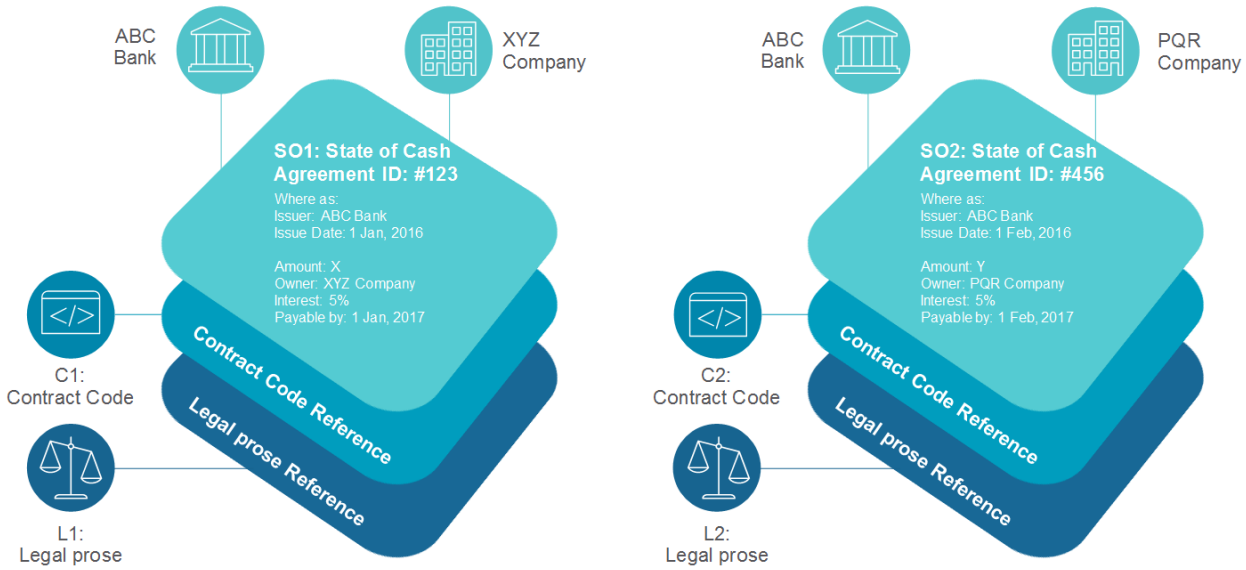
new inter-node protocol flows and the smart contracts that determine allowed changes."[14]

Das Corda Framework selbst bietet dabei den zugrunde liegenden, technologischen Rahmen, um eigene und neue *Flows* zwischen den Nodes zu implementieren. Dabei bietet Corda weitere Features (siehe Abbildung 1):

Ein weiterer Vorteil ist die Integration von Metadaten. Diese Möglichkeit ist bei einer klassischen Blockchain, z.B. der Bitcoin-Blockchain nicht gegeben, da diese nur einen Wert pro Schlüssel speichert: "[...] Corda database rows can contain arbitrary data, not just a value field. Because the data consumed and added by transactions is not necessarily a set of key/value pairs, we don't talk about rows but rather *states*."[15]

Passend gibt es dazu in Abbildung 2 eine schematische Darstellung von States in Corda:

Abbildung 2
States in Corda



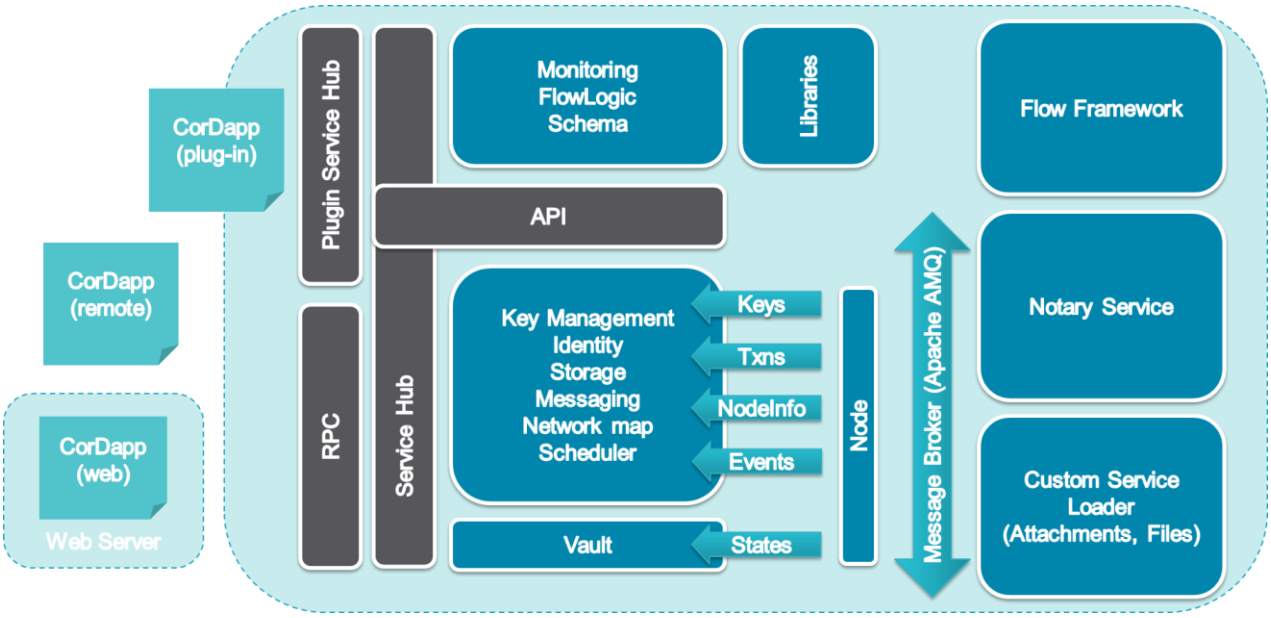
Besonders ist hier anzumerken, dass Contract Code und *Legal Prose* zusammen den *State of Cash* ergeben. Damit kann jeder Handelstransaktion immer ein rechtliches Dokument wie ein Kaufvertrag angefügt werden.

Technisches Design des Prototypens

Die micobo GmbH, als Unternehmensberatung spezialisiert auf Finanztechnologien, hat vor dem Hintergrund der Relevanz von Distributed Ledger Technology für Kapitalmärkte ein Minimal-Viable-Product (MVP) zur Umsetzung eines Schuldscheindarlehens basierend auf der R3's Corda Plattform entwickelt.

Abbildung 3

Überblick über das Corda Framework[16]



Vor dem Hintergrund dieser Entwicklung des avisierten Prototyps ist es wichtig, einige grundlegende technische Module des Corda-Frameworks näher zu betrachten.

Die Zusammensetzung der technischen Komponenten des Frameworks wird in der nachfolgend dargestellten technischen Architektur (siehe Abbildung 3) schematisch veranschaulicht:

Wichtig hierbei ist zu erwähnen, dass durch die Modularität der Architektur und durch die Verwendung von etablierten technischen Standards eine hohe Flexibilität und Integrität realisiert wird. Der Fokus liegt dabei auf der Interoperabilität der einzelnen Bausteine.

Wie in obiger Abbildung 3 dargestellt, stellt eine wesentliche Schnittstelle in der Corda-Architektur *Apache ActiveMQ*[17] dar, welches als Message Broker zwischen Nodes und Services agiert.

Um einen spezifischen Überblick zu geben, werden nachfolgend die wichtigsten Komponenten des Corda Frameworks beschrieben:

Notary. Notaries stellen einen bestimmten Netzwerk-Service dar, der sowohl dafür zuständig ist, das *Double-Spending*-Problem zu lösen, als auch (optional) Transaktionen zu validieren. Hierbei können verschiedene Consensus Algorithmen eingesetzt werden.

Nodes. Nodes sind JVM Run-Time-Entitäten, die eine eindeutige Identität im Netzwerk besitzen und über Schnittstellen mit anderen Nodes und dem *Node Owner* kommunizieren sowie Services aufrufen. Dabei werden alle relevanten (historischen und gegenwärtigen) States in einem *Vault*[18], inklusive Transaktionsdaten, Attachments und Flow-Checkpoints in einem dedizierten Storage-Service gesichert.

H2. Als zugrundeliegende Datenbank zur Sicherung von Node-Daten wird derzeit eine lokale *H2-Datenbank* (Java SQL-basiert) verwendet. Hierin werden Node-abhängige States, Transaktionen und Attachments gespeichert. Hier findet sich beispielsweise die *VAULT_CASH_BALANCES*-Tabelle, welche die aktuellen Zustände inklusive „Kontenstände“ des Vaults speichert.

SQL. Durch die Verwendung der zuvor genannten relationalen Datenbanktechnologie ist es möglich, Node-Daten über SQL abzufragen. Dies erleichtert zudem die Datenintegration anderer Datenbanksysteme.

JVM. Corda operiert auf der Java Virtual-Machine (JVM). Hierbei ist zu erwähnen, dass der Corda Kern selbst mittels Kotlin (als zugrundeliegende Programmiersprache) implementiert wurde. Lösungen und vor allem auch entsprechende *CorDapps*, die zusätzliche Funktionalitäten bereitstellen, können mittels JVM-kompatiblen Technologien implementiert werden.

MQ Messaging. Wie bereits eingangs genannt, wird die Kommunikation zwischen Nodes und Services (Notaries, etc.) durch den Apache ActiveMQ Message Broker realisiert.

HTTP. Da der Zugriff auf Nodes über HTTP realisiert wird, sind verschiedene HTTP-basierte Anwendungsfälle wie beispielweise die Ein- und Anbindung an Web-Applikationen denkbar.

REST-API. Schließlich ist besonders erwähnenswert, dass Corda eine REST/JSON API zur Anbindung und Kommunikation unter Nodes verwendet.

Der Anpassungsprozess

Der Anpassungsprozess zur Realisierung des Prototyps beginnt mit der Einrichtung der Entwicklungsumgebung. Hierzu werden folgende Komponenten benötigt:

- Als Java-IDE wird aufgrund der profunden Integrität und Unterstützung von Kotlin z.B. *IntelliJ* von *JetBrains*[19] empfohlen.
- Es wird zudem *Java SE Development-Kit* in Version 8 benötigt.
- Als *Git*-basiertes Projekt wird zudem die aktuelle *Git*-Version benötigt.
- Zudem wird nach der Installation der Komponenten die Corda-Version über das entsprechende GitHub Repository lokal repliziert.

Um Anpassungen am Quellcode vornehmen zu können, wurde das lokal geklonte Corda-Projekt mittels IntelliJ importiert. *Gradle*[20] wird dabei als integriertes Build-Management-Tool verwendet. Deshalb sind jegliche Build Operationen in dem entsprechenden Command-Line Interface vorzunehmen. Als Basis zur Entwicklung des Prototyps wurde die von Corda bereitgestellte *Demobench*[21] verwendet, da diese bereits grundlegende Transaktionen (wie Issue, Pay oder Exit) sowie eine grafische Benutzeroberfläche (Graphical User Interface (GUI)) bietet, um Operationen auf Nodes auszuführen und mit diesen zu interagieren. In dem Zusammenhang wurde zunächst eine Quellcode-Analyse hinsichtlich der Integration und des Umgangs mit Modulen zur Erzeugung von

- dem neuem Transaktionstyp „Schuldscheindarlehen“,
- zugrundeliegenden Flows,
- Smart-Contracts sowie
- der GUI

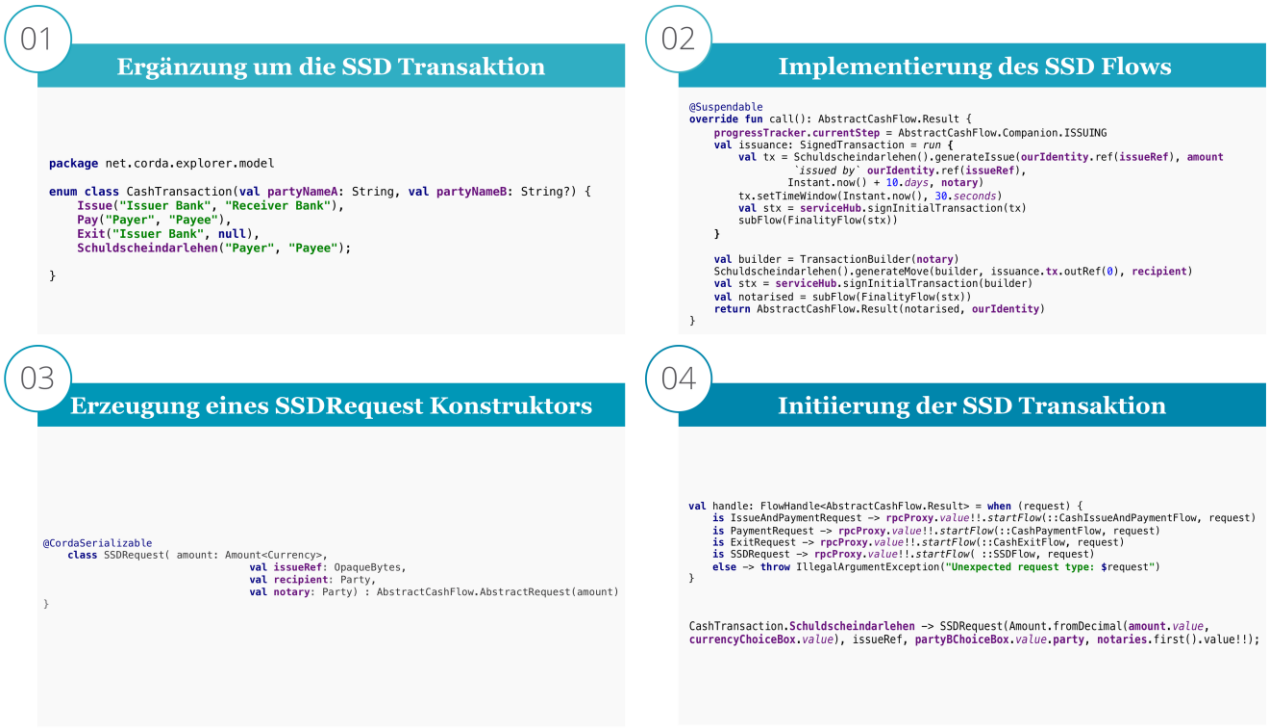
durchgeführt.

Die Anpassungen zur Erzeugung eines Schuldscheindarlehen-spezifischen Transaktionstyps sowie die Einbindung eines Flows mit spezifischen Attributen und mit Smart Contracts wurden entsprechend in verschiedenen **.kt*-Dateien vorgenommen. Die nachfolgende Abbildung 4 gibt einen Überblick über den Anpassungsprozess:

Wie in obiger Abbildung 4 dargestellt, wurde zunächst der Transaktionstyp „Schuldscheindarlehen“ entsprechend im Quellcode ergänzt (1). Da Transaktionen durch Flows automatisiert werden, wurde danach ein entsprechender SSD Flow mit spezifischer Flow Logik mittels Transaction Builder[22] erzeugt (2). Die Integration des Schuldschein-spezifischen Flows (kurz SSD-Flow) wurde durch die Erzeugung eines *Request Konstruktors* realisiert (3). Der Flow wurde nun durch den SSDRequest- und die startFlow()-Methode initiiert und die relevanten Attribute dem Konstruktor zugewiesen (4).

Abbildung 4

Der Anpassungsprozess im Quellcode



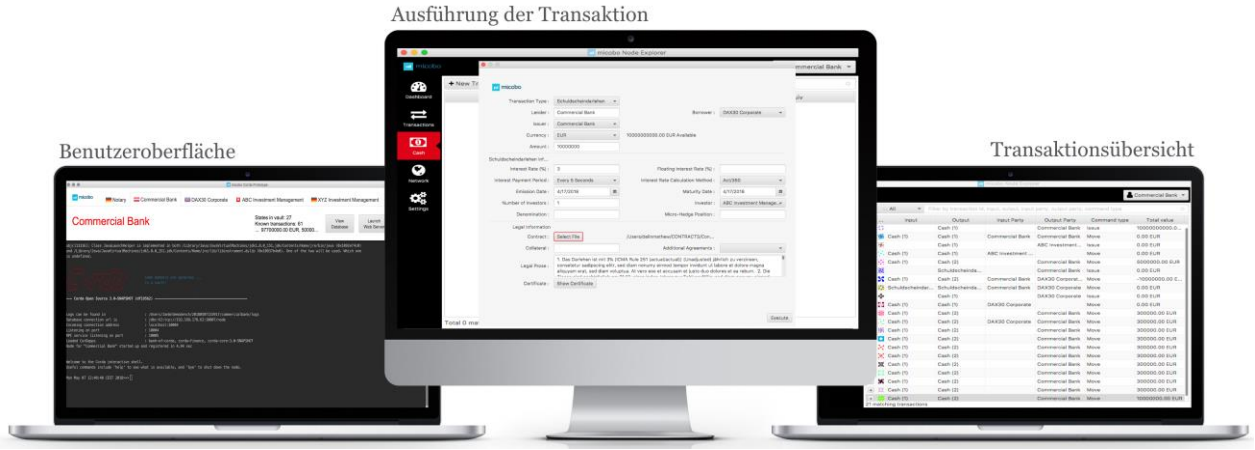
Nach Anpassung des Quellcodes zur Integration der Geschäftslogik wurde die GUI im *Node Explorer*[23] mittels *JavaFX*[24] in den zugehörigen **.fxml*-Dateien entsprechend der neuen Transaktion adaptiert. Der Node-Explorer stellt eine wesentliche Komponente der GUIs dar, die es dem Benutzer ermöglicht, mit Nodes zu interagieren und Transaktionen auszuführen.

Zum Testen der Transaktion auf dem Ledger wird ein Build-Prozess unter Verwendung von Gradle durchgeführt und die Software anschließend ausgeführt.

Nachfolgend in Abbildung 5 eine exemplarische Darstellung des Prototyps:

Abbildung 5

Übersicht über den Prototypen

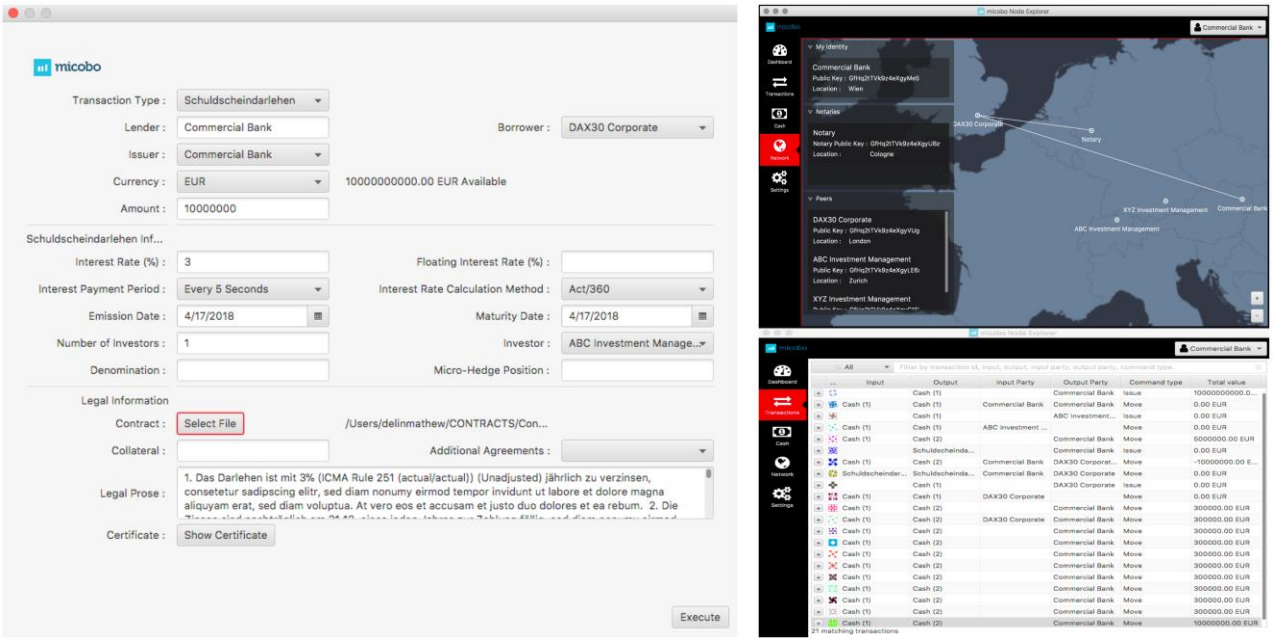


Die Bedienung erfolgt über die GUI mit dem Transaktionstyp „Schuldscheindarlehen“ im Node Explorer. Abbildung 6 zeigt die Benutzerschnittstelle zur Erzeugung der Transaktion. Die in der Benutzerschnittstelle eingegebenen Werte werden in den jeweils zugrunde liegenden Flow integriert und somit in die Transaktion geschrieben. Die ausgeführten Transaktionen werden dann in einer Transaktionsübersicht im Node Explorer dargestellt.

In der zugehören Flow-Logik beziehungsweise dem Smart Contract können nun korrespondierende Befehle definiert werden, welche die Geschäftslogik der Handelstransaktion steuern – beispielsweise Emission, Zins- oder Tilgungszahlungen.

Abbildung 6

Transaktionsausführung und -übersicht im GUI



Abschließende Bewertung und gewonnene Erkenntnisse

Nachfolgend werden die wesentlichen Erkenntnisse bei der Implementierung eines MVP im Financial Services Bereich dokumentiert.

Als wesentliche Feststellung lässt sich als großer Vorteil von Corda nennen, dass elementare Geschäftsvorgänge, beispielsweise „Issuance“ und „Payments“, bereits inhärenter Bestandteil des zur Verfügung gestellten Frameworks sind. Dies gilt darüber hinaus auch für enorm hilfreiche Module, wie dem Node Explorer. Design-Prinzipien wie „Need-to-know“ sorgen für tatsächliche Vertraulichkeit zwischen den Nodes und der zugrundeliegende Technologie-Stack sorgt für hervorragende Interoperabilität mit anderen (Legacy-)Systemen einer Organisation. Darüber hinaus stellt die Flexibilität aufgrund der Einbindung der JVM als zentrales Designprinzip eine Vielzahl an Entwicklungsmöglichkeiten und Programmiersprachen für die Implementierung von Geschäftslogiken bereit. Dementsprechend kommen wir zu dem Entschluss, dass Corda das Framework der Wahl für die Etablierung von „non-public“-Anwendungsfällen im Distributed Ledger Bereich ist.

Trotz unserer guten Erfahrungen mit R3's Corda bleiben dennoch abschließend einige Aspekte von Corda kritisch zu hinterfragen:

- Corda ist trotz des V2.0 Releases noch in der Entwicklung und dementsprechend muss auch der eigene Quellcode regelmäßig angepasst werden. Zudem sind diverse interne Funktionen als „depricated" markiert.
- Nodes sind arbeitsspeicherintensive Prozesse. Dies ist beim Hardware-Sizing des Node-Ecosystems zu berücksichtigen.[25]
- Globale Interdependenzen:
    - Es bestehen erhebliche Abhängigkeiten zu anderen Frameworks. Dies bedeutet gegebenenfalls hohen Aufwand bei Patches bzw. Weiterentwicklungen.
    - Servicemodule der Nodes (z.B. Messaging, Consensus etc.) basieren auf Drittanbieter-Modulen.
    - Weitere Abhängigkeiten zu Drittanbiter-Software bestehen, wie H2 als Datenbank (MySQL/PostgreSQL ist vorbereitet) und JKS als KeyStore Format. Besser wäre hier ein offener Standard.
- Die Node ist der leistungsfähigste Teil des Frameworks und bisher nur auf Singlethreading angelegt.
- Hierdurch werden diverse Elemente (zum Beispiel Events) serialisiert. Offen bleibt, ob dies für bestimmte Anwendungsfälle, wie zum Beispiel Notary Services ausreichend performant ist.

Prof. Dr. Philipp Sandner leitet das Frankfurt School Blockchain Center. Er kann über E-Mail (email@philipp-sandner.de) kontaktiert werden, via LinkedIn (https://www.linkedin.com/in/philippsandner/) und ist auch bei Twitter aktiv (@philippsandner).

Christoph Impekoven ist Gründer und Geschäftsführer der micobo GmbH und beschäftigt sich intensiv mit den Technologien rund um DLT und Crypto Assets im Bereich der Kapitalmärkte.

Jan-Philipp Wiemann ist FinTech Lead Analyst bei der micobo GmbH und hat federführend am gezeigten Prototyp mitentwickelt.

Referenzen

Brown, R. G. (2016, Oktober). R3 Corda: What Makes It Different, https://www.corda.net/2016/10/r3-corda-makes-different/, 18.02.2018.

Brown, R. G., Carlyle, J., Grigg, I., Hearn, M. (2016). Corda: An Introduction, https://docs.corda.net/_static/corda-introductory-whitepaper.pdf, 18.02.2018.

Coll, J. (2017, Februar). Corda Architecture, in London Corda MeetUp, London, England, https://discourse.corda.net/t/corda-architecture/605, 20.02.2018.

Hearn, M. (2016). Corda: A distributed ledger. Corda Technical White Paper, Version 0.5, https://docs.corda.net/_static/corda-technical-whitepaper.pdf, 28.02.2018.

JPMorgan Chase & Co (o.D.). Quorum, https://github.com/jpmorganchase/quorum, 22.02.2018.

Kuhlman, C. (2016, September). From Chaos to Order – Rebranding Eris to Monax, https://monax.io/2016/09/30/from-chaos-to-order, 16.02.2018.

Lindholm, T., Yellin, F., Bracha, G., Buckley, A. (2015). The Java® Virtual Machine Specification - Java SE 8 Edition.

R3 Limited (2017). R3 Corda V2.0 documentation, https://docs.corda.net, 16.02.2018.

---

1 www.r3.com/about
2 Brown (2016)
3 Brown et al. (2016), S. 8.
4 https://docs.corda.net/key-concepts-notaries.html
5 https://docs.corda.net/tutorial-observer-nodes.html
6 https://monax.io/2016/09/30/from-chaos-to-order
7 Clearmatics betreibt die Plattform des Utility Settlement Coins – eine digitale Währung für Deutsche Bank, UBS, Santander, BNY Mellon und der Broker Icap. Der USC wird dabei nicht als konventionelle Cryptocurrency emittiert, sondern soll bei einer Zentralbank in Fiat in Euro / US-Dollar hinterlegt werden.
8 Indirekte Beteiligung durch Bitcoin Sidechains und Hyperledger.
9 https://github.com/jpmorganchase/quorum
10 UTXO: unspent transaction output
11 Hearn (2016), S. 18.
12 Grundsätzlich alle Programmiersprachen, die sich an die JVM Spezifikation halten.
13 Hearn (2016), S. 38.
14 R3 Limited (2017)
15 Hearn (2016), S. 6.
16 In Anlehnung an Coll (2017)
17 http://activemq.apache.org
18 https://docs.corda.net/vault.html
19 https://www.jetbrains.com/idea/
20 https://gradle.org
21 https://docs.corda.net/demobench.html
22 “A TransactionBuilder is a transaction class that's mutable (unlike the others which are all immutable). It is intended to be passed around contracts that may edit it by adding new states/commands. Then once the states and commands are right, this class can be used as a holding bucket to gather signatures from multiple parties” (R3 Limited, 2017)
23 https://docs.corda.net/node-explorer.html
24 https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm
25 RAM-Auslastung von mehr als 750 MB ist die Regel