TU WIEN Informatics

Towards Compliance-By-Design for a Decentralized Business Process Execution Engine

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Thomas Lielacher, BSc
Matrikelnummer 00826260

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Projektass. Dipl.-Ing. Thomas Preindl, BSc
Projektass. Dipl.-Ing. Martin Kjäer, BSc

Wien, 25. September 2023

Thomas Lielacher	Wolfgang Kastner

TU WIEN Informatics

Towards Compliance-By-Design for a Decentralized Business Process Execution Engine

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Thomas Lielacher, BSc
Registration Number 00826260

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Projektass. Dipl.-Ing. Thomas Preindl, BSc
Projektass. Dipl.-Ing. Martin Kjäer, BSc

Vienna, 25th September, 2023

Thomas Lielacher | Wolfgang Kastner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Thomas Lielacher, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 25. September 2023

Thomas Lielacher

Widmung

Ich widme diese Arbeit meinem Vater Reinhard, der uns leider viel zu früh verlassen musste. Du hast mich dazu inspiriert meinen Träumen zu folgen und niemals aufzugeben.

Wir werden dich niemals vergessen.

Dedication

I dedicate this work to my father Reinhard, who unfortunately had to leave us far too early. You inspired me to follow my dreams and never give up.

We will never forget you.

Danksagung

Zuallererst möchte ich meiner Partnerin Jennifer für ihre unendliche Liebe und Geduld danken. Ohne ihre Unterstützung wäre ich nicht so weit gekommen.

Mein besonderer Dank gilt meinem Betreuer Wolfgang Kastner, der es mir ermöglicht hat, Teil der Automation Systems Group zu sein und an ihrer spannenden Forschung teilzunehmen. Ich möchte mich auch bei Thomas Preindl und Martin Kjäer für all ihr Feedback, ihre konstruktive Kritik und unsere anregenden Diskussionen bedanken. Ohne sie wäre diese Arbeit nicht die Gleiche.

Zu guter Letzt möchte ich mich bei meiner Familie und meinen Freunden für gemeinsame Unternehmungen und Abwechslung bedanken, um meine Batterien wieder aufzuladen, damit ich weiterhin mein Bestes für diese Arbeit geben kann.

Acknowledgements

First and foremost, I want to thank my partner, Jennifer, for her never-ending love and patience. I would not have come this far without her support.

Special thanks go to my advisor, Wolfgang Kastner, for allowing me to be part of the Automation Systems Group and participate in their exciting research. I also want to thank Thomas Preindl and Martin Kjäer for all their feedback, constructive criticism, and inspiring discussions. This work would not be the same without them.

Last but not least, I want to thank my family and friends for joint adventures and diversion to recharge my batteries to be able to continue to give my best for this work.

Kurzfassung

Die Blockchain-basierte Ausführung von Geschäftsprozessen hat in den letzten Jahren in Wissenschaft und Industrie viel Aufmerksamkeit erregt. Blockchains und andere Distributed-Ledger-Technologien ermöglichen ein vertrauensbasiertes Ökosystem, das den Bedarf an Vermittlern reduziert, indem es eine sichere und transparente Möglichkeit zur Aufzeichnung, Überprüfung und gemeinsamen Nutzung von Daten zwischen den beteiligten Parteien bietet. Darüber hinaus verbessert die Unveränderlichkeit verteilter Ledger die Datenintegrität und Nachvollziehbarkeit, was es für Unternehmen einfacher macht, Bedingungen und Vorschriften einzuhalten. Diese Arbeit zielt darauf ab, bestehende Ansätze zu verbessern, indem die Datenperspektive eines kollaborativen Geschäftsprozesses berücksichtigt wird. Wir ermöglichen es, Einschränkungen für diese Daten zu definieren, um robuste Prozesse zu definieren, die mit den relevanten Vorschriften konform sind.

Auf der Grundlage einer Design Science-Methodik stellen wir ein Konzept für eine Rule-Engine vor, die in bestehende Lösungen integriert werden kann, um den geteilten Prozessstatus und die ausgetauschten Nachrichten auf Konformität zu überprüfen, ohne dass dafür Code in einer Programmiersprache wie Solidity geschrieben werden muss. Auf der Grundlage dieses Konzepts implementieren wir einen Prototyp, der die Sprache Friendly Enough Expression Language (FEEL) des DMN-Standards verwendet, die speziell entwickelt wurde, um für Entwickler und Fachleute leicht verständlich zu sein und so ein gemeinsames Verständnis fördert. Um den praktischen Nutzen unseres Prototyps zu zeigen, implementieren wir die Integritätseinschränkungen von realen Geschäftsprozessen. Außerdem integrieren wir ihn in eine bestehende Blockchain-basierte Engine zur Ausführung von Geschäftsprozessen.

Unsere Ergebnisse zeigen, dass es möglich ist, eine solche Rule-Engine zu implementieren und dabei essentielle Eigenschaften wie *Terminierung*, *Determinismus* und *Verifizierbarkeit* zu erhalten. Anders verhält es sich mit der *Erweiterbarkeit*. Diese Eigenschaft ist zwar erforderlich, um unterschiedlichste Geschäftsprozesse zu realisieren, kann aber die zuvor genannten Eigenschaften erheblich abschwächen.

Abstract

Blockchain-based business process execution has gained much attention from academia and industry in recent years. Blockchains and other Distributed Ledger Technologies enable a trust-based ecosystem that reduces the need for intermediaries by providing a secure and transparent way to record, verify, and share data among the collaborating parties. Moreover, the immutable nature of distributed ledgers enhances data integrity and auditability, making it easier for organizations to comply with terms and regulations. This work aims to improve existing approaches by considering the data perspective of a collaborative business process. We enable it to define constraints on this data to define robust processes compliant with the relevant regulations.

Following a Design Science methodology, we introduce a concept for a rule engine that can be integrated into existing solutions to verify the shared process state and the exchanged messages for compliance without the need to write code in a programming language such as Solidity. Based on this concept, we implement a prototype utilizing the Friendly Enough Expression Language (FEEL) of the DMN standard, specifically developed to be easy for developers and business professionals to understand, thus fostering a common understanding. To show the utility of our prototype, we implement the integrity constraints of real-world business processes. Further, we integrate it into an existing blockchain-based business process execution engine.

Our results show that it is possible to implement such a rule engine while maintaining core properties, such as *termination*, *determinism*, and *verifiability*. The situation is different with *extensibility*. While this property is required to realize most diverse business processes, it can significantly weaken the properties mentioned before.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

Since the introduction of Bitcoin in 2009, cryptocurrencies and other Distributed Ledger Technologies (DLTs) have gained much attention — not only for financial applications. Academia and industry alike envision Distributed Ledger Technologies as a key technology for a broad domain of use cases, including the Internet of Things (IoT), supply chain management, and self-organized network orchestration. This interest is mainly caused by their fundamental properties, such as tamper evidence, immutability, and transparency. DLTs ensure all these properties without relying on a trusted third party [RGG+18, WHH+19]. Another active field of study is business collaboration by leveraging DLTs [VXBP19]. Blockchain-based business process execution engines can take advantage of the properties mentioned earlier to support the execution of business processes between potentially untrusted parties in a decentralized environment [LPGBD+19, MGH+18]. Such systems benefit from improved documentation and increased transparency and traceability for all participants without needing a trust relationship between them [BSS21].

When designing such blockchain-based systems, a challenge is to decide whether the data can be stored on-chain or off-chain. By its nature, all the data stored on the blockchain is replicated to every participating network node. This redundant data storage has implications for data privacy. Collaborating organizations might not want to keep all the relevant data on the blockchain as it may contain sensitive business information. Different approaches have been proposed to overcome this issue [EH18]. The Baseline protocol[1] is the first attempt to standardize the synchronization of the state of a system, e.g., a process execution engine, over the internet by combining advances in cryptography, such as zero-knowledge proofs, messaging, and using a blockchain as a common frame of reference. Two research projects of TU Wien, FMCHAIN and DiCYCLE, try to explore

[1] https://docs.baseline-protocol.org/

the applicability of Distributed Ledger Technologies in the architecture, engineering, and construction industry by employing a decentralized blockchain-based business process execution engine and Building Information Modeling (BIM) to automate processes among different organizations.

An open problem in such replicated and decentralized process execution engines is ensuring compliance-by-design, i.e., forcing all state transitions to comply with some predefined rules. Business protocols and legal regulations are already considered when the business process is modeled. Especially in such trustless, decentralized environments, compliancy-by-design can be an essential property to improve collaboration among the participants [SGN07, Loh13, GV06b]. As every party can 'propose' a new state of the process execution, it should be possible to define rules based on the current state of the system to specify if a state transition is valid or not and can be integrated into the shared state. This concept is comparable to that of cryptocurrencies. Every cryptocurrency defines a protocol with rules that must be fulfilled by a transaction to be included in a block.

Existing blockchain-based approaches such as Caterpillar by López-Pintado et al. [LPGBD+19] that make use of BPMN models to generate smart contracts are limited to the possibilities of BPMN to express such rules. Thus, more complex conditions must be manually implemented, which can be complex and error-prone and may cause additional effort. Existing non-blockchain-based concepts like the Formal Contract Language by Sadiq et al. [SGN07] and PENELOPE by Goedertier et al. [GV06b] may not be directly applicable in a replicated and decentralized system as there emerge additional requirements and restrictions in such environments.

This work aims to discuss the shortcomings of the aforementioned existing solutions and close this gap by proposing a system together with a simple rule language to define preconditions for the state transitions of a decentralized business process execution engine. The benefit of such a rule engine is that compliance-by-design is enforced and verifiable for all parties. Additionally, process models enriched with compliance rules are better reusable because the system participants can more easily adapt rules if laws or other regulations change. Furthermore, the rules can be written by Business Analysts, thus further reducing the software engineering effort for introducing an inter-organizational process execution engine [RD05].

1.2 Aim of the Work

In this thesis, we propose a rule engine for a decentralized process execution engine with off-chain state transition messages. This rule engine shall bridge the gap between abstract compliance rules and their implementation in terms of compliant blockchain-based business process execution. The mapping of concepts from business process modeling to such a decentralized and blockchain-based process execution eases the amount of work necessary to implement new business processes. Thus, it explores the possibilities of ensuring compliance-by-design in decentralized and trustless environments.

The overall goal is to present a new approach to how a rule engine that can be used to verify state transitions for compliance can be integrated into a blockchain-based business process execution engine. This is accomplished by comparing the shortcomings of existing blockchain-based and non-blockchain-based concepts and proposing a new approach to close this gap. Implementing a prototype based on the concept shows such a system's technical feasibility. The prototype of the rule engine will be integrated into the existing prototypical business process execution engine of FMCHAIN and DiCYCLE. Process models need to be enriched by constraints encoding the compliance rules to do so. These rules can range from simple access control conditions to more complex ones involving the current state of the system and the changes that a specific state transition would cause. If one of the participants causes a modification of the process state, the execution engine can then call the rule engine to determine whether this transition is compliant. If so, the state change can be replicated to the other participants of the decentralized business process. The process engine replicas will also check the state change using the rule engine and change their local state accordingly if the validations succeed. This way, all non-faulty replicas of the process execution engine will eventually enter a consistent and compliant state.

Finally, the following questions will be answered:

- *RQ1*: Which requirements do a rule engine for a blockchain-based decentralized business process execution engine have to meet?
- *RQ1.1*: Which rule engine properties are necessary for the execution of business processes?
- *RQ1.2*: Which properties must be fulfilled by the business process execution engine so that the rule engine can provide its full utility?
- *RQ2*: Which technologies and approaches are suitable for the concept of the rule engine, and what combinations or adaptations are necessary to satisfy the defined requirements?

1.3 Methodology

Design Science The overall methodology follows a Design Science approach based on Hevner et al. [HMPR04]. Figure 1.1 depicts the basic framework of this method. The primary artifacts of this process are the literature review, the requirements the rule engine has to fulfill, and a conceptual model for the rule engine and its interaction with a business process execution engine.

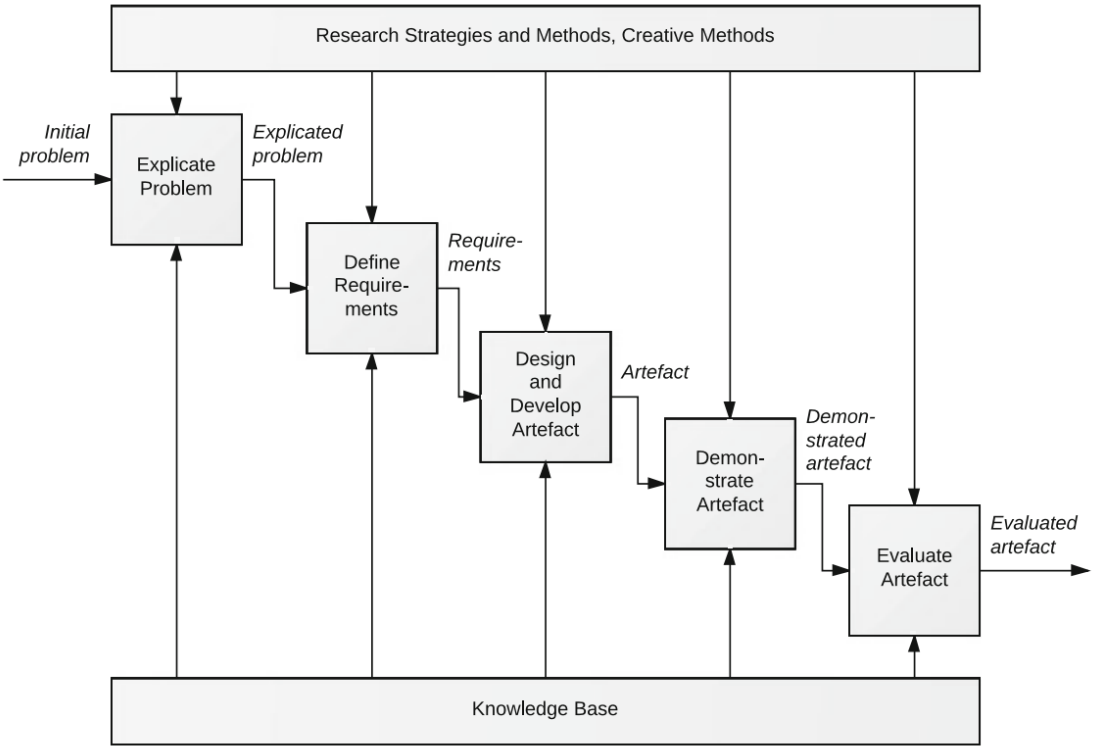


Figure 1.1: The method framework for Design Science [JP14]

Literature Review The literature review will serve multiple purposes:

- It is performed to establish background knowledge on the research areas that are touched by the thesis. These include Distributed Ledger Technologies, state machine replication, business process modeling, artifact-centric modeling, and trust management languages.
- It will be used to identify the state-of-the-art rule engines for blockchain-based and non-blockchain-based business process engines and to compare existing concepts.
- It will be the primary source of non-functional requirements for the rule engine.

Thus, the literature review is needed to answer parts of all research questions, but especially for *RQ1* and *RQ1.2*.

We conduct a narrative literature review as described by Rhoades [Rho11]. It will cover theoretical information about *compliance-by-design* and existing approaches to how this can be reached in process-centric and artifact-centric business processes. We further explore and compare existing solutions for blockchain-based business process execution.

We also cover some fundamental literature on State Machine Replication to identify properties relevant to the rule evaluation.

Requirements Engineering We will perform requirements engineering to identify functional requirements for the system. As the main focus of this work is to create the conceptional model, we will define three use cases together with the project partners from the research group *Integrale Bauplanung und Industriebau* to also ensure functional feasibility and to be able to answer *RQ1* and *RQ1.1*.

For the requirements engineering, we will follow a top-down approach. We first define the most fundamental non-functional requirements for the rule evaluation. The primary source of these is the literature review. The use cases will serve as a source for the functional requirements for the rule evaluation in general and for the expression language used to define the constraints in particular.

Prototyping Based on the concept, we implement a prototype for the Rule Evaluator to answer research question *RQ2*. This step includes specifying a rule language that we use to implement the compliance rules of the use cases used for requirements engineering. Additionally, the prototype will be evaluated against the requirements from *RQ1* and integrated into the prototype of a decentralized business process execution engine used for the FMCHAIN and DiCYCLE projects. We will choose suitable methods for this evaluation based on the requirements.

Other methods will be employed as needed to conceptualize and implement the prototype. These include business process modeling, artifact-centric modeling, UML, and other state-of-the-art software engineering methods.

1.4 Structure of the Thesis

The remainder of this work is structured as follows: in Chapter 2, we briefly introduce Distributed Ledger Technologies and explain some fundamental concepts of the currently largest cryptocurrencies: Bitcoin and Ethereum. Further, we acquaint the reader with business process modeling and how Distributed Ledger Technologies can be used to reduce trust issues in potentially trustless collaborative business processes. Section 2.4 presents state-of-the-art research in relevant areas.

In Chapter 3, we first introduce the use cases used for requirements engineering and the prototype implementation later. After that, we list the requirements derived from the literature review and the use cases. The requirements will also be the answer to *RQ1*. We further present the concept of our rule engine and how it can be integrated into a decentralized blockchain-based business process execution engine. Additionally, we define which properties the execution engine has to bring so that the rule engine can work as intended, thus answering *RQ1.1* and *RQ1.2*.

Based on the concept from Chapter 3, we implement a prototype in Chapter 4. We discuss suitable technologies and show which design goals we followed during the implementation to answer *RQ2*.

The concept and prototype evaluation in Chapter 5 is split into three parts. First, we use the prototype to implement the constraints from the use cases of Chapter 3 to show the practical utility of our approach. After that, we verify the integrability of our prototype by integrating it into an existing prototype of a decentralized business process execution engine. The last part is a qualitative evaluation of whether the requirements from Chapter 3 are fulfilled. We conclude the chapter by discussing the presented evaluation results and giving a short overview of open problems and further improvements for future work.

Chapter 6 concludes the thesis by summarizing and discussing the presented results.

CHAPTER 2

Background and State-of-the-Art

2.1 Distributed Ledger Technologies

Bitcoin and most other cryptocurrencies belong to the group of Distributed Ledger Technologies (DLTs). Rauchs et al. define a DLT as "[...] a system of electronic records that enables independent entities to establish a consensus around a shared 'ledger' — without relying on a central coordinator to provide the authoritative version of the records." [RGG+18, p. 23] The authors further list five properties a DLT must provide:

- **Shared recordkeeping**: Multiple parties are enabled to create, maintain, and update a shared set of records.
- **Multi-party consensus**: In contrast to, for example, classical banking, where all transactions have to go through a trusted central party, all parties have to come to an agreement about the ordering and validity of transactions.
- **Independent validation**: Every party is able to verify the validity of transactions and the integrity of the whole system.
- **Tamper evidence**: Every party is able to detect changes to the data that did not follow the consensus protocol.
- **Tamper resistance**: It is infeasible for a single party to change past records.

Since its first introduction in 2009, Bitcoin has gained much attention and led to the development of a massive number of similar systems — not only for financial applications. Some of these new blockchains share most of their properties with Bitcoin and differ just in some details. Others employ entirely new algorithms and pursue different goals than being only a medium for transferring values. Some do not even use a blockchain in the

general sense anymore[1]. Academia and industry envision Distributed Ledger Technologies as a key technology for several domains, including the Internet of Things (IoT), supply chain management, and inter-organizational business collaboration [RGG+18, WHH+19].

The following two sections briefly overview the current two largest cryptocurrencies based on their market capitalization: Bitcoin and Ethereum.

2.1.1 Bitcoin

The concept of Distributed Ledger Technologies existed before Bitcoin and other blockchain technologies. However, Satoshi Nakamoto was the first to propose a peer-to-peer network, ensuring all the properties as mentioned earlier and enabling to transact electronic values without relying on a trusted third party. In traditional banking, one of the primary responsibilities of the trusted central authority (the bank) is to prevent double-spending, i.e., to avoid a digital token from being spent more than once. Consequently, this central party must be trusted by all other participants (thus the name 'trusted third party') and represents a single point of failure. If it stops working, none of the other participants can send transactions anymore [Nak09].

Bitcoin solves this problem elegantly via its consensus protocol, also called *Nakamoto consensus* or *Proof-of-Work*. The protocol can be divided into different parts that, in combination, ensure the security of Bitcoin.

Proof-of-Work Special nodes of the network, the so-called *miners*, are competing to solve a sufficiently hard and computationally expensive problem. They need to find a value (*nonce*) that, when hashed together with the other fields contained in the block header of the current block, results in a hash value below a certain threshold. If such a nonce is found, the mining node sends the resulting block to the other nodes in the network. The other nodes can validate the block, e.g., if all transactions and the nonce are valid. The threshold, or *difficulty*, is adjusted every 2016 blocks, depending on the average computational power of the whole network to mine these blocks. The goal is to find a new block every ten minutes on average [Nak09, WHH+19].

Network Protocol Each mining node must follow a clearly defined protocol. The nodes broadcast new transactions to all other peers via a simple gossip protocol[2]. The transactions are collected into a new block. Simultaneously, the node works on finding a solution for the hash puzzle, as already mentioned above. When the node finds a solution, it broadcasts the block to all other peers. If the block is valid, the other nodes start working on the next block using the accepted block as a predecessor. Thus, the

[1] IOTA, for example, uses a concept called *tangle*. Transactions are not grouped into blocks anymore but form a directed acyclic graph.

[2] As it is not feasible for a node to create connections to all other nodes of the network, a node connects to a relatively small number of peers. If a new transaction or block is received, it will forward it to all nodes it is connected to, which will in turn forward it to the nodes they are connected to. This way, the information spreads in the network, eventually reaching all nodes.

longest chain is considered correct and will be extended by all honest nodes. If two nodes broadcast two competing blocks simultaneously, the tie will be broken when the next block is found extending one of the two candidates. Thus, if honest nodes control the majority of the computing power, the valid chain will grow the fastest and outpace any conflicting branch. To change a past block in this longest chain, an attacker would have to recalculate the Proof-of-Work for this block and every block after and catch up with the current longest chain [Nak09, WHH+19, RGG+18].

Incentives Incentives refer to the rewards and motivations that encourage individuals and entities to participate in the Bitcoin network, particularly in activities such as mining and transaction validation. The first transaction of a block, the *coinbase transaction*, contains a reward for the creator of the block by creating a predefined amount of new coins. The amount was 50 Bitcoin when it was launched and is halved every 210,000 blocks, making Bitcoin deflationary and limited to 21 million coins. Additionally, transactions can include a small *transaction fee* for the block's miner. It is assumed that the transaction fees will become more and more relevant with each Bitcoin halving to compensate the mining nodes for their work [Nak09, WHH+19].

Blockchain Figure 2.1 shows a simplified illustration of Bitcoins blockchain datastructure. Each block consists of a header and the transactions contained in the block. The header contains a hash pointer to the preceding block, metadata, the nonce, and the Merkle root of the transactions. A hash pointer is a pointer together with the hash of the data the pointer is targeting. The nonce is the result of the Proof-of-Work, as mentioned above. The metadata contains additional information about the block, such as a timestamp when the block was mined and a block number. The Merkle root is the root hash of a Merkle tree of the transactions of the block, as shown in Figure 2.2. The tree's leaf nodes are labeled with the hash value of a transaction. The inner nodes are labeled with the hash of the concatenation of the labels of its children. The data structure allows other participants to efficiently check if a specific transaction is part of a block [Nak09, WHH+19].

Accounts Bitcoin is a public, permissionless blockchain. This means everyone can operate a node in the peer-to-peer network and create arbitrarily many accounts. Bitcoin makes use of public-key cryptography. To create an account, a user only has to create a public-private key pair and derive the account address from the public key. A valid transaction contains, among other information, the addresses of the sender and receiver, the sender's public key, and is signed with the sender's private key. The signature ensures that other participants can verify the validity of the transaction. Unlike other cryptocurrencies, Bitcoin does not store the balance of an account but uses the Unspent Transaction Output (UTXO) model. A transaction in that model can have multiple inputs and multiple outputs. The inputs consume existing UTXOs, and the outputs produce new UTXOs. Hence, to determine the balance of a particular address, one has to sum up all unspent transaction outputs of that address [Aca22].

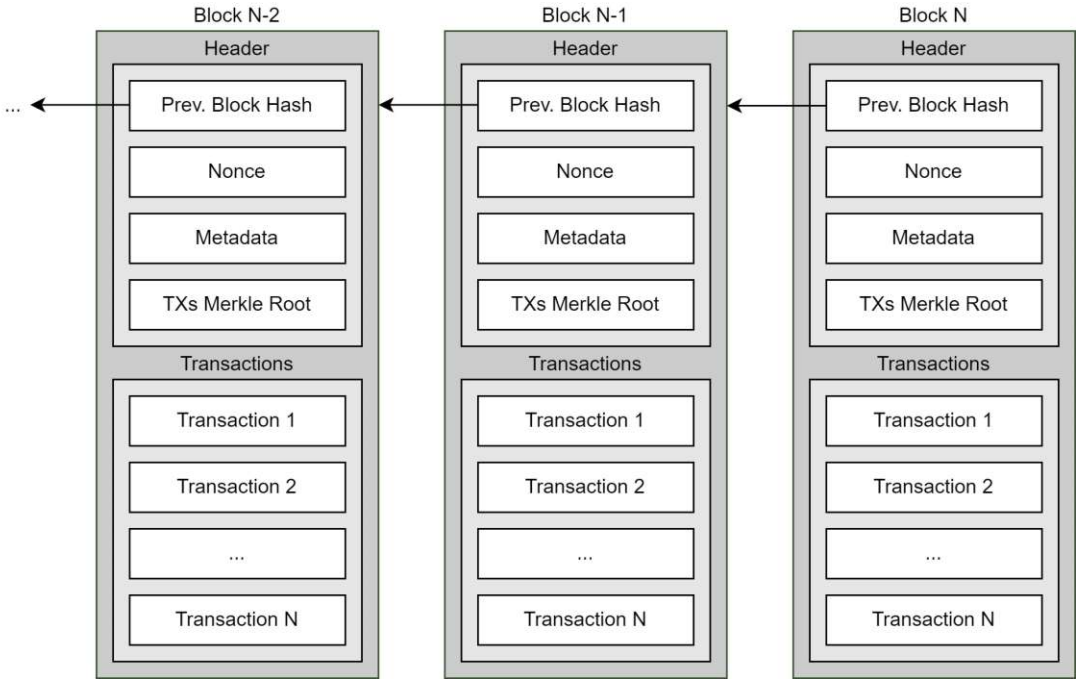


Figure 2.1: A simplified representation of the Bitcoin blockchain

2.1.2 Ethereum

As mentioned, Ethereum is currently the second-largest cryptocurrency behind Bitcoin based on market capitalization. It was introduced by Vitalik Buterin in 2014 as a "next-generation smart contract and decentralized application platform" [But20, p. 23]. Ethereum is an open-access and decentralized blockchain network. Ether (ETH) is the native cryptocurrency of the platform. Although it shares some principles with Bitcoin, it is often referred to as 'second-generation blockchain'. Besides the ability to transfer Ether between two accounts, Ethereum also provides a Turing-complete execution environment, the Ethereum Virtual Machine (EVM), enabling the development of multi-purpose decentralized applications (dApps) [But20].

Accounts In contrast to Bitcoin, Ethereum is an account-based blockchain. It means that the fields of an account are stored as part of the state of the blockchain. Transactions can be seen as state transitions of these accounts. Each account is uniquely identified via a 20-byte address. Ethereum also knows two kinds of accounts: externally owned and contract accounts. Externally owned accounts are controlled by their associated private key, and their state consists only of the balance. In contrast, contract accounts are controlled by so-called 'smart contracts'. Their state includes the byte code and the contract storage [But20].

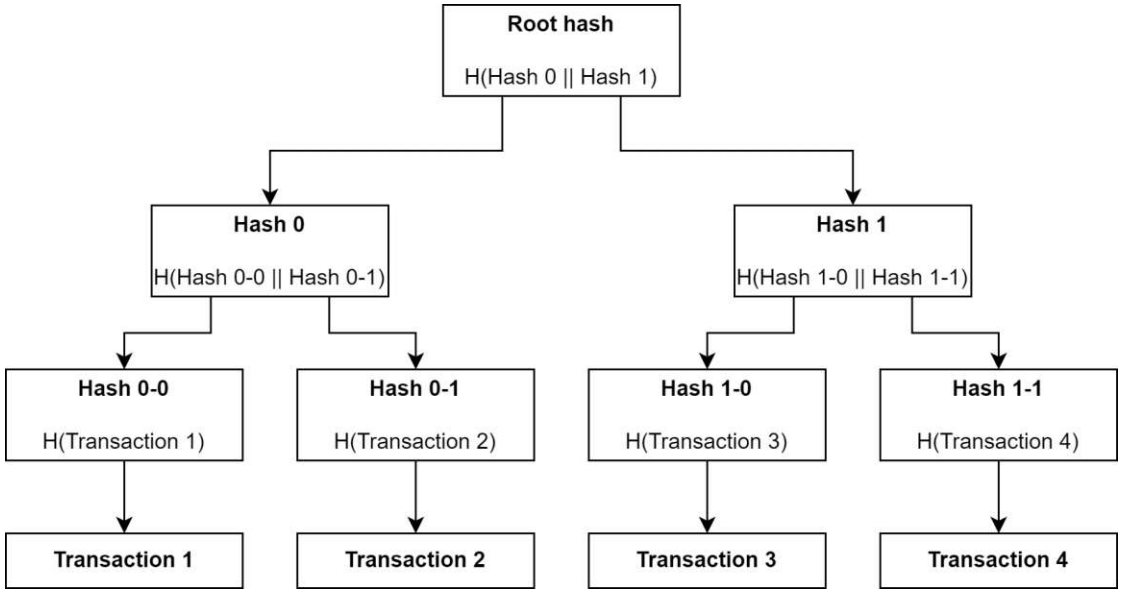


Figure 2.2: Graphical representation of a Merkle tree.

Smart Contracts Smart contracts are pieces of code stored on the blockchain and can be called by externally owned accounts or other smart contracts. They have direct control over their own Ether balance and storage to keep track of persistent variables. Via transactions, an externally owned account can call a smart contract function. The code for smart contracts can be written in a high-level programming language (mainly Solidity or Vyper) and is compiled into a bytecode, which is then stored on the blockchain. The EVM executes this bytecode, i.e., every verifying node in the network executes it. Ethereum uses a concept called *gas* to prevent non-termination and denial-of-service attacks. Gas is used to quantify the amount of computational resources a particular operation or transaction consumes. Each operation or instruction in a smart contract has a gas cost associated with it, and transactions must provide a certain amount of gas to cover these costs. If a transaction or smart contract execution runs out of gas before completion, it will be reverted, and any changes made during execution will be discarded. The gas can be 'bought' using Ether. Some restrictions must be considered when developing a smart contract. As the code execution must be fully deterministic, for example, synchronous I/O operations such as network communication are impossible [But20, Woo22].

Applications Ethereum allows the implementation of a wide range of applications. They range from games over financial applications such as Decentralized Exchanges (DeX) and Decentralized Finance (DeFi) to enterprise applications. One of the most important aspects is the ability to create tokens on top of Ethereum. It is possible to define a custom token based on one of the multiple token standards[3] within a few lines of code. These tokens are often used as governance or utility tokens.

2.1.3 Permissionless versus Permissioned Distributed Ledger Technologies

Bitcoin and Ethereum are permissionless blockchains, i.e., any peer can join or leave the network at any time and participate in the consensus-finding process at will. In such open networks, a mechanism has to be integrated into the consensus protocol to prevent Sybil attacks[4]. Proof-of-Work and Proof-of-Stake are well-known examples. These mechanisms increase the security of the blockchain, but in general, the transaction throughput is reduced (cf. Section 2.1.4) [RGG+18, WG18].

On the other hand, permissioned DLTs only have a limited set of validating nodes. A central authority manages them. Due to the comparably small numbers of nodes participating in the consensus-finding, these networks can use traditional consensus algorithms such as Practical Byzantine Fault Tolerance (PBFT). The benefits are immediate transaction finality and higher throughput compared to permissionless blockchains. However, due to the smaller number of validation nodes, permissioned blockchains can have reduced security properties. A powerful malicious actor could compromise a certain number of validators and manipulate the blockchain to its benefit. Popular examples of permissioned DLTs are R3 Corda and Hyperledger [RGG+18, WG18, WHH+19].

2.1.4 Drawbacks

The blockchain industry and research are still in an early stage, and several challenges and drawbacks must be considered. Swan [Swa15] argues that the central challenges are scaling up permissionless blockchain networks' transaction throughput and improving transaction finality. Bitcoin can currently process up to 7 transactions per second. This relatively small number directly results from the average block time of ten minutes and the maximum block size. Ethereum performs better and can currently process up to 25 transactions per second. Compared to centralized payment providers, there is much room for improvement. Visa claimed 2016 to process 1,700 transactions per second on average and up to 10,000 transactions at its peak.

[3] A token standard defines a common interface for all smart contracts implementing a token. The interface defines a set of operations, for example, to query an account's current balance or transfer tokens from one account to another. If a token conforms to one of the standards, it can be traded on exchanges without adaptions and managed by wallets supporting the standard. The most popular token standards are *ERC-20* for fungible tokens and *ERC-721* for Non-Fungible Tokens (NFTs).

[4] Rauchs et al. define a Sybil attack as "a class of attack in which a malicious actor gains influence or disguises malfeasance by creating numerous false identities." [RGG+18, p. 59]

Moreover, Bitcoin provides only probabilistic transaction finality[5]. Hence, to consider a transaction completed, an involved party usually waits for up to six additional blocks to be added to the blockchain. In other words, the party has to wait about 60 minutes to be sure the transaction is final. Since Ethereum was upgraded to Proof-of-Stake, a transaction reaches finality after two so-called "epochs". An epoch is a period of 32 slots, each taking 12 seconds. Usually, a transaction can be considered final after about 15 minutes [Cona].

Unfortunately, it is still being determined if the mentioned challenges can be solved completely. Vitalik Buterin coined the term 'blockchain trilemma'. The trilemma states that trade-offs between the three properties, security, decentralization, and scalability, are inevitable. While it is possible to achieve two of the three properties, the remaining one can not be reached anymore (i.e., you can only choose one of the three sides of the rectangle as shown in Figure 2.3: scalability and decentralization, scalability and security, or security and decentralization) [HHS20, Fou19].

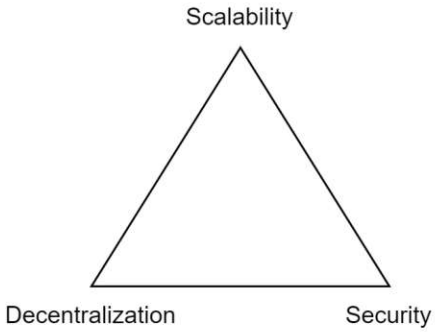


Figure 2.3: The blockchain trilemma

Different solutions have been proposed to improve the scalability of blockchain networks. They can be grouped into first-layer solutions and second-layer solutions. First-layer solutions propose modifications to the blockchain protocol itself. Examples that are already implemented by some cryptocurrencies are increasing the block size (e.g., Bitcoin Cash) or switching from a linear blockchain to a directed acyclic graph (e.g., IOTA). Another promising approach is the concept of 'sharding'. The idea behind sharding is to split up the network into subsets ('shards'). Each shard is then responsible for a different set of transactions. This mechanism allows the network to scale with the number of shards [HHS20].

[5]Transaction finality refers to the moment when it becomes infeasible to revert or alter a transaction that has been added to the blockchain. The opposite of probabilistic finality is deterministic or immediate finality.

Second-layer solutions are implemented outside the blockchain (or 'off-chain'). The blockchain itself is only used as a common source of truth. Examples of this approach are payment channel networks (such as Lightning[6] and Raiden[7]) or rollups[8]. The benefit of layer 2 solutions is that there need not be any change to the consensus algorithm of the blockchain. Ethereum, for example, had planned to use sharding for its consensus layer. However, this idea was dropped due to the rapid development of other second-layer scaling solutions [Conb].

Another drawback that is worth mentioning is the possibility of hacking smart contracts. While the quasi-Turing-completeness of the Ethereum Virtual Machine enables interesting new opportunities, it also opens a space for new categories of software vulnerabilities. There are already numerous cases of hacked smart contracts that caused a loss of cryptocurrencies with a value of several million dollars. Probably the most prominent hack was the so-called 'The DAO hack' [Fin16]. The challenge in this area of software development is that once deployed on the blockchain, the code of a smart contract cannot be changed anymore. These events led to a substantial scientific interest in the security analysis of smart contracts [CPNX20].

2.1.5 Off-chaining

When developing a decentralized application (dApp), an important consideration is deciding what data must be stored on the blockchain and what can be stored off-chain. The same holds for computations. Optimizing the amount of on-chain storage and computation can have considerable advantages. As mentioned, Ethereum supports the development of smart contracts with a Turing-complete execution environment (the EVM). Each instruction of the EVM costs a predefined amount of gas. The most expensive instructions are the computation of the smart contract itself and writing operations to its permanent storage [But20, Woo22].

Considering the high costs of some EVM instructions, the first advantage of off-chaining is reducing costs for the application's users, the application provider, or both. Another advantage is that most off-chaining techniques increase the overall scalability and performance of blockchain-based systems (cf. Section 2.1.4). Last, off-chaining data or computations can guarantee the confidentiality of sensitive or private data, especially in inter-organizational collaboration. Nevertheless, it must be ensured that none of the fundamental properties of the blockchain and blockchain-based applications are violated. These are integrity, availability, and no need for trust in any of the participants [ET17, EH18].

[6] https://lightning.network/

[7] https://raiden.network/

[8] Rollups allow transactions to be executed outside of layer 1 and then the transaction data is posted to layer 1 where consensus is reached [Conb].

Off-chain Storage Off-chaining storage is defined as storing data on a node that does not have to be part of the blockchain network itself. Before the state transition of a smart contract can be performed, the relevant data has to be received from this storage node. After the transition, the potentially altered state is again persisted on the storage node [ET17].

Off-chain storage imposes significant challenges. Current blockchain technologies such as Ethereum rely on the state transitions being fully deterministic [Woo22]. This is why accessing external resources (e.g., a web service or an external storage system) is impossible. Several solutions exist for this restriction. The smart contract could rely on some external party writing the data to the blockchain (a so-called 'oracle' [Pro21]). Unfortunately, this approach is only suitable for small amounts of data, which may also change independently of the execution of a smart contract (e.g., stock market data). Another approach would be to only store references to the data and leave the retrieval to the client application. The challenge here is to retain data integrity. Maintaining data integrity can be solved by using content-addressable storage systems [ET17]. The address of the data in the storage system is also its hash value. When retrieving the data, the client can recalculate the hash code and compare it to the address to check for illegal modifications.

A significant disadvantage of off-chain storage is the reduced availability of the data. Blockchain systems are highly available as every full node stores the entire state. By off-chaining data, this guarantee is lost. While it can be improved by using redundant, decentralized storage systems, the liveness guarantee is still weakened [ET17, EH18].

Off-chain Computation Eberhardt et al. defines off-chain computation as "the execution model where the state transition function is computed by an Off-chain node, and the resulting state is then persisted on-chain after verification of the computation of the state transition" [EH18, p. 8].

Intuitively, this means that the computation is partially or entirely outsourced to an untrusted third party. This enables the use of private data as input for the computation. Unlike off-chaining storage, the liveness of the system can still be guaranteed. Nevertheless, caution is advised because naively off-chaining computations could introduce a trust problem. After all, the external party could lie about the result. To fix the trust problem, the calculation must be verifiable on-chain. Various approaches have been proposed for that.

The *verifiable off-chain computation* approach describes a technique where a prover executes the computation and publishes the result, including a proof certifying the correctness. A verifier should then be able to verify the correctness by performing a relatively cheap verification without interactively communicating with the prover and without gaining any knowledge about the computation inputs. This can be realized using non-interactive zero-knowledge proofing schemes such as zk-SNARKS or Bulletproofs.

Another interesting approach is *secure multi-party computation-based off-chaining*. The private input data is split into multiple parts and distributed to a set of off-chain nodes. The nodes then compute the state transition based on their shares and publish the results afterward. This scheme ensures that a node never has full access to the secret data in its entirety. A fundamental property the protocol has to fulfill is auditability, meaning that the correctness of the computation can be verified [EH18].

2.1.6 Baseline Protocol

The Baseline Protocol is an open-source initiative to create a framework that enables confidential and complex inter-organizational collaborations. It does so by using advances in Distributed Ledger Technologies and cryptography. A Baseline Protocol Implementation (BPI) provides mechanisms to synchronize state using a public blockchain as a common frame of reference. It is developed as an OASIS open standard [Bas].

Figure 2.4 shows an illustrative example of the interactions of a buyer and seller using a Baseline Protocol Implementation. The master service agreement between the two parties is implemented based on a BPI that contains all the agreed-upon terms, such as billing terms, pricing, or discounts. The states of the ERP systems of the buyer and seller are synchronized over the BPI. Based on the mutually agreed-upon master service agreement, the buyer creates an order. A cryptographic proof validates the compliance of the order with the terms of the contract. The proof and the order are sent to the seller, who can validate the proof. If the proof is validated positively, the seller also creates a cryptographic proof of the acceptance of the order. Like before, the acceptance and the cryptographic proof of compliance are sent to the buyer [Bas].

With the master services agreement implemented using a BPI, the buyer and seller can rely on the compliance of the business process to the previously agreed-upon terms. Without the BPI, the parties would have to validate the data after the reception, often manually. This can lead to a time-consuming back-and-forth between the buyer and seller until they reach a satisfactory state.

The Baseline Protocol makes use of the previously mentioned off-chaining techniques to ensure data privacy (cf. Section 2.1.5). If a state synchronization is required, the BPI creates a cryptographic proof of the validity of the state object and commits it to the blockchain. This cryptographic proof can either be the Merkle root of the data or a zero-knowledge proof verifying the compliance of the data to some predefined rules. The state object is sent to the receiver with the cryptographic proof via a secure off-chain messaging protocol (e.g., NATS). This ensures that the public blockchain will never contain any sensitive data and that it is impossible to observe what process is being executed and who the actors are.

The Baseline Protocol specification consists of three separate OASIS open standards, all defining a set of requirements a Baseline Protocol Implementation has to fulfill.

The **Core specification** [FOT21b] is the most important one. It defines the essential concepts of the Baseline Protocol. Most importantly, it specifies a reference architecture,

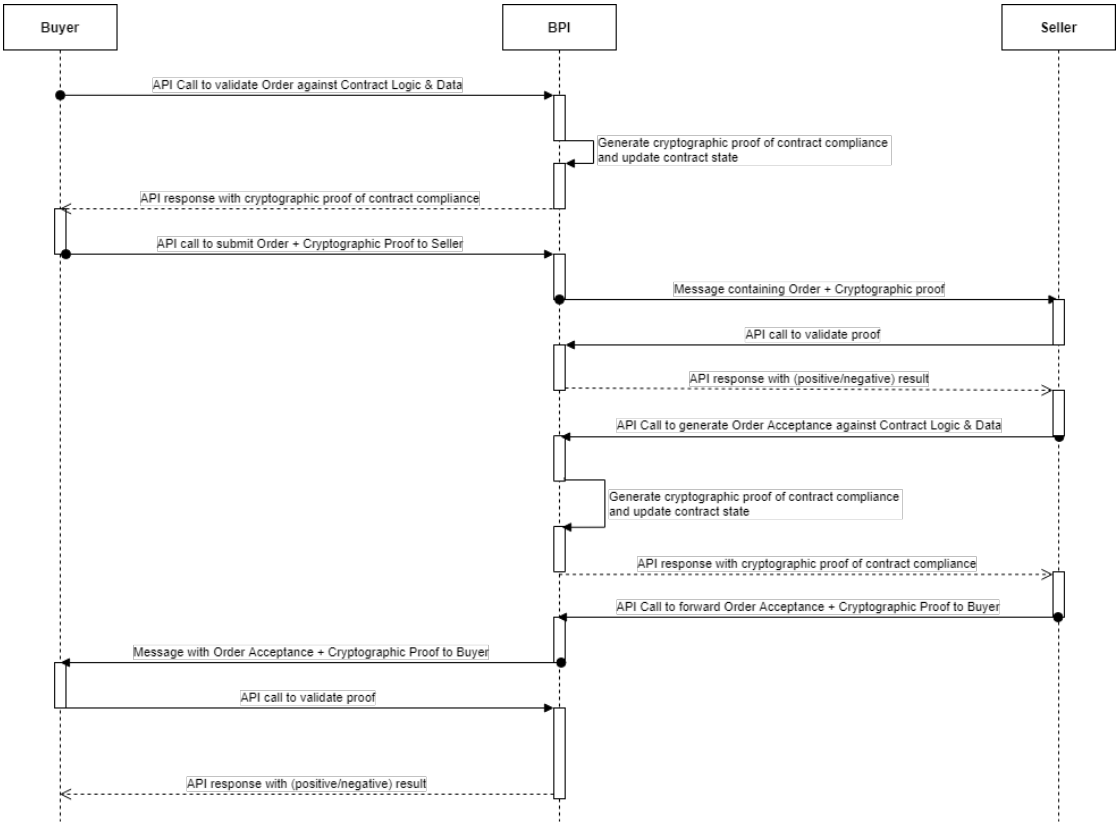


Figure 2.4: Illustrative example of the interaction of a buyer and seller via a BPI [Bas]

the layers the architecture is comprised of, and their interactions. Figure 2.5 illustrates the layers of the architecture. The *BPI Abstraction Layer* defines a set of APIs that can be used by external applications to interact with the BPI. The *Middleware Layer* manages the messaging between the counterparties and the agreed-upon workflow definition. The correct and verifiable execution of the business process is ensured by the *Processing Layer*. The *Consensus-Controlled State Machine (CCSM)*[9] *Abstraction Layer* enables the interaction of the layers above with one or more CCSMs. The Middleware and Processing Layers are the most interesting parts of the specification for our intended rule engine. We will, therefore, describe these in more detail. For a detailed description of the other layers, please refer to the Baseline Core Specification [FOT21b].

Middleware Layer The Middleware Layer defines all the capabilities of an authorized user of the BPI, a so-called *BPI Subject*. This involves the definition of the workflows and the agreement of the counterparties to a set of business rules. It further defines how

[9]A CCSM is defined by the Baseline Core Specification as "a network of replicated, shared, and synchronized digital data spread across multiple sites connected by a peer-to-peer and utilizing a consensus algorithm." [FOT21b] Thus, the term can be considered synonymous with DLT.

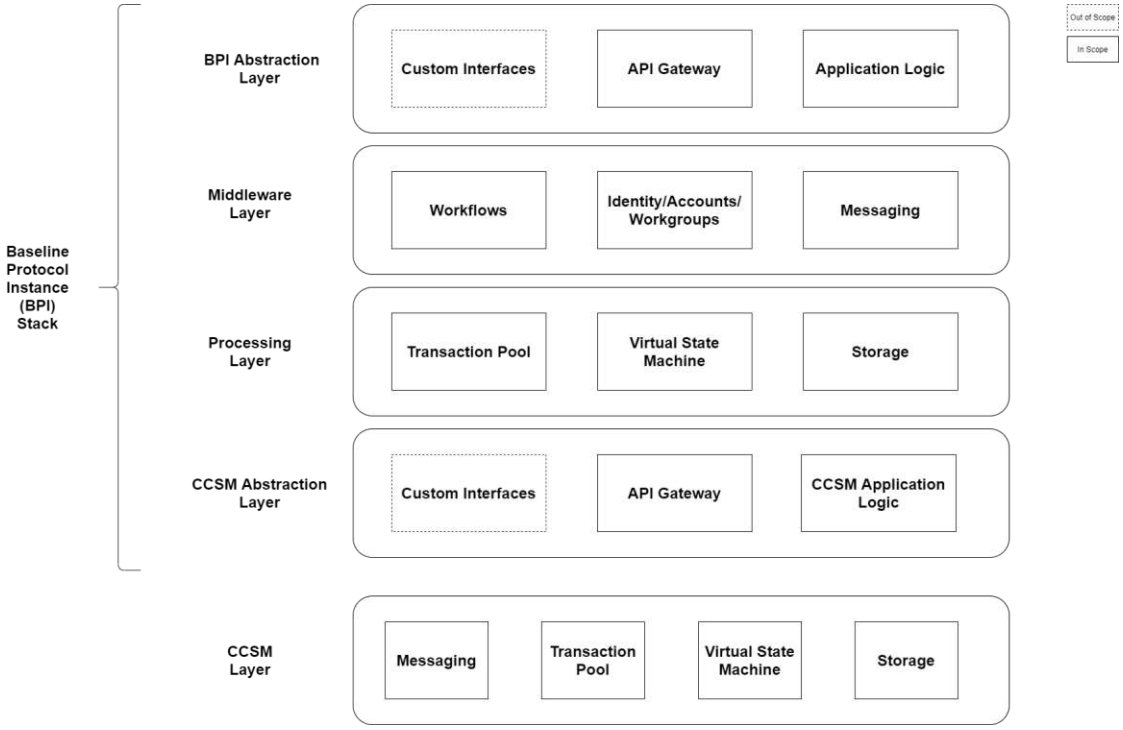


Figure 2.5: Baseline Protocol Reference Architecture [FOT21b]

BPI Subjects should communicate using secure off-chain communication technologies and how two or more BPIs can functionally interoperate with each other [FOT21b].

Processing Layer The Processing Layer further defines how a business process is represented and executed within a BPI. A *workflow* consists of a series of *worksteps*, each associated with a specific workgroup. A *workgroup* is a set of authorized users of a BPI, also called *BPI Subjects*. A *workstep* is characterized by its input, the deterministic application of a set of rules, and the deterministic generation of verifiably correct output. The output of a workstep, together with the input provided by a member of the associated workgroup, forms the input for the next workstep of the workflow. Figure 2.6 illustrates this conceptual execution of a workflow.

Technically, the workflows, worksteps, and even the data that the parties exchange (e.g., the order in the example above) are represented by *State Objects*. BPI Subjects can alter the State Objects through BPI transactions. State changes, as requested by BPI transactions, must verifiably comply with the agreed-upon business rules. This system creates a Virtual State Machine (VSM) within the BPI. The execution framework of this VSM must again be deterministic, i.e., all BPIs applying the same state change to the same object must arrive at the same result. This further requires that state transition validation computations must either complete or abort in a finite amount of time [FOT21b]. Unfortunately, it remains unclear how 'deterministic timeouts' are guaranteed by the

standard. Ethereum, as mentioned, uses *gas* to ensure that a computation either completes or aborts within a deterministic amount of computational steps.

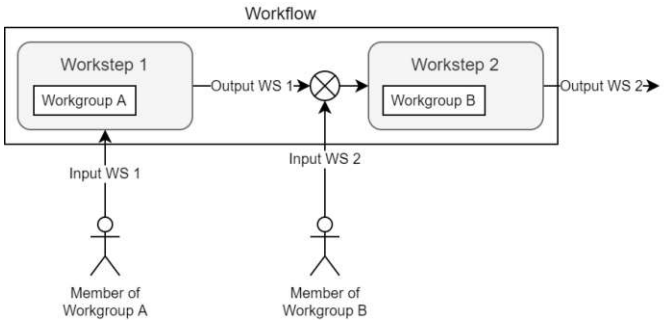


Figure 2.6: Conceptual execution of a Baseline workflow involving two worksteps and two workgroups

The **CCSM specification** [FOT21a] specifies requirements for a CCSM to be utilized in a Baseline Protocol Implementation. For example, the CCSM must support the privacy preservation of transactions and their execution, and the used consensus algorithm must have mathematical proof of security. The third and last specification, the **API specification** [TJFO21], defines a set of REST APIs that can be used by external applications to interact with the BPI.

2.2 Business Processes and Business Process Modeling

Each company has to perform a series of logically related activities to reach its business goals and provide a product. These can be a few manual steps or complex systems between multiple actors with mutual dependencies and conditions. Business Processes are an essential concept to reach these business goals effectively and efficiently by improving the collaboration of the actors [Wes19].

Weske defines a *Business Process* as "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." [Wes19, p. 5] This definition is, therefore limited to processes within a single organization. However, today, more than ever, it is no longer the case that a company works on its own. For example, it is dependent on suppliers along the value chain. From this traditional perspective, one actor must rely on the other to behave correctly. For simple processes such as the often-quoted buyer-seller example, this may be sufficient. However, that may not be enough for long-term business relationships.

Dumas et al. provide a broader definition for Business Processes: "[...], we define a business process as a collection of inter-related events, activities, and decision points that involve a number of actors and objects, which collectively lead to an outcome

that is of value to at least one customer." [DRMR18, p. 6] This definition not only focuses on the activities but also mentions events and decision points that are essential aspects of modern business processes. Reacting to events and making decisions based on changing pre-conditions is crucial for an organization to deliver improved and more tailored products to its customers [Wes19].

Business Process Management (BPM) includes concepts, methods, techniques, and tools to identify, discover, analyze, redesign, execute, and monitor business processes to optimize their performance. An essential part of BPM is the graphical representation of the business processes as Business Process Models [DRMR18, Wes19].

2.2.1 Business Process Modeling

Over the years, several different modeling notations emerged, more or less specialized to different scenarios. They can be grouped into *Process-centric Modeling* and *Data-centric Modeling*.

2.2.1.1 Process-centric Modeling

The probably best-known and most often used business process modeling language is Business Process Model and Notation (BPMN). The Object Management Group (OMG) developed it as an open standard. In BPMN, activities are represented by rounded rectangles containing the name of the activity. These activities can be connected by solid arrows representing the sequence flow, i.e., in which order the activities must be executed. Control nodes (or *gateways*) are represented by a diamond shape containing different symbols. There are gateways for forking and merging the sequence flow and expressing the dependence on some event or condition. Last but not least, events are represented by circles marked with a symbol representing the type of event, for example, a message or a time-based event. For a complete list of available symbols and their semantics, please refer to the current BPMN 2.0 specification[10] [Wes19, DRMR18]. Most tools that can generate smart contracts based on business process models use BPMN (cf. Section 2.4).

BPMN is not restricted to modeling single-organization business processes. Collaboration diagrams can be used to express the interaction between two or more business entities. Figure 2.7 shows an example of such a diagram. It shows the FMCHAIN use-case and will be used as a running example throughout the thesis. See Section 3.1.3 for a complete use-case description.

Besides activities, events, and gateways, additional elements can be used in collaboration diagrams. Most importantly, pools and swim lanes. A pool represents major process participants (or *resources*), for example, the facility management and the inspection body. A pool can have one or more swim lanes assigning activities to specific roles within an organization. It is important to note that sequence flows are only allowed within a single

[10] https://www.omg.org/spec/BPMN/2.0/

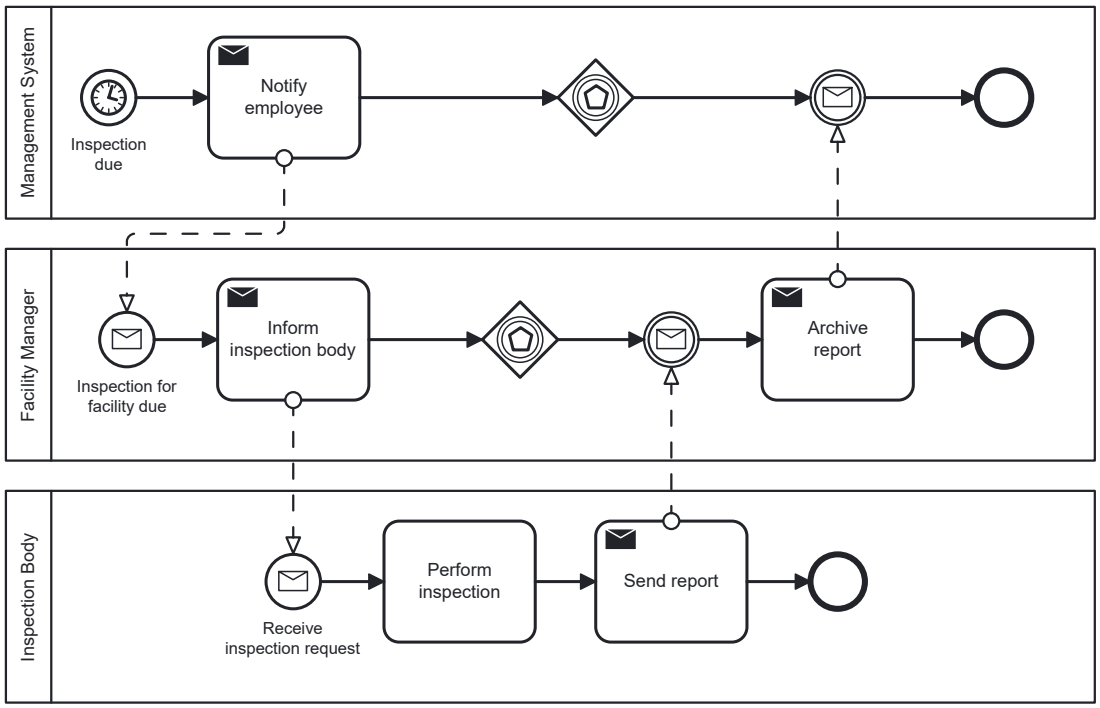


Figure 2.7: Example of a BPMN collaboration diagram (FMCHAIN use-case)

pool. The interaction between the collaborating parties is represented via a message flow, an arrow with a dashed line [Wes19, DRMR18].

Since version 2.0, BPMN also supports so-called *choreography diagrams*. The difference is that choreography diagrams do not define the internal process of all collaborating parties, such as collaboration diagrams, but focus on the exchanged information between the parties. This makes such diagrams well suited to describe inter-organizational decentralized processes where no participant ever has complete control and knowledge over the process. Instead, each involved party only receives the information required to act independently of the other parties. Choreographies can be used where it is impossible to create a centralized process control because of organizational boundaries or large settings, where a centralized Business Process Management System (BPMS) can become a bottleneck [PSHW20].

2.2.1.2 Data-centric Modeling

Besides process-centric modeling languages such as BPMN, other approaches have been developed for specific usage scenarios. The most prominent alternative is *data-centric* respectively *artifact-centric* modeling. Several variants and modeling languages emerged, such as *Business Artifacts*, *Business Objects*, and *Case Management Model and Notation (CMMN)*. All of them share the same basic idea. They focus on *artifacts* or *cases* that

represent actual or conceptual business entities. Instead of specifying a concrete sequence of activities that must be performed to finish a business process, the entities are modeled in a more declarative way by specifying their data schema (or information model) and lifecycle (lifecycle model). This approach creates a simple and robust structure for processes that might not be entirely predictable before the process starts and require the active involvement of knowledge workers making decisions and planning during the run-time [CH09, DHPV09, HBC+16].

Figure 2.8 and Figure 2.9 show a simplified illustration of the FMCHAIN workflow in an artifact-centric way. The lifecycle model describes the possible states of the involved business artifacts *inspection* and *inspection request*. Usually, some kind of finite state machine is used, where each state corresponds to a specific stage in the lifecycle of an artifact.

The information model specifies the data held by instances of their respective artifacts. Well-designed business artifacts are self-contained, meaning that all the data needed by the artifacts and for the execution of the process is present in the artifacts. This can range from simple scalar values to complex nested data structures. It should also capture the current progress of an artifact through the process. Thus, a snapshot of all artifacts can represent the runtime state of a business process [BHS09].

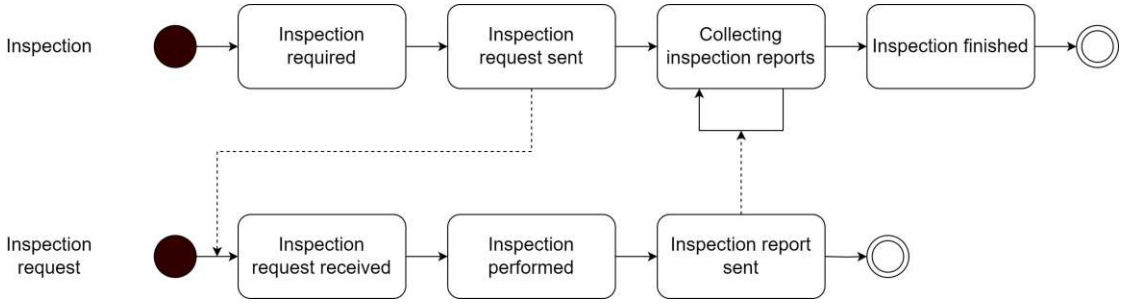


Figure 2.8: Lifecycle model of the FMCHAIN workflow (simplified)

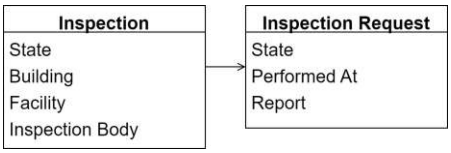


Figure 2.9: Information model of the FMCHAIN workflow (simplified)

Hull et al. [HBC+16] argue that artifact-centric process modeling is better suited to model blockchain-based collaborative business processes. The authors attribute this to several essential properties of artifact-centric modeling. Multiple scenarios in which artifact-centric modeling and methods based thereon were used in practice showed that it was possible to solve challenges other approaches could not effectively resolve. These methods proved most effective when the targeted business operations spanned across multiple business silos — which is very relevant for blockchain-based collaborations.

Further, it has been shown that artifact-centric process models can be formally verified by using well-known techniques such as LTL-FO. For example, it can be verified if a business artifact can reach a specified lifecycle state or if there are cases where this is not possible [DHPV09, HBC+16]. Still, it remains to be shown if artifact-centric process models are superior in blockchain-based inter-organizational business processes or vice versa. Maybe none of them is, and a new blockchain-aware business process modeling notation is required.

2.2.2 Achieving Compliance of Business Process Models

Compliance has received more and more attention over the last few years. Compliance ensures that the business processes are in accordance with terms that are either required by law or are agreed upon by the collaborating parties. Currently, there are two main approaches to achieving compliance in business processes [SGN07, SKGL08].

The first one is *compliance-by-detection*. It is a retrospective approach that relies on logs and reports generated by IT systems during the execution of the business process to verify given compliance requirements. The technical challenge to automate the detection is to completely and correctly log all relevant events in a machine-readable format in a secure and integer way [SGN07, SKGL08].

The second approach, *compliance-by-design*, tries to produce process models that already incorporate the required compliance rules in a way that makes non-compliant behavior technically impossible. This means compliance-by-design is a preventative approach. This mechanism needs to be implemented securely, preventing any circumvention and manipulation. Furthermore, the implementation must be correct and complete [SGN07, SKGL08, Loh13].

None of the two approaches is technically or economically feasible for arbitrarily complex compliance requirements. Therefore, it must be carefully weighed which aspects of the compliance requirements can be enforced automatically [SKGL08].

Distributed Ledger Technologies can provide benefits for both approaches. For compliance-by-detection, the events (i.e., the execution of activities) are stored as transactions on the blockchain. Thanks to the five main properties of DLTs as discussed in Section 2.1, the other participants can rely on the integrity of the recorded data. They can independently verify the execution of the process against the agreed-upon terms and conditions. Prybila et al. [PSHW20] and other commercial BPMS that provide adapters to blockchain technologies implement this approach. In contrast, for compliance-by-design, the execution of the process itself happens via transactions on the blockchain. This means the full specification of the process, including its compliance requirements, is, for example, encoded as a smart contract on the blockchain. If a participant wants to execute a process step, it sends a corresponding transaction to the blockchain system. The transaction will only be accepted if it conforms with the predefined process model. Caterpillar [LPGBD+19] can be seen as an implementation of this approach.

2.2.2.1 Discussion

Nonetheless, most of the academic work regarding business process model compliance deals with laws and other regulations, such as the often-cited Sarbanes-Oxley Act. The Sarbanes-Oxley Act is a U.S. federal law that was introduced after the well-known frauds at Enron, WorldCom, and others. It introduced requirements for financial record keeping and reporting for corporations [cond]. However, especially in trustless collaborations using choreographies, it is beneficial if the collaborators can rely on the partners to adhere to the agreed-upon terms and conditions. In a choreography, none of the partners has ever complete control over the process and data [PSHW20]. Usually, a message is exchanged if a participant has completed its sub-process and another participant has to get active. To make sure the process so far and the message itself are compliant with the predefined rules, the receiver would validate the message and the process (as far as it is visible to the receiver). Traditionally, the receiver would validate the message against the predefined rules to check the message and the process so far for compliance. If it detects any nonconformities, it will reject the message [RD05, LSNW20]. The sender could blame the receiver for falsely rejecting the message, and the receiver could blame the sender for behaving non-compliantly. Such a dispute resolution can cause delays and additional costs for all parties [Hul17]. With a system that inherently guarantees the conformance of all participants to the laws, regulations, and master service agreements, such situations belong to the past. The rule engine of this thesis aims to close this gap.

2.3 DLTs for Collaborative Business Processes

BPM is a well-known strategy for organizations to improve productivity, competitiveness, and efficiency. Traditionally, these processes are executed centrally for a single company. Although most BPMS allow the assignment of specific tasks to external actors, a centralized approach is inefficient when multiple entities have to be involved in the execution of a single process [GGSGL+20].

Unfortunately, inter-organizational business process management is far more challenging than the intra-organizational case. Issues such as who is responsible for the process execution or who hosts the database arise. The participant hosting the infrastructure obviously has advantages over the other collaborators. For example, they could modify the data to their favor without the notice of the other participants [SSSJ19]. In modern business, it is often the case that corporations cooperating in some areas compete in other areas[11]. If such parties come together to achieve a joint business goal, the entity executing the joint business process must be neutral. An often-used solution for this problem is to select a non-involved third-party responsible for hosting the required services and serving as a mediator in the case of disputes [WXR+16]. However, finding a suitable body that all participants trust can be difficult. This can result in a different mediator being used for each collaboration and also results in additional effort and costs. Additionally, in

[11] This is a so-called *coopetition*.

large inter-organizational settings such as choreographies involving many participants, a central BPMS can become a bottleneck [PSHW20].

Another characteristic of collaborative business processes is that they consist of sub-processes executed by different entities. The integration of these processes requires extensive exchange of information. This leads to data redundancy, and no party has ever full knowledge about the process, i.e., how, when, and where tasks have been conducted [CCD+19]. Further, the other collaborators have no influence on the execution of the sub-process. This uncertainty creates a need for trust in the correct and compliant execution of the process. Such uncertainties can lead to so-called 'process vulnerabilities', as defined by Müller et al. [MOR20]. Vulnerabilities describe the repercussions, e.g., costs or delays, that occur when parts of the process or the process as a whole are not executed as intended. To increase the trustworthiness of a process, one can either reduce uncertainties or build confidence. The objective of reducing uncertainties is twofold: it seeks to lower the likelihood of undesired events occurring by taking proactive measures or to mitigate the consequences when such events occur through reactive measures. Building confidence aims to enhance the perceived reliability of the process without modifying the process itself.

We want to pick up the FMCHAIN use case as our running example again to exemplify uncertainties and trust issues that can arise in collaborative business processes. Even in this small business process, there is room for trust issues. Certain building facilities such as escalators or elevators are required by law to be inspected in a given interval. A governmental body responsible for verifying the correctly performed inspections could doubt that inspection happened within the prescribed interval. The facility management company could argue that they informed the inspection body early enough, but the inspection was not performed promptly.

Distributed Ledger Technologies are seen as a well-suited technological solution to execute and manage collaborative business processes to solve the issues mentioned above [GGSGL+20, WXR+16]. Müller et al. [MOR20] argue that DLTs can reduce uncertainties and build confidence in several ways. The authors identify six different 'blockchain-based trust patterns'. These are technical and functional properties of blockchains, i.e., a transparent event log or tamper resistance, that can reduce uncertainty and build confidence when applied to different business process components.

As stated by Hull [Hul17], dispute resolution is a substantial expense in today's business world. Whether it is a dispute about a product purchase, goods moving through a supply chain, or a loan setup, these disagreements often take weeks to resolve. This ties up a considerable amount of money that could be used elsewhere. DLTs act as a single source of truth and have the potential to speed up dispute resolution significantly. With the use of DLTs to execute the business process, the process state is shared and distributed to every node of the network. If all state-changing messages are recorded on the blockchain, the observability of the process is ensured. This allows the other participants to monitor the process transitions. Other stakeholders, such as external auditors, can inspect past

executions for compliance [CCD+19, MWA+18]. DLTs provide a trustworthy environment without needing a trusted third party for hosting or mediation [WXR+16].

Think about the trust issues described above for our FMCHAIN use-case running example. Thanks to the immutable audit trail provided by DLTs, it is easy to find out if all inspections happened as required and when the facility management company informed the inspection body about the upcoming inspection.

Nevertheless, the integration of DLTs into Business Process Management and Execution is in an early stage, and multiple challenges remain left to be addressed [GGSGL+20]. Besides the general drawbacks of DLTs (cf. Section 2.1.4), some challenges arise from the requirements specific to collaborative business processes. Due to its unique characteristics, the existing standardized modeling languages do not provide adequate support for representing the various interactions between an organization and a blockchain. For example, handling probabilistic finality is challenging when mapping traditional business process models to DLTs. When a transaction gets orphaned (i.e., not included in the longest chain of the blockchain), the transaction must be created again. Other parties must again wait for a specific number of blocks to be added after the block of interest to the blockchain to be sure the required change to the process, in fact, happened [FHBL19].

Because of the low transaction volume compared to traditional systems and probabilistic finality, some authors argue that DLTs are currently only suited for processes in which some seconds of latency are not an issue [WXR+16, MWA+18, Men18]. Other challenges include how the usability of blockchain-based systems can be improved or how business processes could be redesigned to leverage the full potential of DLTs [MWA+18].

2.4 State of the Art

Since this thesis touches multiple research areas, we did not perform a structured literature review but a narrative literature review [Rho11]. Nevertheless, we want our approach to be as transparent as possible. The following steps were conducted during the literature review:

1. Perform a search on *ResearchGate* and *Google Scholar* with the following search phrases:
   - compliance by design
   - compliance by design blockchain
   - blockchain business process execution
   - blockchain collaborative business process
   - blockchain inter-organizational business processes
   - blockchain rule engine

2. From the previous step's results, those papers were excluded that were either too old (e.g., a publication year of 2012 or before for papers on blockchain topics), had a relatively small number of citations (< 15), or could not contribute any relevant concepts to the subject of this thesis.

3. Starting from the remaining papers, a snowballing approach was conducted, i.e., examining the reference list of a paper to find new papers to include or find papers that reference the paper at hand. We again started at step 2 for the newly found papers.

2.4.1 Compliance-by-Design

Goedertier et al. [GV06a] recognize the lack of attention paid to avoid hard-coding policies and regulations into business process models. They argue that business rule modeling is essential to achieving flexibility and reusability of business process models. The authors further show how the four types of business rules as defined by Wagner [Wag02] — integrity constraints, derivation rules, deontic assignments, and reaction rules — can be used to create less complex business process models. Unfortunately, the paper neither addresses the implementation of such a system nor proposes a language for specifying such rules. Rosenberg et al. [RD05] propose a system integrating business rules into BPEL engines. They define a Rule Interceptor Service that validates the business rules *before* or *after* the execution of a BPEL activity.

Also, Hull [Hul17] emphasizes using business rules, decision tables, and Decision Model and Notation (DMN) to tackle the need for change in collaborative business processes. The processes (and their blockchain-based implementation) need to be adapted as change occurs in several dimensions, such as country policies, different product types, and diverse approaches to transport and finance. The mentioned techniques can simplify change management and lower the effort.

Sadiq et al. [SGN07] conceptualize a Formal Contract Language (FCL) to ensure compliance-by-design of process-centric business process models. Process annotations are used to enrich process models with control objectives. These are formal models of the corresponding compliance requirements and represent the obligations, permissions, and prohibitions a company has to fulfill to be deemed compliant. An example of such a control objective in the domain of facility management could be: *The time since the last inspection of an elevator must not exceed one year.* These objectives are then encoded as expressions in FCL and attached to the process model to facilitate a better understanding of the interaction between the process model and the control model. The FCL encodings can be converted into RuleML for automatic processing during the execution of the process.

Lohmann [Loh13] chooses a different approach for reaching compliance-by-design and focuses on artifact-centric business process models. As mentioned in Section 2.2.1.2, such process models follow a more declarative approach. The author uses Petri nets to formalize the business processes and compliance rules because of their strict mathematical

foundation. A compliant business process model is synthesized by the composition of the Petri nets of the business process and the compliance rules. As mentioned later, Caterpillar, the model-driven approach by López-Pintado et al., can be seen as a blockchain-based approach for achieving compliance-by-design.

2.4.2 Blockchain-Based Business Process Engines

In [SSSJ18], Sturm et al. propose a relatively simple workflow execution engine that is based on smart contracts executed on the Ethereum blockchain. In [SSSJ19], their work is extended by resource awareness, i.e., the capability to define which collaborator is allowed to execute a particular task. Their prototype supports direct allocation, role-based allocation, and — to some degree — the separation of duties patterns defined by Russel et al. [RAHE05]. Unfortunately, exclusive gateways, parallel gateways, and basic task activities are the only supported BPMN elements.

Caterpillar, a blockchain-based process execution engine by López-Pintado et al. [LPGBD+19], is more advanced regarding supported BPMN elements. The system uses a specifically crafted compiler, transforming the BPMN process model into Solidity smart contracts deployed on the Ethereum blockchain. Each step of the business process is triggered via transactions of the participants and executed on the blockchain. The authors argue that their approach ensures compliance-by-design, because the smart contracts can check the process's current state and the transaction's input data. Hence, no party can change the state in a non-compliant way. A disadvantage of this approach is that input data to tasks and data which is required to evaluate conditions of decision gateways have to be stored on-chain. This means that a collaborator is forced to store potentially sensitive data on the Ethereum blockchain, where it is accessible to all other participants of the network.

Their initial proposal has been refined and extended several times. In [LPDGBW19a], the authors tackled the problem of missing access control and introduced a dynamic role binding for Caterpillar. López-Pintado et al. propose a policy specification language that defines how an actor can be nominated or released to or from a role while a process is already in progress. The authors extend the language in [LPDGBW22] to enable the deferred assignment of sub-processes and the dynamic selection of execution paths within a process model as it executes, i.e., allowing the actors to collaboratively guide the execution of a process instance in response to their specific needs. A further improvement is an interpreted execution of the workflow models [LPDGBW19b]. The models are no longer transformed into hard-coded smart contracts but are stored in a space-optimized representation on the blockchain. The benefits of this approach are reduced costs because the interpreter only has to be deployed once and increased flexibility as the participants can apply changes to the model and already running instances.

A system called *Lorikeet* by Tran et al. [TLW18] is quite similar to Caterpillar. Like Caterpillar, it is a model-driven engineering approach that can generate smart contracts implementing a specific business process out of a BPMN process model. The difference is that the authors recognized the need for a data store that the process participants can

trust. Theses so-called *registries* are smart contracts that store digital twins of real-world assets such as land titles. *Regerator*, the result of an earlier work of the same authors, is used to generate the smart contracts for the registries [TXW+17]. BPMN is extended to allow the business process interaction with the registry contracts. This allows the rather easy creation of blockchain-based business processes for managing assets.
Although the authors only propose using the registries to represent real-world assets, using them for arbitrary data objects is conceivable. As Lorikeet supports transferring the ownership of these assets, the functionality could be used to represent the handover of the data object to another participant during the execution of a collaborative business process. Only the particular participant would then be allowed to modify the data object. Unfortunately, this consideration is only feasible for small data objects, as storing data is among the most expensive operations on most blockchains.

Another work in that direction is done by Madsen et al. [MGH+18]. However, instead of using BPMN to model the workflows, the authors use DCR graphs. Like BPMN, DCR graphs can be seen as a process-oriented way of specifying the business process but have a more formal mathematical foundation.

In contrast to creating business process execution engines specifically crafted for Distributed Ledger Technologies, another approach is to make existing execution engines compatible with blockchain technologies. Falazi et al. [FHBL19] propose such a system. The authors created an extension for BPMN called *BlockME* (Blockchain-aware Modeling and Execution). BlockME defines four new task and event types, specifically designed to cope with the peculiarities of blockchain-based systems: *SubmitTransactionTask*, *ReceiveTransactionTask*, *EnsureTransactionStateTask*, and *OrphanedTransactionEvent*. The interesting part of the work is that the authors explicitly consider the probabilistic finality of Proof-of-Work blockchains. SubmitTransactionTask and ReceiveTransactionTask have an attribute *WaitUntil* specifying a minimum number of block confirmations to consider the transaction final. The OrphanedTransactionEvent can be used to react if a transaction gets orphaned, i.e., the transaction is part of a branch that gets replaced by a longer one. They also provide transformations for these new types into standard-compliant BPMN 2.0 constructs. The benefit of this approach is that it can be used with all already existing, well-established process execution engines supporting BPMN 2.0. Sometimes, these BPMS already support the interaction with DLTs on their own. For example, Bonita BPM allows interacting with the Chain Core blockchain by using connectors[12].
The problem with these solutions is that they use only some of the benefits blockchain systems would provide. For example, all of them are still centralized, i.e., all participants have to trust the provider of the Business Process Execution Engine.

2.4.2.1 Discussion

The presented works all support compliance-by-detection or compliance-by-design to some degree. While this can improve dispute resolution at the process level, they completely

[12] https://github.com/Bonitasoft-Community/bonita-connector-chain

ignore the data perspective. For the collaboration using a decentralized execution engine to be successful, all participants must have the same understanding of the exchanged data. This reveals the main drawback of using pure BPMN as the source for a model-driven approach. BPMN currently does not provide a data perspective of processes or choreographies. Thus, it neither supports the definition of formal rules on the data that gets exchanged by the process participants. If such checks are required, they must be manually integrated into the generated smart contracts' source code, which causes additional software engineering effort and is not a trivial task.

Meyer et al. [MPB+13, MPB+14] identified the need for a shared data perspective in process choreographies. The authors propose to define a global data model that contains all data objects to be exchanged by the participants. The message type is specified by referring to one of the data objects defined in the global data model.
We want to take this idea further and enable the definition of (business) rules for the messages encoding the agreed-upon terms and conditions of the collaboration. The goal is that all collaborators can rely on the correct execution of the business process, compliant with all of the defined rules, to make dispute resolution easier or unnecessary in the best case.

Lichtenstein et al. [LSNW20] does an interesting work that also goes in that direction. The authors propose an approach for data-driven process choreography implementation on blockchains. The choreography models are represented by data objects that are stored on the blockchain. The messages, as prescribed by the choreography model, manipulate the state of the respective data objects. This means that the state of the objects implicitly captures the current state of the execution. However, the more interesting part of this work is that the authors also recognized the need to define conditions for the choreography execution to prevent incomplete state changes. They propose a *design by contract* approach to reach that goal. The tasks in the choreography model are annotated by small code blocks encoding post-conditions that must be fulfilled after the respective state change was applied.
This basic idea of this concept has much in common with our intended rule engine. The main differences are that all the state is still stored on the blockchain (with all the disadvantages we already have outlined) and that it is required to write small pieces of code in Solidity, for example. We want to provide a simple expression language to define the conditions for execution. We expect that this approach is less error-prone due to lower complexity and that persons without programming knowledge are able to write and understand the conditions.

2.4.3 Others

Prybila et al. [PSHW20] propose a mechanism for runtime verification of business processes by utilizing the Bitcoin blockchain. The authors define verification as "evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition" [PSHW20, p. 817]. Their work focuses on choreographies, where no collaborator ever has full control and knowledge of the process, and the

process owner can hand over the process control to other participants. This handing over is conceptualized via a token that is sent as a Bitcoin transaction to the blockchain. The token contains metadata about the process, e.g., to identify the process instance the token belongs to or the currently executed activity. To enable parallelism, the token can be split and re-joined. The chain of transactions related to a token provides proof of the valid or invalid process execution. In the case of a violation, it is possible to identify the culprit unambiguously.
Handing over the token from one participant to the next requires a series of off-chain messages between them. The result of the communication is a Bitcoin transaction that is signed by both the sender and receiver of the token. The signature of the receiver is required to ensure non-reputability. As already mentioned in Section 2.2.2, this work only allows compliance-by-detection on the execution of the process. It does not consider the data that presumably has to be exchanged during the handover of the process control. Again, the receiver must manually validate the received data against the agreed-upon master services agreement. Our intended solution to define rules on this data could further increase the trust in such a system.

An interesting work regarding the possible architecture of our intended rule engine is [ACC+18] by Astigarraga et al. The authors combine Hyperledger Fabric and a library called *nanoRETE* to implement a blockchain-based rules framework. Hyperledger Fabric is a permissioned DLT that allows smart contracts to be implemented with JavaScript. nanoRETE is a prototypical implementation of the Rete algorithm using JavaScript. This allows the evaluation of Event Condition Action (ECA) rules via smart contracts on Hyperledger Fabric.
Although an interesting approach, using a private permissioned blockchain has its downsides. Unfortunately, the concept can not be easily applied to public blockchains. The authors use Hyperledger Fabric's ability to use CouchDB as a state database. Therewith, it is possible to store even large amounts of data and access it via the smart contracts implementing the rule engine. This would also be possible in a public blockchain, but the costs for doing so are very high, and the data would be visible to all blockchain nodes.

The rule engine can also be seen as a part of the state transition function of a state machine replication system. In such systems, a state machine is replicated among several nodes. Requests are sent to and executed by all replicas of the state machine. If all replicas are executing the requests in the same order and the execution itself is deterministic, all non-faulty nodes will eventually have a consistent state. Much research on state machine replication has been done in the last decades. Although, the main focus is on consensus and the ordering of the inputs. A deterministic implementation of the state transition function is left out [Lam78a, Sch90, BSA14, Lam78b].

Cryptocurrencies such as Bitcoin and Ethereum can also be considered as an implementation of a state machine replication system. Both provide a protocol that defines whether a given state transition is valid. While the capabilities of Bitcoin are restricted by a simple non-Turing-complete scripting language, Ethereum allows arbitrary computation to be

executed as part of the state transition. Despite the Turing-completeness of Ethereum's virtual execution environment, determinism is guaranteed via a concept called 'gas'. This ensures that an execution succeeds or fails within a finite amount of steps. Existing programming languages for smart contracts, and the virtual machines they are based on, provide a good source for some requirements of the system we are planning to create. Nevertheless, with some restrictions, they can be seen as fully-fledged programming languages and thus are too complex so that a person without a software engineering background could write compliance rules with them [Nak09, But20, Woo22].

CHAPTER 3

Requirements and Model

3.1 Use Cases

In the following section, we present three use cases of inter-organizational business processes that are used to derive functional requirements for the rule engine, as shown in this thesis. We also extend the process models of the processes with compliance and integrity constraints to demonstrate the utility of the proposed rule engine. Often, such constraints are implicit knowledge of domain experts and not explicitly expressed in the process models. In Section 5.1, the use cases and the compliance rules are used to evaluate the proposed concept.

3.1.1 Use Case 1: DiCYCLE

This use case is borrowed from DiCYCLE[1], which is an interdisciplinary research project between the Institute of Building and Industrial Construction and the Institute of Computer Engineering at TU Wien. The project deals with processes happening at the end-of-life phase of buildings and how these processes can be improved by digitalization using Building Information Modeling (BIM) and Distributed Ledger Technologies. One of the processes that were specified together with domain experts aims to improve the traceability of Construction and Demolition Waste (CDW) (for example, concrete or waste steel). CDW is the largest waste stream in the EU. Therefore, a recovery target of 70% was set by the 2008 Waste Framework Directive [CDW]. This means that at least 70% of the non-hazardous wastes must be either recycled, re-used (e.g., using entire parts for a new building) or backfilled[2]. For the owners of the building that is going to

[1] https://www.industriebau.tuwien.ac.at/forschung/forschungsprojekte/dicycle/

[2] EU Commission Decision 2011/753/EU defines backfilling as "a recovery operation where suitable waste is used for reclamation purposes in excavated areas or for engineering purposes in landscaping and where the waste is a substitute for non-waste materials."

be demolished, this means additional costs. They must find suitable recycling plants for the individual CDWs, and usually, these plants must get paid for their service. Thus, it might be beneficial for the building owners to ship the waste to countries with less regulated environmental laws (environmental dumping) or dump the waste somewhere without permission. To avoid that, the legislator prescribes traceability of the wastes, from demolition to recovery.

The process we are describing focuses on the recycling of such wastes. When a building is going to be demolished, the expected amounts of CDWs are determined. For each recoverable CDW, a suitable recycling contractor has to be found. Once that is finished, the demolition can begin. The demolition company is responsible for separating the waste as required. Usually, this happens by creating individual heaps for the different materials. Eventually, when enough material is collected, it will be transported to the recycling contractor. The recycling contractor must countersign the origin of the CDW, the type, the amount, and that it is not contaminated with other wastes.

When everything of a particular CDW was transported to the recycling contractor, the sum of the approved amounts must match (with some deviation) the initially determined expected amount. If this is not the case, the general contractor or the building owner must create a deviation report to explain the deviation. Once the demolition is finished and all CDWs are transported to their respective recycling contractor, the process is completed. Please note that the described process is highly simplified for better understanding.

We have modeled this process using artifact-centric business process modeling to show the utility of our proposed concept for this business process modeling approach. Figure 3.2 depicts the information model of the process, and Figure 3.1 shows the lifecycle model.

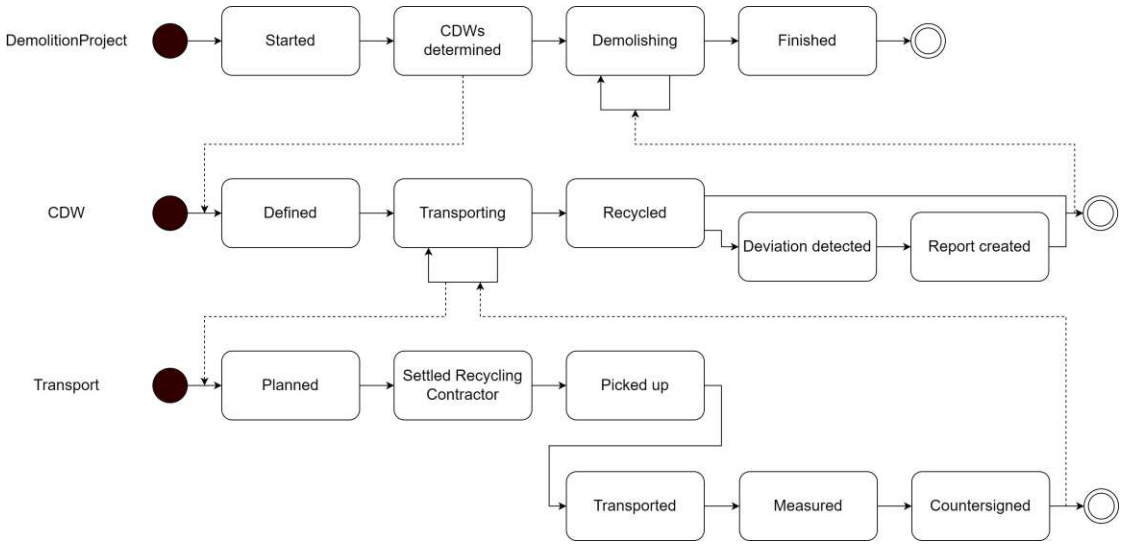


Figure 3.1: Lifecycle model of the DiCYCLE use case

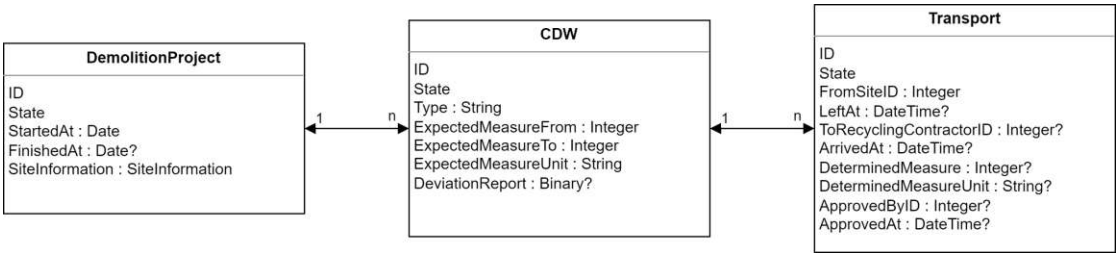


Figure 3.2: Information model of the DiCYCLE use case

The process must follow some rules that are not entirely expressed in the process model. Every rule is identified via a unique identifier to enable referencing them later in the thesis.

UC1-R1: The sum of the measured amounts of the countersigned transports for a CDW to the recycling contractor must be in the expected range.
This rule is at least partially represented in the process model by the two states *Deviation detected* and *Report created*. If the sum is within the expected range, the lifecycle of the CDW is finished. Otherwise, a deviation report must be created first.

UC1-R2: The recycling contractor must be entitled to the treatment of a specific CDW. Usually, a governmental agency[3] keeps a record of those recycling companies. Also, the recycling contractor must be settled before the actual transport leaves the demolition site.

UC1-R3: Only an employee of the specified recycling contractor can countersign the CDW transport.

UC1-R4: All CDWs must be in their final state before the demolition project can be finished.

UC1-R5: The measurements at the recycling contractor must be recorded in the same unit as was used for the expected amounts.

UC1-R6: *ExpectedMeasureFrom* and *DeterminedMeasure* must be greater than 0. *ExpectedMeasureTo* must be greater than *ExpectedMeasureFrom*.

UC1-AC: A disadvantage of artifact-centric modeling is the lack of ability to express the responsibilities of the different actors, i.e., which state transition of the lifecycle can be performed by which actor. Usually, this is part of a separate specification. In our simplified scenario, we assume that most of the transitions are executed by the general contractor of the demolition project. Nevertheless, some transitions must be accomplished by specific other actors. For example, only an entitled recycling contractor is allowed to countersign the delivery of a CDW transport to the recycling plant. A

[3]In Austria, for example, the Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation, and Technology.

transport company usually executes the other transitions regarding the transport of a CDW. The general contractor or the building owner must create the deviation report.

3.1.2 Use Case 2: Supply Chain Scenario

This business process is often used (in adapted form) in related literature such as Fdhila et al. [FRMKR15] and Weber et al. [WXR+16]. We use it as we expect the reader to be familiar with it. It is easier to classify the concepts of this thesis in the context of this relatively well-known scenario.

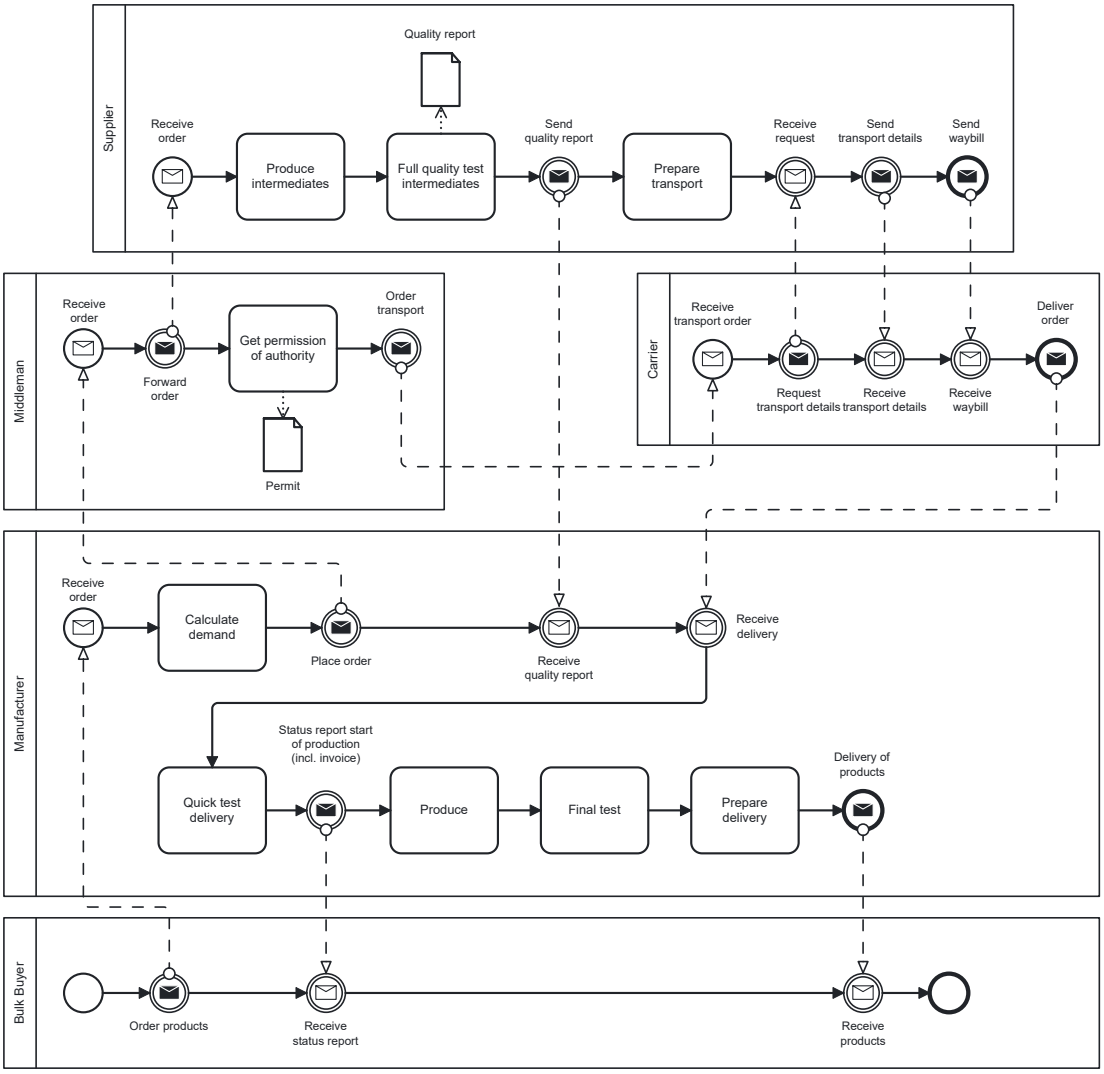


Figure 3.3: BPMN diagram of the supply chain use case (adapted from [FRMKR15])

Figure 3.3 shows the BPMN diagram of the described process. This collaborative business process involves five actors and describes the manufacturing and delivery of product

orders. First, a bulk buyer places an order at the manufacturer. The manufacturer receives the order and calculates its demand for intermediate products. The charge for the intermediates is sent to a middleman, which forwards it to a supplier. The supplier starts with production. In the meantime, the middleman also orders transport. We assume a special transport, e.g., an overlong or extra-wide heavy vehicle, is necessary. Thus, the middleman must also request permission from the relevant authorities. When the intermediates are finished, they are picked up by the carrier and delivered to the manufacturer. The manufacturer can then start producing the main products and deliver them to the buyer.

Processes such as the described one leave room for uncertainties and trust issues. Hence, the bulk buyer and the manufacturer have signed a master service agreement that governs their transactions. The terms of this MSA can again be seen as rules for our intended rule engine:

UC2-R1: If the buyer orders at least one thousand pieces of a particular product, the manufacturer will grant them a discount of 2%.

UC2-R2: The manufacturer must deliver the products within 30 days of reception of the order.
This term implies when the manufacturer may receive the intermediate products from the supplier at the latest. To meet the deadline, it must receive the intermediates at most 20 days after the customer places the order.

UC2-R3: The intermediate products must meet certain quality criteria. Therefore, the supplier must perform a full quality test of the intermediates. The results of these tests are sent to the manufacturer.

UC2-R4: After the intermediates have been delivered, the manufacturer performs a quick quality check and compares the results to the quality report received from the supplier.
We assume the two companies are in coopetition. Hence, the bulk buyer must not know the suppliers of the manufacturer. Because of this, the manufacturer can not simply forward the quality report of the supplier to the buyer. Still, the buyer wants to be sure that the quality requirements of the intermediates are met.

UC2-R5: The manufacturer performs a final test for each product.

3.1.3 Use Case 3: FMCHAIN

This use case is borrowed from FMCHAIN[4], which was a research project at the Institute of Computer Engineering at TU Wien. The project's scope was to investigate how inspection and maintenance processes in the operational phase of a building could be improved by digitalization employing DLTs and Building Information Modeling (BIM) data.

[4] https://www.auto.tuwien.ac.at/index.php/projectsites/354-fmchain

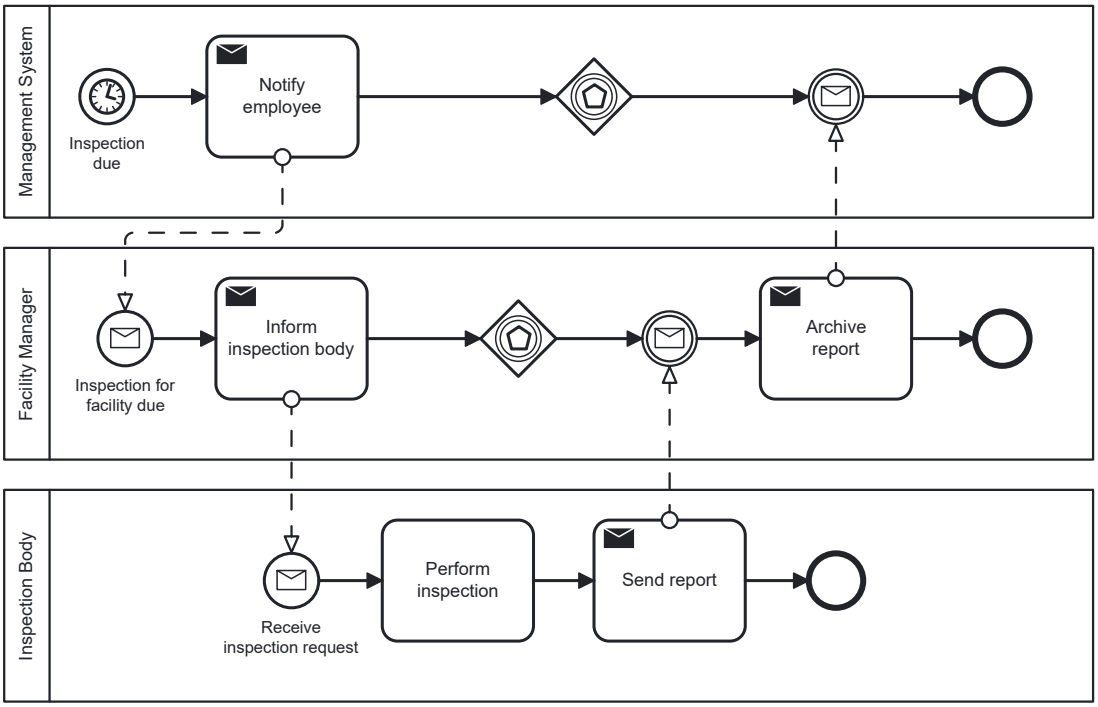


Figure 3.4: BPMN diagram of the FMCHAIN use case

Figure 3.4 shows the BPMN diagram of this process. Certain facilities of buildings, such as escalators or elevators, must undergo an inspection at prescribed intervals. Only accredited inspection bodies are allowed to perform this inspection. The shown process models this scenario in a simplified way[5].

First, an integrated management system notifies an employee of the facility management company of the upcoming inspection. The employee selects a suitable inspection body and sends them a request containing information about the building and the specific facility to be inspected. The inspection body performs the inspections and confirms the operating safety by issuing an inspection report. The process is finished after the facility management archives the report in the management system.

We identified the following rules to be relevant for our rule engine:

UC3-R1: The inspection must be conducted before a specific deadline is reached.
The legislator prescribes an inspection interval for facilities such as elevators or escalators. The inspection must be performed within this interval to ensure that the facility can remain in operation.

UC3-R2: The inspection body must provably issue a compliant report.

[5]The process reflects the regulations of the Austrian *Hebeanlagen-Betriebsverordnung 2009* and may differ for other countries.

The legislator also defines that after the inspection is finished, the inspection body must issue an inspection report containing some predefined elements.

UC3-R3: Only accredited inspection bodies are allowed to conduct an inspection. Those inspection bodies must be included in a publicly accessible register.

3.2 Requirements

Based on the literature review of Section 2.4 and the use cases as defined before, we now define the requirements the rule engine has to meet. This answers the research question *RQ1*.

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in the upcoming requirements are to be interpreted as described in *RFC2119*[6] when, and only when, they appear in all capitals, as shown here.

3.2.1 Non-functional Requirements

R1 The rule evaluation MUST not block other nodes. It MUST either succeed or fail within a finite time.
This means that, for example, infinite loops are not allowed in the definition of rules. Essentially, this requirement prevents Denial of Service attacks on a node of the system via a specifically crafted rule.
This requirement results from the system's replicated state machine character (as mentioned in Section 2.4). It must be guaranteed that all non-faulty replicas eventually end up in the same state if the state transitions are executed in the same order [Sch90, BSA14].

R2 The rule evaluation MUST be deterministic.
As the requirement before, this requirement is implied by the state machine replication approach of the system. As such, the rule engine is part of this system's deterministic state transition function. When applied to the same input parameters, the rule evaluation must produce the same result on every non-faulty node [Sch90, BSA14].

R3 The rule evaluation MUST be verifiable.
The participants of the decentralized business process MUST be able to verify whether the process was executed compliant with the predefined rules. An external stakeholder (such as an auditor or governmental body) MUST also be able to verify whether the process has been executed compliant with the relevant laws and regulations [FKRMR22].

R4 The rule evaluation MUST be sound for reporting the correctness of the current state of the system.

[6] https://www.rfc-editor.org/rfc/rfc2119

The evaluation logic is always correct when it claims a rule to be fulfilled. This is an essential requirement for the rule engine. It means that only if all of the conditions described by a rule are met the rule is considered to be fulfilled. If this were not the case, none of the participants could trust the system, reducing its utility ad absurdum.

R5 The rule evaluation SHOULD be complete for reporting the correctness of the current state of the system.
Additionally to the consequence of the requirement before, the evaluation logic is always correct when it claims a rule to be not fulfilled. This requirement is intentionally considered optional as its absence is not as fatal as the requirement before. A system that is sound but not complete could provide a mechanism (e.g., all participants agree on the correct execution of the process) to override the rule engine for cases in which the evaluation of the rule engine comes to the wrong result.

R6 The rule engine MUST use an expression language with well-defined syntax and semantics.
For example, natural language is not suited to be used. It is rarely unequivocal and leaves room for different interpretations.

R7 The expression language SHOULD be easy to use, also for people without programming skills. It MUST NOT be required to write program code in any programming language such as Solidity.
As mentioned in Chapter 2 writing code for a smart contract is not a trivial task and can lead to bugs that can be exploited by a malicious actor. A domain-specific expression language provides a reduced instruction set and is therefore easier to use than a fully functional programming language.

R8 The rule engine SHOULD use an expression language already used in business process modeling.
Using a language that is already established in business process modeling has the advantage that the training time for new users is shorter than if you have to learn a completely new language.

3.2.2 Functional Requirements

3.2.2.1 General

R9 The rule engine MUST allow the definition of constraints. Constraints in our context are expressions that evaluate to a boolean value.

R10 The rule engine MUST allow the definition of global integrity constraints. These are constraints that must be fulfilled in every state of the process.
An example of such global integrity constraints is rule *UC1-R6*. For the whole lifetime of a CDW, *ExpectedMeasureTo* must be greater than *ExpectedMeasureFrom*.

R10.1 The rule engine MUST allow the definition of global integrity constraints as defined in requirement *R10* on the global shared process state. The global shared process state contains all the data that is needed to digitally reflect the process execution of a particular process instance until now and is shared with all participants of the system. For example, the data described by the DiCYCLE use-case information model (Figure 3.2) can already be considered the global state of a process instance of that business process.

R11 The rule engine MUST allow the definition of state transition constraints. These constraints must only be fulfilled when a specific state transition is executed. In other words, the state transition may only be executed if the constraint is fulfilled.
An example of such constraints is rule *UC2-R5*. Before proceeding to the state *Prepare delivery*, every product must undergo a final test.

R11.1 The rule engine MUST allow the definition of state transition constraints as defined by requirement *R11* on the metadata of a state transition. This metadata MUST at least include the following:

- An identifier of the participant who performed the state transition.
- A list of user identifiers who signed the state transition.
- A timestamp when the state transition has been performed.

The message sender and the list of users who signed the state transition are required for specifying resource allocation. For example, the artifact-centric approach of the DiCYCLE use case does not explicitly specify which actor can execute which state transition. The rule engine can solve this issue by specifying constraints on the user information of the metadata. The timestamp is, for example, required by constraint *UC3-R1* to verify if the inspection was conducted before the specified deadline.

R11.2 The rule engine MUST allow the definition of state transition constraints as defined by *R11* on the state transition data of a state transition. We define the state transition data as all the data that needs to be atomically changed to transition the process instance from a state σ_t to the next state σ_{t+1}.
If a business collaboration is realized as choreography, there will not be any shared process state. All the information required to continue the process is transported via the messages (i.e., state transition data) between the participants [LSNW20]. Hence, we can not define constraints on the shared process state but only on the state transition data.

3.2.2.2 Data Types, Control Structures, and Operations

This section specifies the basic functionality of the rule language. The requirements were derived from the use cases presented before and inspired by existing programming languages. The aim is to provide enough functionality to represent the constraints from the use cases and keep the language as simple as possible.

R12 The expression language of the rule engine MUST at least support the following simple data types: numeric values, strings, dates (and time), boolean values, and a *null* value (i.e., representing a missing value).

R13 The expression language MUST support the following complex data types:

- Objects: An object maps property names to values. A value may be of a simple data type as defined in requirement *R12* or again of a complex data type as defined here.
- Arrays: An array is a collection of values of the same data type. A value can again be a simple or a complex data type.

R14 The expression language MUST support the usage of arithmetic expressions, boolean expressions (such as comparisons), and conjunction and disjunction of expressions within constraints.

R15 The expression language MUST support simple control structures such as if-conditions and loops over arrays.

R16 The expression language MUST be extensible. It MUST support user-defined functions and external functions.
The rule engine, as defined in this thesis, only provides a basic set of functionality. As the requirements may vary significantly from use-case to use-case, the rule engine needs to be extendable in an easy way to support further scenarios.

3.2.2.3 Resource Allocation

Resource allocation means which person is allowed to perform a process step in a business process. There are different patterns in how these resources can be allocated. We use the definitions of Weske [Wes19] in the upcoming requirements.
From a software design perspective, the resource allocation could be directly a part of the Business Process Engine or could be *outsourced* to the rule engine. As discussed in Section 3.3.4, we have decided on the latter option.

R17 The rule engine MUST support the direct allocation of resources.
Direct allocation (or static allocation) means that the specific person allowed to perform a particular process step is already defined when the process starts.

R18 The rule engine MUST support the deferred allocation of resources.
In contrast to direct allocation, for deferred allocation (or dynamic allocation), the individual capable of performing a given process step is determined during the runtime of the business process.

R19 The rule engine MUST support role-based allocation.
Role-based allocation does not define a specific individual allowed to perform an activity but a role. All individuals with this assigned role are treated as functionally equivalent and allowed to perform the activity.

R20 The rule engine SHOULD support *Separation of Duties*.
An example of separation of duties is a document that two persons must countersign. This pattern MAY also be generalized to an m-out-of-n scheme, meaning that at least m of possible n persons have to countersign the document.

3.3 Model

In the following sections, we thoroughly introduce the model of our proposed rule engine. We show how it can be integrated into an existing blockchain-based BPMS and discuss design decisions that happened along the way.

3.3.1 Integration with a Blockchain-based BPMS

One of the essential parts of the proposed model is how the rule engine can be integrated with an existing blockchain-based BPMS. This not only requires a well-defined interface description of how the Business Process Engine (BPE) can call the rule engine, but also the execution engine has to bring specific properties with it so that the rule engine can be fully functional.

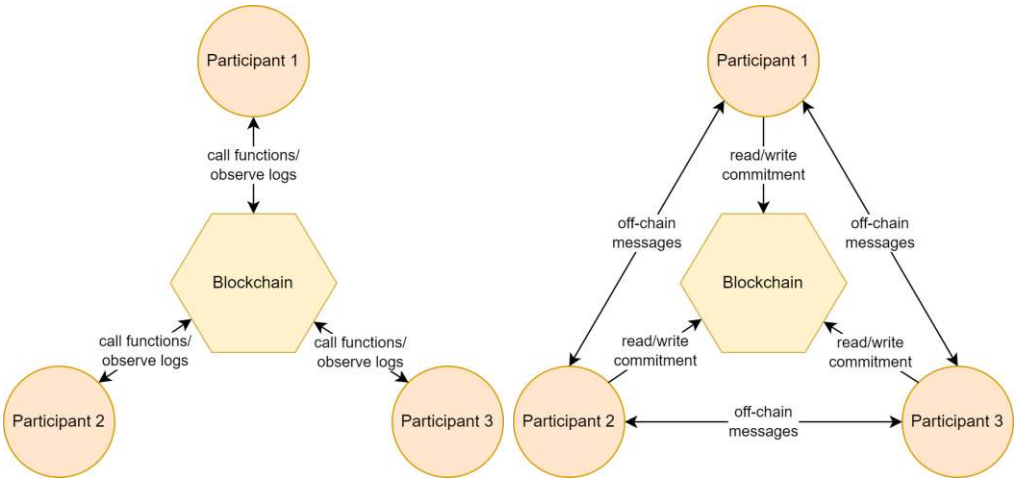
Our literature review (Section 2.4) indicates two competing approaches to realize a blockchain-based BPMS. The first approach is to completely implement the system as a set of smart contracts on a blockchain capable of executing smart contracts (Ethereum, for example). Usually, a smart contract is deployed for each new business process instance. The process participants call specific functions on these smart contracts via transactions to advance the process to its subsequent state. The smart contract checks if the state transition is valid (e.g., if the transaction's sender is allowed to perform the particular transition or if all the business rules are met). In fact, this largely corresponds to the original definition of a smart contract[7].

Nonetheless, this approach has some considerable downsides. One of them is that all the data that is relevant for the execution of the business process or that is subject to business rules must be stored on the blockchain. Unfortunately, storing data is cost-intensive on a blockchain such as Ethereum. Further, this means the data is accessible to all blockchain network participants. This might be a considerable problem if the data contains sensitive or private information, which often is the case in the setting of inter-organizational business processes.

[7]The term *smart contract* was first defined by Nick Szabo in 1996 as "[...] a set of promises, specified in digital form, including protocols within which the parties perform on these promises." [Sza96, p. 50]

The second approach is to use DLTs only for a common source of truth. In this case, the BPMS itself is completely implemented off-chain. It can be seen as a replicated state machine: the state (workflow definitions, workflow instances, and other shared data) is replicated to all system participants. If a party performs a state transition, a message that reflects the state change is created. This message is then used to calculate a commitment (e.g., a cryptographically secure message hash) that is stored on the blockchain. After that, the message is sent to all other system participants. Each receiver can recalculate and compare the commitment to the sender's on-chain commitment. If the values match, the state change gets incorporated into the local state of the node.

The other participants can also deny the state transition proposal if the values do not match. In this case, the sender must undo its local state transition. This way, the system reaches eventual consistency, whereas the abovementioned approach guarantees strong consistency. However, the problems from the on-chain approach are solved using this off-chain approach. It produces significantly fewer costs because only the commitments must be stored on the blockchain, which requires far less memory than the system's whole state. Also, the exchanged data remains confidential.



(a) BPMS fully implemented on-chain (b) BPMS with off-chain state replication

Figure 3.5: A comparison of blockchain-based BPMS approaches.

Figure 3.5 shows schematic representations of the two approaches in comparison. They differ significantly in how a rule engine, as we imagine, could be integrated with them. Based on the drawbacks of the on-chain approach, we argue that the off-chain approach is the more promising one in our setting. We have therefore decided to focus on this approach for the remainder of the thesis.

3.3.1.1 Requirements of the Rule Engine to the BPE

In order for the evaluation of the rules to work as designed, it is essential that the BPE also meets some requirements. During our concept creation, we identified the following requirements of the Rule Evaluator for the BPE.

Verification of Off-Chain State Transition Messages The execution engine of the BPMS must validate the incoming state transition messages in several ways:

- Validate the signatures: We expect the off-chain state transition messages to be at least digitally signed by the sending participant. Optionally, it could also be signed by additional parties. The BPE must verify the validity of the signatures. This ensures integrity, authenticity, and non-repudiation of the message.
- Verify against the on-chain commitment: As mentioned before, the sending participant stores a commitment of the message on the blockchain. The BPE of the receiving parties must verify the off-chain message against the on-chain commitment (e.g., recalculating the hash code of the message and comparing it to the value stored on-chain).
- Verify whether the state transition is valid: The BPE must validate if the state transition from the previous state to the newly proposed one is, in principle, allowed (i.e., if the process model defines a transition between the two states).

Eventual Consistency As in most distributed systems, the decentralized BPE must ensure eventual consistency. This property has to be ensured in good cases where the verification mentioned above succeeds but also in bad instances in which this verification fails. This could, for example, be accomplished by using a simple two-phase commit protocol, which we briefly explain within the following lines.

The participant issuing the state transition sends the state transition message to all other participants. The other participants verify the message and respond with an acceptance or rejection message. If all participants accept the state transition, the issuing participant informs all other participants to commit it. If a single participant answered with a rejection, the issuing participant would inform all other participants to ignore the state change. A disadvantage of this simple approach is that a single participant could block the execution of the whole process. Another possibility is to employ a generalized consensus algorithm such as Tendermint[8].

Prevent Replay Attacks Replay attacks are a form of attack in which a valid transmission is re-used fraudulently. The BPE must prevent scenarios like these: Alice, Bob, and Mallory participate in workflows using the decentralized process execution

[8]Tendermint is a Byzantine Fault Tolerant (BFT) consensus algorithm used to secure and validate transactions in distributed networks [Kwo14].

engine. Mallory recorded a valid (i.e., signed and compliant with all rules) state transition message of Alice during a previous workflow enactment. She now resends the same message in the current process execution to trick Bob into believing Alice changed the process state.

A simple solution to prevent replay attacks is incorporating nonces in the state transition messages. The nonces should be unique for each message. The recipient can check if the nonce has been used before and reject the message if it has.

Causal Message Ordering An important property the BPE must ensure is a causal ordering of the received messages. A happens-before relation can formalize such causal ordering. If messages a and b are received for the same process, and a occurs before b, then the two are in a happens-before relation, or $a \rightarrow b$. The relation is also transitive, i.e., if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$. The relation defines a total ordering for processing state transition messages of a single process [Lam78b].

Ensuring causal ordering of the messages is crucial for the rule evaluation to be deterministic, i.e., come to the same result on all non-faulty nodes. The necessity of this property can again be explained best using an example. For an illustration of the messages and their order, please see Figure 3.6. Alice sends the following state transition messages in this order to Bob: $\Delta_1 = (state : producing, amount : 50)$[9], $\Delta_2 = (state : producing, amount : 100)$, and $\Delta_3 = (state : production\_finished, amount : 100)$. These state transition messages are causally related via the happens-before relation as described before. Additionally, a state transition constraint from state *producing* to state *production_finished* is defined: $\rho = amount \doteq 100$[10]. The constraint is fulfilled for Alice, and she will end up in state *production_finished*. Bob processes the messages in the following order: $\Delta_2, \Delta_1, \Delta_3$. Bob will reject the state transition Δ_3 because constraint ρ is not fulfilled for him. Thus, Bob will stay in state *producing*, leading to an inconsistent system state between Alice and Bob.

The causal message ordering also includes messages that change the environment state in a way that is relevant for the processing of another message. Assume that the state transition message from the example before Δ_3 has another constraint $\rho_{acc} = sender(\Delta) \in environment\_state.roles.production\_manager$, specifying that a user performing the state transition must have the role 'production manager'[11]. Now, if the rule ρ_{acc} is evaluated under the correct environment state $\sigma^E[t]$, the rule evaluates to *true* as in this state, the user had the required role. However, if the environment $\sigma^E[t+1]$ would be used for evaluation, the rule would evaluate to $false$. Thus, the rule evaluation would not be deterministic.

[9] This describes a state transition which sets the shared state variable 'state' to the value *producing* and the variable 'amount' to 50. We use this notation for state transitions from now on.

[10] The meaning of the rule is that the shared state variable 'amount' must be equal to 100.

[11] For a more detailed explanation of this rule, please refer to Section 3.3.4.

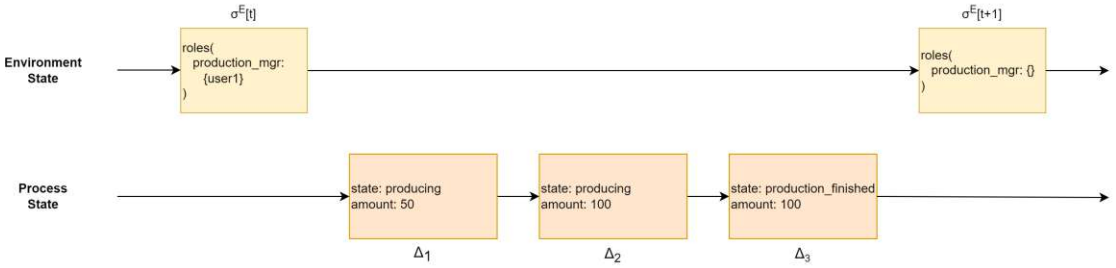


Figure 3.6: Evolution of the environment state and the process state of the BPE

There are several possibilities for ensuring causal message ordering in a distributed system, for example, Lamport timestamps [Lam78b] or, again, a generalized consensus algorithm such as Tendermint [Kwo14].

Data and Data Retention In general, the data that a BPMS manages can be classified into a hierarchy with three layers, as shown in Figure 3.7 [Wes19]:

- The *configuration data* contains all the core information the BPMS requires to do its work. For example, the business process models and rule definitions in our scenario.
- *Environment data* contains data needed during the enactment of multiple business processes and thus has a lifetime independent of a single business process. Examples of such environment data could be role assignments (for resource allocation), customer order history, etc.
- The *process data* represents the data model of a single business process execution. It contains all the required data while enacting a single business process.

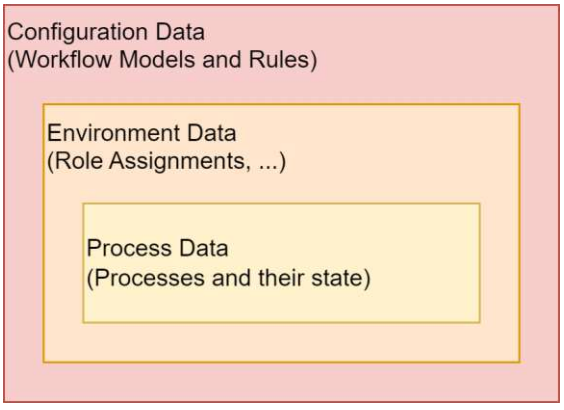


Figure 3.7: Hierarchy of data managed by the BMPS

Therefore, the environment and process data are primarily relevant for the execution of the business processes. Due to the requirements, it must be possible to refer to data from these two categories in the rules of our rule engine. Nevertheless, it must be possible to verify the compliant execution of the processes for external stakeholders. Thus, the BPE must provide a traceable change history for all entities within these two data categories. This could, for example, be accomplished by following an approach such as *Event Sourcing*. Here, not the complete state of an entity is persisted, but all changes to the state are recorded as a sequence of events (i.e., the state transitions).

3.3.2 Architecture

We now introduce the architecture of a decentralized BPMS utilizing a rule engine to ensure compliance-by-design. Figure 3.8 shows a diagram of the components of the BPMS and their relations (adapted from Weske [Wes19]). The roles of the components of this architecture are as follows:

Business Process Modeler The business process modeling subsystem is used to create or change business process models, containing information such as the structure of the process, the participants, and the activities.

Rule Creator As part of the business process modeling subsystem, the rule creator's task is to support the definition of rules based on the data models of the processes. It allows for defining global integrity constraints and state transition constraints for specific state transitions.

Business Process Model Repository The business process model repository holds the business process models and the rule definitions created by the business process modeling and rule creator components.

Business Process Environment The business process environment is responsible for triggering the instantiation of a specific business process based on its model for enactment.

Process Execution Engine The execution engine is the core component of the BPMS. Based on the process model, it is responsible for instantiating and controlling the process enactment. It communicates with the blockchain to create and verify commitments. It also calls or is called by external systems that are integrated into the process execution.

Rule Evaluator The execution engine uses the Rule Evaluator to verify the compliant execution of the process according to the defined rules.

External Systems External systems such as Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM) are integrated over the BPE to participate in a business process enactment.

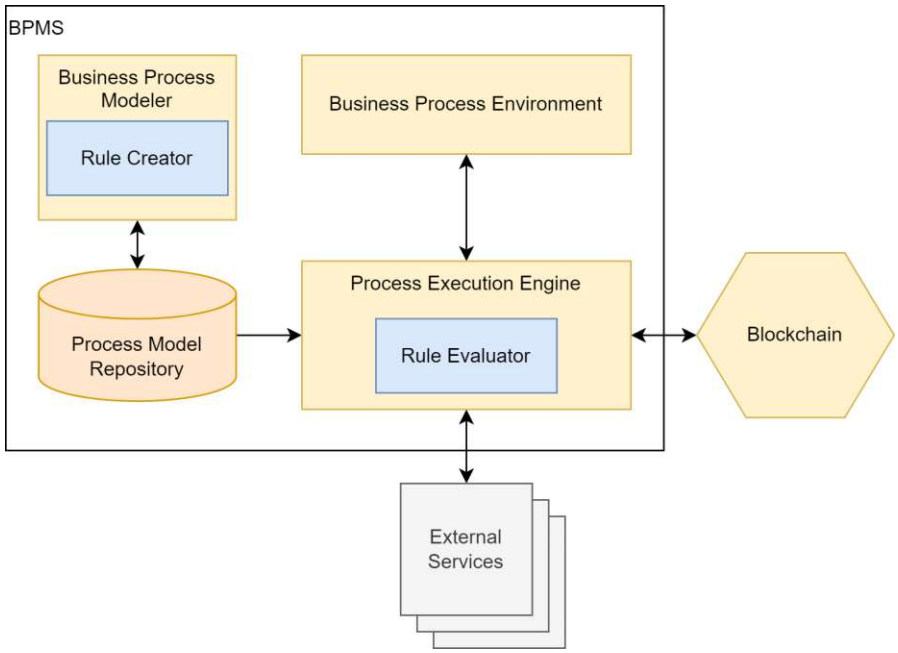


Figure 3.8: Architecture of a blockchain-based BPMS with rule engine (adapted from [Wes19])

The remainder of this chapter is mainly focused on the interactions between the process execution engine, the rule evaluator, and the blockchain. Figure 3.9 shows a sequence diagram of this interaction with two participants. The two-phase commit protocol mentioned above is left out for simplicity, as it is not necessary for an interaction involving only two participants.

An external system wants to change the state of the process. The BPE uses the Rule Evaluator to verify the modified state against the rules defined in the process model. If the rule evaluation fails, a callback to the external system signals this circumstance. Please note that this verification of the rules on the side of Participant 1 (the sending participant) is not strictly required. Its purpose is to reduce costs since invalid states are not committed unnecessarily to the blockchain and sent to the other participants.

If the rule evaluation on the sender's side succeeds, the BPE creates a commitment on the used blockchain. For example, the BPE could calculate a hash code of the state transition message and store it via a transaction on the blockchain. After that, a message including the state transition data, the commitment, and the ID of the blockchain transaction is sent to Participant 2 (the receiving participant). There, the first step is to verify the received state transition data against the commitment, i.e., recalculating the hash code and comparing the result to the value stored on the blockchain. If this step succeeds, the

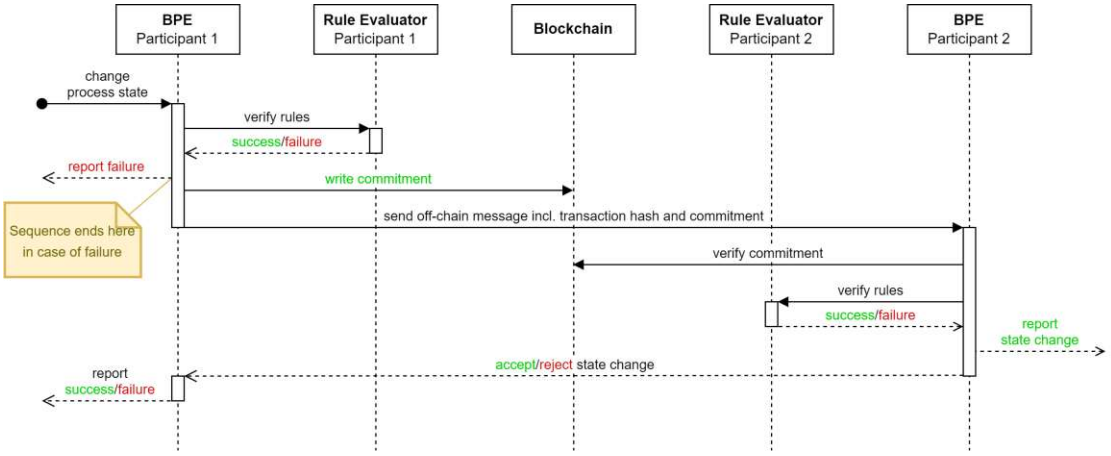


Figure 3.9: Interaction of the BPE and the Rule Evaluator with two participants

receiver's BPE uses its Rule Evaluator to verify the compliance of the state transition. If the evaluation succeeds, the required external system of the receiver is informed about the state change. After that, a message about the acceptance of the transition is sent to Participant 1, where the BPE can inform an external system about the successful state transition if required. If the rule evaluation on the receiver's side fails or the commitment of the message cannot be verified, a rejection message is sent to Participant 1. There, the BPE must call the external system to trigger a compensating transaction (e.g., revert the state change).

3.3.3 Rule Evaluation

In the upcoming sections, we further specify how the Rule Evaluator evaluates the rules and how determinism and verifiability can be ensured for this evaluation.

The Rule Evaluator needs the following information to be able to evaluate all rules as defined by the requirements:

- **A finite set C of constraints**: These are the constraints (or rules) the Rule Evaluator needs to evaluate. As defined in the requirements, we support two kinds of constraints: global constraints that must hold for the whole execution time of the process and state transition constraints that must hold for a particular state transition. The BPE must collect the global and state transition constraints relevant for the transition as defined by the state transition message Δ and provide them to the Rule Evaluator.
- **The current process state $\sigma^p[t]$**: The BPE needs to provide the current process state, including the state transition that led to the current rule evaluation. The rules are always evaluated *a-posteriori* in our concept. This is important because the rule engine must not know how the state transition message Δ needs to be incorporated

into the process state. This may vary significantly for different business processes or maybe also within a single process. For example, in an artifact-centric process, the process state represents all the artifacts as defined by the data model of the process. The state transition message represents the changes some of these artifacts required to perform the state transition. Conversely, if the collaborative business process is realized as process choreography, there is no or just a minimalistic shared process state. Therefore, the state transition messages must include all relevant information the receiving participant needs to perform their work.

Providing the previous process state to the rule evaluation might also be necessary for some use cases. It is the case if there are rules that must compare the resulting process state with the previous one. For example, if a rule needs to check if a counter variable is always incremented by 1. In our scenarios, such rules do not exist. Therefore, we currently do not consider the previous process state for simplicity.

- **The current environment state** $\sigma^E[t]$: The environment state is required by the Rule Evaluator as it contains information that can survive a single process execution. It may include user-role assignments which are used by resource allocation and access control (see Section 3.3.4) or other information from the 'outside' world that needs to be stored within the environment state to make external calls unnecessary (see Section 3.3.3).

- **The state transition message** Δ: This may initially seem redundant because the current process state already incorporates the state change specified by Δ. However, as explained above, there might be scenarios such as process choreographies where the process state contains little or no information, and the whole information needed by the receiving participant is included in the state transition message. With the state transition message at hand, it is also possible to define rules on it to specify which changes can be performed via a specific state transition.

We can now formally define how the rules are evaluated in first-order logic.

Formal Definition Let $\sigma^p[t]$ be the process state of a process instance p at the time t and let $\sigma^E[t]$ be the environment state of the system at time t. Υ is the state transition function that applies a given state transition Δ to the current process state $\sigma^p[t]$ such that $\sigma^p[t+1] = \Upsilon(\sigma^p[t], \Delta)$. Let C^p be a finite set of global integrity constraints for process instance p and C^p_Δ be a finite set of state transition constraints for the state transition described by Δ of process p. The state transition Δ is only valid if the following relation holds:

$$\sigma^p[t+1] = \Upsilon(\sigma^p[t], \Delta) \text{ iff. } \sigma^E[t], \sigma^p[t+1] \models C^p \cup C^p_\Delta$$

A simple algorithm for how the Rule Evaluator can evaluate the rules is as follows:

Algorithm 3.1: *check_rules*

Input: An unordered finite set C of global constraints and state transition constraints, the current process state σ^p, the current environment state σ^E, and the state transition message Δ

Output: *true* if all constraints evaluate to *true*, else $false$.

```
1 valid ← false;
2 forall c ∈ C do
3 |   valid ← eval(c, σ^p, σ^E, Δ);
4 |   if valid = false then
5 |   |   return false;
6 |   end
7 end
8 return true;
```

3.3.3.1 Determinism

Determinism is an essential property for rule evaluation. On the one hand, it guarantees that all system participants eventually end up in the same state. On the other hand, it also ensures verifiability for external stakeholders that verify the compliant execution of the workflows at a later time. For example, an auditor wants to confirm whether all CDWs were adequately transported to a recycling contractor as described in our DiCYCLE use case. They could do so by replaying the process execution and checking if the compliance rules were fulfilled in every state. This would not be possible without a deterministic rule evaluation.

To reach such a deterministic rule evaluation, we have to solve several problems. The first one is handling *time*. As defined in the requirements for the rule engine, it must be possible to define rules over date and time values. An example of such rules is *UC3-R1* which requires the inspection to be conducted within certain intervals. This means that the difference between the date of the last inspection and the current date when the state of the workflow transitions from *Perform inspection* to *Send report* must be within this interval. Thus, the system must have a notion about the current date and time. We clearly can not use the actual current date and time as this would make the rule evaluation indeterministic (the result of the rule evaluation would depend on the moment when it is performed). We have to use a common reference time, which is the same for all participants and can also be used for later verification.

A simple solution for the mentioned problem would be a timestamp created and signed by the sending participant and sent as part of the state transition message. When the other participants receive the message, they compare the timestamp to their current date and time. If the timestamp is within a certain threshold, they accept the transition. The problem with this solution is that it is not externally verifiable, e.g., by an auditor.

If the participants collude, they can lead the auditor to believe that the execution was compliant, even if it was not.

A better solution is to use the timestamp of the blockchain transaction that represents the state transition, i.e., the rule evaluation for the respective state transition must be performed by using the timestamp of the related blockchain transaction as the current date and time. The disadvantage of this choice is that there is not yet a blockchain transaction when the sender evaluates the state transition on its side for cost reduction reasons. However, in this case, the sender can use the actual current date and time. This opens a small time window in which the rule evaluation on the sender side could succeed and could fail for the other participants. Still, this would be fine because the system would still stay in a legal and consistent state. After all, the other participants would reject the state transition. However, this solution only works for processes and constraints in which some seconds of delay are not critical due to the delay until a new block is mined.

Another problem for determinism is the non-termination of the rule evaluation. The rule evaluation must succeed or fail within a finite amount of time to ensure that all participants eventually reach the same system state. Blockchains such as Ethereum use a concept like *gas* for this problem. Gas is used to quantify the amount of computational resources a particular operation or transaction consumes. Each operation or instruction in a smart contract has a gas cost associated with it, and transactions must provide a certain amount of gas to cover these costs. If a transaction or smart contract execution runs out of gas before completion, it will be reverted, and any changes made during execution will be discarded. Unfortunately, we can not use such a concept in our case, as the rule evaluation is performed off-chain. However, we do not expect to need anything comparable either. The difference between the EVM and our rule engine is that the EVM is a Turing-complete execution environment, i.e., it can perform arbitrarily complex computations. In contrast, we expect our rule language to be more likely a non-Turing complete domain-specific language. This makes it easier to use and also reduces the risk of non-termination. As specified in the requirements, the rule language must only allow loops over a finite number of array items. Still, further research must be done on tackling possible non-termination of the constraint evaluation.

The last problems we have identified are external calls (e.g., retrieving data from an external data store) and randomness during the evaluation of the rules. We do not expect the need for randomness in the area of inter-organizational business processes. Therefore, we will not investigate this issue any further.

A solution to eliminate the need for external calls could be to integrate the required information into the environment data of the BPE. An additional program could detect changes to the information outside the system and update the data within the BPE accordingly. Such programs are often referred to as *oracles* in the area of DLT. Also, one category of external calls could be deterministic and thus be allowed during the evaluation of rules. These are calls to a blockchain network (e.g., to call a zero-knowledge proof verifier contract). This could be the same blockchain used to settle the commitments but

also a different one. However, it must be considered that the call must deterministically return the same result. Therefore, we must not invoke the smart contract for the most recent block of the blockchain but must go back to the block height, where the transaction that contained the commitment to the state transition was included. If the same blockchain is accessed that is used for the commitment transaction, this would be the same block. For example, the Ethereum JavaScript client *web3js* allows specifying a block number for all interactions with the blockchain.

3.3.3.2 Verifiability

We define verifiability as the ability to verify the compliant process execution during and after the enactment of the process for involved participants and external stakeholders. Several already mentioned building blocks ensure verifiability.

The first building block is the commitment that the sender of a state transition message stores on the blockchain. It ensures the authenticity and non-repudiability of the message, as well as a provable time stamp that is included in the block header. In other words, the message's sender commits to its content and compliance with the business process and rules. By acknowledging the state transition, the other participants certify the transmitted state transition's correctness and compliance.

The second building block is the traceable change history for all entities the BPE has to store, as mentioned in Section 3.3.1.1. The participants and even an external stakeholder can verify the compliance of the process at each point in the history of the process, even after the execution of the process has finished.

The third and last building block is the determinism of the rule evaluation to ensure that the verification consistently achieves the same result. The result will be the same regardless of when the rules are evaluated.

3.3.4 Resource Allocation and Access Control

An interesting problem in our domain is resource allocation and access control. For a centralized system, it is relatively easy to solve. All users interacting with the business process get accounts with the required permissions. For a decentralized system, the problem is more complicated. By its nature, there is no single system to configure users and their permissions. Still, the other participants and sometimes external stakeholders want to verify that only authorized persons executed a given state transition. Therefore, specifying how such a decentralized system can implement resource allocation and access control is essential.

For our concept, this introduces the problem of where to locate resource allocation and access control functionally. The use cases described at the beginning of this chapter show that resource allocation and access control differ depending on the business process. In some scenarios, storing user-role assignments in the environment data of the BPE might be ok. In other scenarios, such as a supply chain process where a producer does not want

a buyer to know its suppliers, other mechanisms may be needed to ensure the anonymity of the suppliers.

One option is to locate this functionality in the BPE itself. However, this seems not to be a wise decision. The resource and access control functionality must be extensible to integrate custom or new authentication methods, and since we defined that the rule engine must be extensible (**R16**), we argue that this functionality is better located within the rule engine itself. This design decision has several benefits:

- The rule engine should prove extensible and dynamic enough to support a wide range of such resource allocation and access control patterns. As described in the requirements, it must come with a basic set of patterns but can also be extended to provide new and not yet anticipated methods.
- Based on the process model and the set of basic patterns provided by the rule engine, it should be possible to generate rules representing the required patterns of the model.
- The expression language of the rule engine should be easy to learn, even for people without programming skills (**R7**). Thus, it should be easy to define rules specifying resource allocation and access control for complex scenarios that can not be represented in a simple business process model.

We now further describe in-depth how each of the required resource allocation patterns mentioned above and access control can be realized.

Direct Allocation Direct allocation is the simplest resource allocation pattern. The process model already specifies the individual capable of performing a state transition. A rule ρ specifying that a user with an identifier $user\_id$ can execute a given state transition can be expressed the following way:

$$\rho = sender(\Delta) \doteq user\_id$$

We assume that the BPE verifies the validity and authenticity of the off-chain message. Therefore, we can be sure that the respective user, in fact, executed the state transition. What is left to be checked is whether the user is allowed to perform the transition or not. $sender(\Delta)$ gives us the identifier of a user performing the state transition. The result can be compared to a constant value $user\_id$. $user\_id$ could be the address of the blockchain account of the user or another means of uniquely identifying a user over all participants.

Dynamic Allocation In dynamic allocation, the user capable of performing a state transition is determined during the execution of a business process instance. For our rule engine, dynamic allocation works quite similarly to direct allocation, except that the sender of a state transition message is not compared to a constant value but to a value that must be part of the shared process state or environment state. Formally, a dynamic allocation rule ρ can be represented like:

$$\rho = sender(\Delta) \doteq process\_state.user\_id,\ process\_state = (user\_id : some\_id)$$

The expression makes use of the shared process state to delay the resource allocation until the process enactment. The user capable of performing the state transition is stored as part of the process state. Some action before this particular transition must set this shared state variable.

Role-Based Allocation Role-based allocation does not define a specific individual allowed to perform an activity but a role. The user-role assignments must be held in some storage that is available for the rule evaluation. Our concept has two options: the process state or the environment state. If the assignments are stored in the process state, they must be defined for every new process instance of a process model. Usually, it will be more favorable that these assignments 'survive' the enactment of a single process instance and are available to all processes. Hence, the assignments should be held in the environment state of the BPE. A rule ρ that checks if a user has a role assignment for a role *some_role* can be expressed the following way:

$$\rho = sender(\Delta) \in environment\_state.roles.some\_role,$$
$$environment\_state = (roles = (some\_role = \{\dots\}))$$

The expression evaluates to true if the sender of the state transition message Δ is contained in the set *environment_state.roles.some_role*.

Separation of Duties Separation of Duties is the most complex of these resource allocation patterns. For example, it could require that two different people perform a state transition together. This can also be generalized to an m-out-of-n scheme. Sometimes, it is combined with role-based allocation to ensure that people performing the activity have distinct organizational roles.

We assume that multiple users can sign the off-chain messages in our concept. So, every user participating in the state transition could sign the state transition message Δ to express their consent. Still, one of them is ultimately responsible for sending the on-chain transaction. By using the set of signers, we can express an m-out-of-n signing scheme in the following way:

$$\rho = signers(\Delta) \subset can\_sign \wedge |signers(\Delta)| \doteq m$$

The *signers* relation gives a set of signers for a state transition message Δ. *can_sign* specifies the set of users eligible to perform the state transition together.

Access Control Another problem that is easy to implement in a centralized system but hard in a decentralized one is access control. We define access control as restricting which shared state variables a participant can modify during a state transition. Reading a particular variable can not be prohibited as the state is replicated to all participants. However, writing a variable should only be allowed for authorized users. We can define a rule that restricts the modification of the process state to some specified variables formally as follows:

$$\rho = changed\_variables(\Delta) \subset \{var_1, var_2, \dots\}$$

The *changed_variable* relation gives a set of variables that the state transition message Δ changes. This set must be a subset of the variables allowed to change during the particular transition.

CHAPTER 4

Implementation

Based on the model presented in Section 3.3, we have implemented a proof-of-concept prototype of the Rule Evaluator. The Rule Creator, as described in the architecture, is out-of-scope of this thesis. In the upcoming sections, we describe which decisions led to the choice of an appropriate expression language and present the design of the prototype.

4.1 Expression Language

Selecting an appropriate expression language is the most crucial choice for our prototype. Therefore, we present which alternatives were evaluated and what the advantages and disadvantages of each are.

4.1.1 Datalog

Datalog is a declarative logic-oriented programming language. Syntactically, it is a subset of the Prolog programming language, but their evaluation models differ significantly [Conc]. It is used as a query language for deductive databases or program analysis. In the last years, it has been used to formalize aspects of the EVM to identify security vulnerabilities in smart contracts [AS19].

A Datalog 'program' consists of facts and rules. Facts are statements that unconditionally hold to be true. Rules are horn clauses that allow deducing new facts from already known ones. Listing 4.1 shows such a simple Datalog program. The facts define a graph via a set of edges between vertices identified by an integer value. As mentioned, we use rules to infer new knowledge from the facts. In this example, we define rules for paths in the graph. The first rule can be read as: "There is a path from x to y if x and y are connected by an edge". The second rule says: "There is a path from x to y if there is a path from x to some z and an edge from that z to y". The query at the end of the program will now yield all paths in the graph ending at vertex 4: (1, 4) and (2, 4).

Listing 4.1: A simple Datalog program

```
// facts
edge(1,2).
edge(2,3).
edge(2,4).

// rules
path(x,y) :- edge(x,y).
path(x,y) :- path(x,z), edge(z,y).

// query
?- path(x,4).
```

Advantages

- Datomic[1] comes with an integrated query engine based on Datalog. If this database system is used by the BPE for state storage, the rules could be defined via Datalog.

Disadvantages

- If the BPE uses a more common data storage technology such as an SQL or No-SQL database, the process and environment states must be translated into Datalog facts before the rules can be applied.
- Based on the facts representing the process and environment states, a Business Analyst would have to define rules to deduce further knowledge. This could be hard due to the rather uncommon declarative approach of Datalog.

4.1.2 JsonLogic

JsonLogic is a relatively new approach for representing logical expressions in a JSON format. It was designed to share logic between a front- and a backend. By design, JsonLogic can be directly applied to JSON data [Woc]. Therefore, it seems like a good fit for our rule engine, as we expect JSON to be the primary format for data exchange between the instances of the decentralized BPE.

JsonLogic represents the rules as an abstract syntax tree. The keys of the JSON are the operators. The values associated with the key are the values on which the operation is applied. JsonLogic already has various pre-defined operators such as logical, numerical, and array operations. There are also operators to retrieve data from the provided data object. Additionally, most implementations allow the definition of custom operators. Listing 4.2 demonstrates how the rule from Section 3.3.1.1 can be represented in JsonLogic.

[1] Datomic (`https://www.datomic.com`) is a distributed database that supports ACID transactions. A fundamental feature of Datomic is that it allows tracking changes to the stored data over time.

Listing 4.2: Demonstration of JsonLogic by representing the example rule from Section 3.3.1.1

```
var rules = {
  "and": [
    {
      "==": [
        { "var": "amount" },
        100
      ]
    },
    {
      "==": [
        { "var": "state" },
        "production_finished"
      ]
    }
  ]
};

var data = { "state" : "production_finished", "amount" : 100 };

jsonLogic.apply(rules, data);
// result: true
```

Advantages

- With the rise of RESTful web APIs, JSON also gained importance as a data interchange format for other types of services. We expect JSON to be the main format for the data exchange between the participants of the BPE. JsonLogic can then be directly applied to these messages.
- Most implementations of JsonLogic allow the definition of additional custom operators. This would meet requirement **R16**, which requires that the expression language must be extensible.
- There exist implementations for many different programming languages.

Disadvantages

- The main disadvantage of JsonLogic is the same as for all JSON-based formats: it is hard to read and write for a human. There would need to be an additional tool for the business analysts in the Rule Creator component, which allows the definition of the rules in a more user-friendly way (e.g., a graphical editor). The final JsonLogic expressions could then be generated from this model.

4.1.3 Friendly Enough Expression Language

The Friendly Enough Expression Language (FEEL) is specified as a part of the Decision Model and Notation (DMN) by the Object Management Group (OMG). It was specifically designed to be easy to understand for both business professionals as well as developers. DMN allows the graphical representation of decisions in business processes. It can be seamlessly integrated with BPMN. Separating the process from decisions allows leaner process models and faster adaption of business rules to changing requirements without changing the business process models. FEEL expressions are used to describe various aspects of these decisions, e.g., their conditions. The DMN standard [Obj23] specifies a graphical notation for FEEL expressions (so-called *boxed expressions*) and the plain expression language. We will focus on the expression language from now on.

FEEL supports various data types such as numbers, strings, dates, and lists. Additionally, there are many built-in operators and functions on these data types. Extending the language with user-defined or externally-defined functions is also possible. User-defined functions are defined as FEEL expressions and can only use already defined operations. Externally-defined functions can be implemented using theoretically any programming language and can be used to realize functionality that is impossible using FEEL alone.

Every FEEL expression gets evaluated under a context. A context is a special data type representing a list of key-expression pairs. The key is the name associated with the expression. The expression can be any other valid FEEL expression, such as a literal of any data type or a user-defined function. The context is used to resolve variables referenced in the expression to their values.

Listing 4.3 shows how the example rule from Section 3.3.1.1 can be realized using FEEL. The context data type of FEEL looks almost identical to JSON. In fact, every JSON can be directly converted to FEEL without any problems. JSON supports the data types string, number, boolean, array, objects (a mapping of property names to any valid JSON value), and the special value *null*. FEEL supports the same data types and more and can be seen as an extension of JSON [Obj23].

Listing 4.3: Demonstration of FEEL by representing the example rule from Section 3.3.1.1

```
1 // context
2 {
3   amount: 100,
4   state: "production_finished"
5 }
6
7 // expression
8 state = "production_finished" and amount = 100
9 // result: true
```

Advantages

- Fully specified syntax and semantics.
- Relatively easy to write and understand, even for non-developers.
- FEEL is already established in business process modeling as part of DMN.
- Various ways to extend the language with additional functions.
- All built-in operators and functions are side-effect-free per specification.

Disadvantages

- At the time of writing this thesis, there is only one non-proprietary, fully standard-compliant implementation.

4.1.4 Discussion

Datalog is an uncommon approach for defining constraints on data. While there are some applications for trust management languages [LM03], the main application areas are in academia. Moreover, we would have to solve the problem of how we can transform the process and environment state into facts to support verification by rules. JsonLogic is a promising alternative. It can be directly applied to JSON data, comes with various predefined operators, and can be extended with additional operators. The main disadvantage is the bad readability. JSON was designed to be processed by computer programs, not humans. Complex constraints are nearly impossible to comprehend. FEEL is the best of the mentioned alternatives. It is specifically designed to be easy to write and understand. The syntax and semantics are thoroughly specified, making differences in the behavior of standard-compliant implementations unlikely. As part of the DMN standard, it is already established in business process modeling. We decided to use FEEL as an expression language for our prototype for these reasons.

4.2 Prototype Design

In the upcoming sections, we describe the prototype implementation of the Rule Evaluator as proposed using state-of-the-art technologies. The prototype will be used for evaluation later on. Not all functionalities of the use cases are fully implemented. The aim was to demonstrate the technological feasibility of the concept and provide guidance for a production-ready implementation. The main guideline that influenced the design of the prototype was that it must be easy to integrate with an existing blockchain-based decentralized BPE. To reach this goal, the Rule Evaluator should fulfill the following properties:

- **Stateless**: The Rule Evaluator does not keep a state of some sort.
- **Single responsibility**: The only responsibility of the Rule Evaluator is to evaluate the rules against the current state. It must not know how to incorporate a state transition message into the current process state. Together with the statelessness mentioned before, this means that all information required to evaluate the rules of a particular request must be part of the request.
- **Dependencyless**: The Rule Evaluator should be independently deployable without any dependencies to perform its work.
- **Well-definded but also generic interface**: The Rule Evaluator must provide a well-defined interface to make it easy to use for a client application. Nevertheless, the interface must also be as generic as possible. The Rule Evaluator should not have to be adapted for each new business process.

We, therefore, decided to implement the Rule Evaluator as a RESTful web API with JSON as data format. This has the following advantages:

- JSON gained importance as a data interchange format over the last few years, not only for RESTful services. Thus, we expect the BPE, in which our prototype gets integrated, also to use JSON as the message format for messages between the participants. This means no data conversion is necessary for calls to the Rule Evaluator. However, even if the BPE uses another format, most other formats can be converted to JSON and vice versa due to its dynamic semi-structured nature.
- RESTful web APIs are platform-agnostic. Such services can be consumed from almost every programming language and environment.
- RESTful web APIs provide a well-defined interface. By using an OpenAPI specification, the client code for interacting with the API can be automatically generated.

Figure 4.1 shows a UML class diagram of the prototype. The following technologies were used for implementation:

- **Java**: Java is still one of the most used general-purpose programming languages. The main reason for choosing it is that the used FEEL implementation is implemented with Scala. Scala, just like Java, targets the Java Virtual Machine (JVM) as an execution environment. Therefore, the library can be easily used from a Java program.

- **FEEL-Scala**[2]: As already mentioned above, we chose FEEL as expression language for the prototype. At the time of writing this thesis, FEEL-Scala is the only non-proprietary, fully standard-compliant implementation of FEEL. FEEL-Scala is an open-source project available on GitHub and mainly implemented by Camunda using the Scala programming language.

- **Spring and Spring Boot**[3,4]: Spring is an application development framework for the Java platform. Together with Spring Boot, it is often used to create stand-alone web applications quickly and easily.

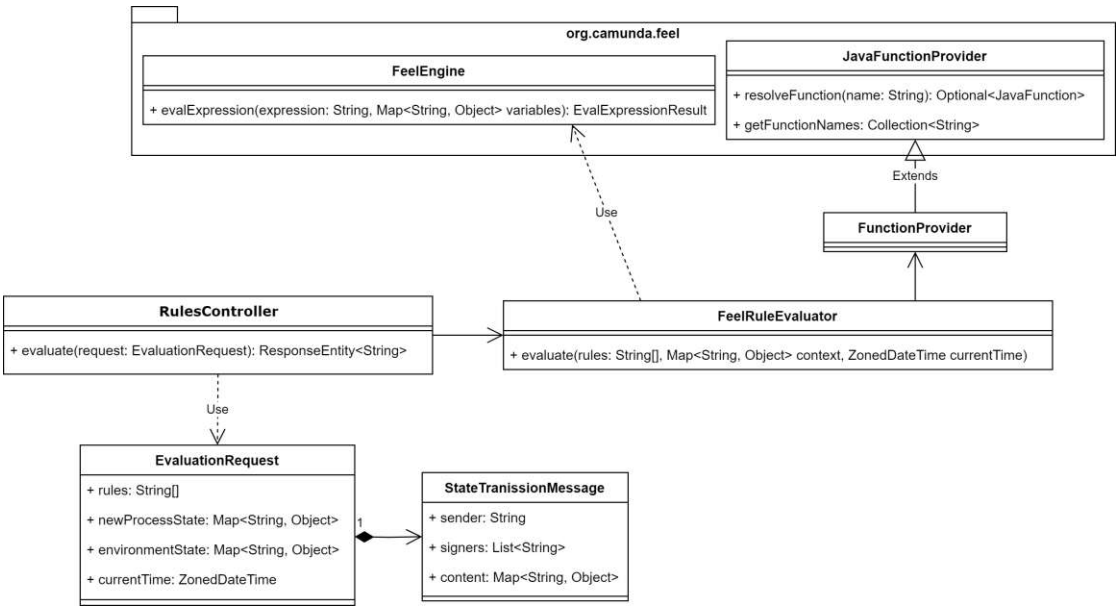


Figure 4.1: Class diagram of the Rule Evaluator prototype

4.2.1 Interface

The aim was to keep the interface of the Rule Evaluator as simple as possible. It provides a single REST endpoint `\rules\evaluate`. This endpoint can only handle `POST` requests from a client. Listing 4.4 shows an example request. The data that must be contained in the request body resembles the information required for the rule evaluation as specified in the concept in Section 3.3.3.

[2] https://camunda.github.io/feel-scala/
[3] https://spring.io/projects/spring-framework
[4] https://spring.io/projects/spring-boot

Listing 4.4: Example request to the Rule Evaluator

```
{
  "rules": [
    "processState.state = \"production_finished\" and processState.
       amount = 100"
  ],
  "processState":   {
    "amount": 100,
    "state": "production_finished"
  },
  "environmentState": {
  },
  "delta": {
    "sender": "6d510352-f8a0-473f-b880-54a17e6c6923",
    "signers": [
      "6d510352-f8a0-473f-b880-54a17e6c6923"
    ],
    "content": {
      "amount": 100,
      "state": "production_finished"
    }
  },
  "currentTime": "2023-08-23T13:34:05Z"
}
```

`rules` contains the rules defined in FEEL the Rule Evaluator should evaluate. Every expression in this collection must resolve to a boolean value. Unary expressions, as defined by the DMN standard, are not allowed. As mentioned in the concept, we are currently supporting two kinds of rules: *global rules* that must be fulfilled over the whole process execution and *state transition rules* that must be fulfilled for a particular state transition to be performed. From the perspective of the Rule Evaluator, these two variants do not differ. The calling BPE must ensure to provide all necessary rules to each call to the Rule Evaluator as specified by the process model.

`processState` must contain the process state under which the rules should be evaluated. The interface definition does not specify a precise data schema for this property. It can contain an arbitrary JSON object. This flexibility is required to support different business processes without adapting the interface. Under some circumstances, there might not be a shared process state. For example, if the collaborative process is realized as choreography. In these cases, the property `processState` can be left empty.

`environmentState`, as the name suggests, contains the environment state for the process. Like `processState`, it can contain an arbitrary JSON object. It can be left empty if no environment state exists or if it is not necessary for rule evaluation.

`delta` contains metadata about the state transition message. `sender` must contain an identification of the user who sent the message. `signers` contains a list of user

identifiers who signed the message. The distinction between the user who ultimately sent the state transition message (and therefore committed the on-chain transaction) and the signers of the message is necessary to support the more sophisticated resource allocation schemes described in Section 3.3.4. `content` must contain the payload of the state transition message, i.e., all the data required to perform the transition.

`currentTime` must contain a timestamp under which the rules should be evaluated. As specified in the concept, this must be the actual current time for evaluation on the sender's side or the timestamp of the blockchain transaction for evaluation on the receiver's side.

The response of the `\rules\evaluate` endpoint depends on whether the rules are fulfilled or not. If all rules resolve to the boolean value `true`, then the Rule Evaluator will respond with `HTTP 200 OK`. If at least one rule evaluated to the boolean value `false` or caused an error during the evaluation, the Rule Evaluator will respond with `HTTP 400 Bad Request`. In these cases, the response body contains an array of objects with an `index` and a `message` for each problem. The `index` property corresponds to the index of the rule in the `rules` collection of the request, and `message` contains a text further describing the error.

4.2.2 Rule Evaluation

As mentioned in the introduction to FEEL, every FEEL expression is evaluated under a context. The context is used to resolve variable names within the expression. Thus, the first step for the rule evaluation in the Rule Evaluator is to create the context from the request body. All properties from the request, except `rules` and `currentTime`, are copied as they are to the context. The resulting context of the example request from Listing 4.4 can be seen in Listing 4.5.

Listing 4.5: The context created for FEEL rule evaluation for the example request from Listing 4.4

```
{
  processState: {
    amount: 100,
    state: "production_finished"
  },
  environmentState: { },
  delta: {
    sender: "6d510352-f8a0-473f-b880-54a17e6c6923",
    signers: [ "6d510352-f8a0-473f-b880-54a17e6c6923" ],
    content: {
      amount: 100,
      state: "production_finished"
    }
  }
}
```

The DMN standard specifies that addressing context entries is path-based. If a rule needs to address, for example, the `amount` property of the state transition message payload, this can be done using the path expression `delta.content.amount` in the rule.

Another important part of the rule evaluation is handling the current time. FEEL provides the built-in function `now()`, which returns the current date and time. The FEEL-Scala library offers an extension point that allows to specify the value to return as the result for `now()`. In our case, we always return the `currentTime` value of the request. Listing 4.6 shows a shortened version of the code used to evaluate the rules from the request.

Listing 4.6: Shortened version of the rule evaluation code

```
1  public void evaluate(String[] rules, Map<String, Object> context,
       ZonedDateTime currentTime)
2  {
3    FeelEngine engine = new FeelEngine.Builder()
4      .clock(new FeelEngineClock() {
5        @Override
6        public ZonedDateTime getCurrentTime() {
7          return currentTime;
8        }
9      })
10     .build();
11
12   for (int i = 0; i < rules.length; i++) {
13     Either<FeelEngine.Failure, Object> result = engine.evalExpression
         (rules[i], context);
14
15     // handle result
16   }
17 }
```

4.2.3 Extensibility

An important requirement for the Rule Evaluator is **R16**. It requires the expression language to be extensible. The DMN standard already specifies such extensibility for FEEL via so-called *externally-defined* functions. These externally-defined functions can be implemented in any programming language and can then be called from FEEL expressions. FEEL-Scala complies with the standard and provides the *function provider* extension point. A function provider is a class that extends the `JavaFunctionProvider` class. It can be registered with the FEEL engine via the `FeelEngine.Builder` class. When the FEEL engine evaluates an expression and detects a function call to a function that is neither built-in nor user-defined, the function provider will be used to resolve it. Listing 4.7 shows a shortened version of the `FunctionProvider` class we have implemented for our prototype. It shows the implementation of the `has allowed changes` function, which can be used to restrict the properties allowed in a state transition message.

Listing 4.7: Shortened version of the FunctionProvider class implemented in our prototype

```
1  public class FunctionProvider extends JavaFunctionProvider {
2
3    private static final Map<String, JavaFunction> functions = new
         HashMap<>();
4
5    static {
6      final JavaFunction hasAllowedChanges = new JavaFunction(Arrays.
           asList("context", "allowedKeys"), args -> {
7        final ValContext context = (ValContext) args.get(0);
8        final ValList allowedKeys = (ValList) args.get(1);
9
10       List<Val> allowdKeyItems = CollectionConverters.asJava(
             allowedKeys.items());
11       Set<String> variables = new HashSet<String>(
             CollectionConverters.asJava(context.context().
             variableProvider().getVariables()).keySet());
12
13       for (Val val : allowdKeyItems) {
14         variables.remove(((ValString) val).value());
15       }
16
17       return new ValBoolean(variables.size() == 0);
18     });
19
20     functions.put("has allowed changes", hasAllowedChanges);
21   }
22
23   @Override
24   public Collection<String> getFunctionNames() {
25     return functions.keySet();
26   }
27
28   @Override
29   public Optional<JavaFunction> resolveFunction(String functionName)
         {
30     return Optional.ofNullable(functions.get(functionName));
31   }
32 }
```

In our prototype, all of these additional externally-defined functions are hard-coded and not implemented to the full extent. In a production-ready implementation of the Rule Evaluator, some plug-in mechanism could add additional functions to the language without recompiling the whole Rule Evaluator.

CHAPTER 5

Evaluation

In the upcoming sections, we evaluate the concept and the prototype in several aspects. First, we show the practical utility of the prototype and the used expression language FEEL by showing example implementations of the constraints from the use cases of Section 3.1. These implementations empirically evaluate that the identified requirements and the implemented concept are sufficient to realize the constraints. After that, we integrate our Rule Evaluator prototype into an existing decentralized BPE in Section 5.2. In Section 5.3, we argue about how well the prototype fulfills the specified requirements. In Section 5.4, we discuss the shortcomings of the prototype and how these could be addressed in future work.

5.1 Implementation of the Use Cases

In this section, we empirically prove that our concept and the prototype based on it are suitable to implement the integrity constraints of the use cases as presented in Section 3.1. We discuss design decisions and assumptions we have made during the implementation. We further mention how we had to extend the basic expression language to be able to represent all aspects of the use cases. For some of the rules, we also sketch alternative approaches.

5.1.1 Use Case 1: DiCYCLE

The specialty of the DiCYCLE use case is the artifact-centric approach we have chosen to represent the business process. We assume that the process state (i.e., the whole data model as shown in 3.1.1) gets replicated to all participants of the BPE. Thus, we can show the suitability of our approach for artifact-centric business processes with a fully replicated process state.

5.1.1.1 Implementation of the Rules

The first integrity constraint to demonstrate is *UC1-R1*: the measured amounts of the countersigned transports for a CDW to a recycling contractor must be in the expected range. We can implement this rule using the FEEL expression language by using built-in functions alone. Listing 5.1 shows the implemented rule. We validate the constraint on the state transition between *Recycled* and the final state *Finished* of a CDW. The rule can be read like this: "For all CDWs of a project that are in the final state `finished`, the sum of `determinedMeasure` of the transports must be included in the expected range or must have a `deviationReport` set."

Listing 5.1: UC1-R1

```
all(
  for c in processState.project.cdws[item.state = "finished"]
    return (includes(
      [c.expectedMeasureFrom..c.expectedMeasureTo],
      sum(c.transports.determinedMeasure)) or
      c.deviationReport != null))
```

The following constraint *UC1-R2* is the first constraint where we make use of the environment state: The recycling contractor must be entitled to the treatment of a specific CDW. Usually, a governmental agency keeps a public record of these recycling companies. We assume this public record gets replicated to the environment state of our BPE. Thereby, we can check if a type of a particular CDW is contained in the list of entitlements of the recycling contractor. We add this rule to the state transition between *Planned* and *Settled Recycling Contractor* of a transport. This allows us to transport a CDW to different recycling contractors as long as they are entitled. Listing 5.2 shows the final rule in the FEEL expression language.

Listing 5.2: UC1-R2

```
all(
  flatten(
    for c in processState.project.cdws return
      for t in c.transports[item.state = "settledRecyclingContractor"
          ] return
        list contains(
          get value(environmentState.entitlements, string(t.
              toRecyclingContractorId)),
          c.type)))
```

The implementation of *UC1-R3* also makes use of the environment state. The rule requires that only employees of a recycling contractor can countersign a transport. As mentioned in the concept, we expect these role assignments to be part of the environment state so that they can be shared between different process executions. As shown in Listing 5.3, the implementation checks that the user who countersigned the transport must be contained in the recycling contractor's role `canCountersign`.

An alternative approach for this constraint would be to check if the sender of the state transition message is part of the role. However, this requires the maintenance of a registry of keys that are assigned to users.

Listing 5.3: UC1-R3

```
all(
  flatten(
    for c in processState.project.cdws return
      for t in c.transports return
        if t.approvedById != null then
          list contains(
            get value(environmentState.canCountersign, string(t.
               toRecyclingContractorId)),
            t.approvedById)
        else true))
```

Constraint *UC1-R4* is relatively easy to implement using FEEL. We add the rule from Listing 5.4 to the transition from *Demolishing* to *Finished* of the demolition project to check that all CDWs of the project are in their final state *Finished*.

Listing 5.4: UC1-R4

```
every c in processState.project.cdws satisfies c.state = "finished"
```

Listing 5.5 shows the implementation of *UC1-R5*. We check the rule for the transition between *Transported* and *Measured* of transport. Alternatively, we could add the same implementation as a global integrity constraint of the process, i.e., it would be checked for all modifications of the process state. However, keeping the number of global integrity constraints rather small could be a performance consideration to reduce the number of constraints that must be evaluated on every state transition.

Listing 5.5: UC1-R5

```
every c in processState.project.cdws satisfies
  (every t in c.transports[item.state = "measured"] satisfies
    t.determinedMeasureUnit = c.expectedMeasureUnit)
```

UC1-R6 is the first presented global integrity constraint. For the whole execution of the process, `expectedMeasureFrom` of a CDW must be greater than 0 and smaller than `expectedMeasureTo`. As seen in Listing 5.6 the implementation of such a global constraint does not differ from state transition constraints. Usually, such constraints will only address the process and environment states and not the state transition message.

Listing 5.6: UC1-R6

```
every c in processState.project.cdws satisfies
  c.expectedMeasureFrom > 0 and c.expectedMeasureTo > c.
     expectedMeasureFrom
```

Listing 5.7 exemplifies how resource allocation and access control can be implemented using the Rule Evaluator for the artifact-centric DiCYCLE process. As already mentioned, the state of the process gets fully replicated to each participant. Without access control, each participant can change every part of the state. Thus, there needs to be a mechanism to specify for a state transition who may execute it and which parts of the state may be modified. The listing implements this aspect for the state transition between *Settled Recycling Contractor* and *Picked up* of a transport. Only a carrier is allowed to perform the transition, and only the fields `state` and `leftAt` of the transport artifact may be changed.
The function `has allowed changes` is not part of the FEEL standard and is implemented as externally-defined function in our Rule Evaluator prototype. The function allows us to specify which context entries are permitted for a FEEL context. The function's implementation can be seen in Listing 4.7.

Listing 5.7: UC1-AC

```
1  list contains(environmentState.roles.carrier, delta.sender)
2
3  has allowed changes(delta.content, ["project"])
4
5  has allowed changes(delta.content.project, ["cdws"])
6
7  all(flatten(
8    for c in delta.content.project.cdws
9      return has allowed changes(c, ["transports"])))
10
11 all(flatten(
12   for c in delta.content.project.cdws return
13     for t in c.transports return has allowed changes(t, ["state", "
         leftAt"])))
14
15 all(flatten(
16   for c in delta.content.project.cdws return
17     for t in c.transports return t.state = "pickedUp" and t.leftAt !=
          null))
```

5.1.2 Use Case 2: Supply Chain Scenario

We can realize this scenario as a collaborative process between the Bulk Buyer and the Manufacturer. The Middleman, Supplier, and Carrier do not directly participate in the implementation. The process requires only a small shared process state containing information about the order.

5.1.2.1 Implementation of the Rules

The constraint *UC2-R1* can be verified when the Manufacturer sends the status report and the invoice to the Bulk Buyer. Listing 5.8 shows how we can validate whether the correct discount was applied. Additionally, we can assert if the total sum is correctly calculated.

Listing 5.8: UC2-R1

```
every p in delta.content.invoice.positions satisfies
  p.qty < 1000 or p.discountPercentage >= 2

every p in delta.content.invoice.positions satisfies
  (p.pricePerUnit * (100 - p.discountPercentage) / 100) * p.qty = p.
     totalPrice

delta.content.invoice.total = sum(delta.content.invoice.positions.
   totalPrice)
```

For the same message, we can validate whether the Supplier delivered the intermediates on time and whether the quality checks were performed correctly. Remember that the two companies are in coopetition, so the Manufacturer wants to keep the quality report of its Supplier private from the Bulk Buyer. We solve this issue in a way that demonstrates the advantages of the extensibility of our prototype by utilizing zero-knowledge proofs.

Zero-knowledge proofs were first introduced by Goldwasser et al. [GMR19]. Such a proof system allows one party (the prover) to prove to another party (the verifier) that a statement is true. The verifier gains no further information about the statement apart from the fact that it is true. In our scenario, the prover is the Manufacturer, and the verifier is the Bulk Buyer. The Manufacturer wants to prove to the Bulk Buyer that they performed the quick quality check upon reception of the intermediates and that the results matched those of the Supplier. The proof is sent with the status report, and the Bulk Buyer can verify its correctness. Listing 5.9 shows the implementation of the constraints *UC2-R2*, *UC2-R3*, and *UC2-R4* in FEEL. `is proof valid` is again implemented as a Java function within the prototype of the Rule Evaluator.

Listing 5.9: UC2-R2, UC2-R3, and UC2-R4

```
date and time(delta.content.intermediatesReceivedAt) <=
  (date and time(processState.order.receivedAt) + duration("P20D"))

is proof valid(delta.content.qualityReport.proof)
```

The last constraint of this use case *UC2-R5* ensures that a final test is performed for each product by the Manufacturer. Here, we leverage the resource allocation functionality of our concept. Listing 5.10 shows the implementation of this constraint. At least two users must sign the message, and all of them must have the role *quality control*. The final test for each product must be positive.

Listing 5.10: UC2-R5

```
count(delta.signers) >= 2

every signer in delta.signers satisfies
  list contains(environmentState.roles.qualityControl, signer)

every i in delta.content.delivery.items satisfies
  i.finalCheckPositive
```

5.1.3 Use Case 3: FMCHAIN

To also show the utility of our prototype for process choreographies, we assume the FMCHAIN process to be implemented as such. Choreographies usually have little or no shared process state at all. No participant has ever full control over the process. The information a participant needs to proceed with the choreography by executing its private process must be contained in the exchanged messages. Thus, the implemented rules of this process only address the transition messages.

5.1.3.1 Implementation of the Rules

UC3-R1 prescribes that an inspection must be conducted within specific intervals. We validate if the inspection body gets informed about the outstanding inspection within a specific time frame before the due date. This ensures enough time for the inspection body to conduct the inspection.

Listing 5.11 formalizes this aspect into a FEEL expression. The rules are checked for the message from the facility management company to the inspection body. The expression makes use of the built-in function `now()` of FEEL. This function returns two possible values depending on where the rule gets evaluated. The sender will use the current date and time because there is not yet a blockchain transaction. On the receiver's side, we use the blockchain transaction timestamp for `now()`. Thus, the rule evaluates to true if the transaction gets included in a blockchain block before the due date.

Additionally, we validate if the message's sender is a specific participant and if the message contains all required properties. Alternatively, the BPE could validate the state transition message against a JSON schema. The message must include the due date and the facility to be inspected. `challenge` and `domain` are required for the response of the inspection body and will be explained later.

Listing 5.11: UC3-R1

```
delta.sender = "15881464-7c1a-43a7-b562-0c52aa862207"

has allowed changes(delta.content, ["inspectionRequest"]) and
  has allowed changes(
    delta.content.inspectionRequest,
    ["dueDate", "facility", "challenge", "domain"])

now() <= (date and time(delta.content.inspectionRequest.dueDate) -
   duration("P30D"))
```

Listing 5.12 implements *UC3-R2* and *UC3-R3* and also covers an aspect of *UC3-R1*. The rules get validated for the response message from the inspection body to the facility management company.

The first expression checks if the message is sent before the due date of the inspection. This should ensure the inspection is performed within the prescribed interval. This does not validate if *all* inspections were performed within this interval. A possible solution could be to add the date of the last inspection and the prescribed interval to the environment state. Thereby, we could also validate the interval.

The second expression uses an externally-defined function that is implemented within our Rule Evaluator prototype as a Java function. It verifies whether the report created by the inspection body complies with the regulations.

The last expression shows the future-proofness of the prototype through its extensibility. We want to verify if the inspection body is accredited. Currently, those inspection bodies are included in a publicly accessible register. However, in our prototypical implementation, we make use of the relatively new Verifiable Credentials (VCs) technology. VCs are a concept that allows individuals or entities to securely store and share their personal information digitally while retaining control over who can access it. These credentials are cryptographically signed, enabling verification by third parties without revealing unnecessary sensitive data. Technically, a VC is a JSON document representing any physical or digital credential, such as a driving license. Any person or institution can issue and sign a VC. It is up to the verifier if they trust the issuer or not. If the holder needs to present the credential, for example, to prove an age of at least 18, they create a Verifiable Presentation (VP). Here, the full potential of the technology comes into play. The holder can decide which identity attributes of the VC they want to present. For example, they can decide only to give proof to be older than 18. A verifier learns nothing about the holder of the VC besides the fact that they are older than 18 years. The VP is again a JSON document containing parts of the VC and is signed by the holder [W3C22b].

The issuer and the holder of the VC are identified by Decentralized Identifiers (DIDs). DIDs provide a self-sovereign approach to identity by allowing individuals to create and manage their unique identifiers, free from the constraints of centralized authorities. DIDs

leverage DLTs to ensure portability, privacy, and security in a decentralized manner. A DID is a URI associating the DID subject with a DID document stored in a registry. The DID document contains, among other information, the cryptographic public keys that can be used to verify the signatures of the VCs and VPs [W3C22a].

In our prototypical scenario, the inspection body holds a VC attesting to be an accredited inspection body for certain facilities. They can prove this fact to the facility management company by sending a VP and the inspection report in the response message after the inspection. The VP contains the `challenge` and `domain` from the initial message to the inspection body to prevent replay attacks. The externally-defined function `is vp valid` verifies the validity of the VP. During this validation, the DIDs of the holder and the issuer must be resolved. This is essentially accessing an external system, which can be a problem for determinism and verifiability of the rules, as we already pointed out in Section Determinism. Therefore, we argue for blockchain-based DID registries. When the DID needs to be resolved, we can access the blockchain with the timestamp we also use for `now()` (i.e., the timestamp of the on-chain transaction containing the commitment). Hence, we can resolve the DID even after years.

Listing 5.12: UC3-R2 and UC3-R3

```
1 now() <= date and time(processState.dueDate)
2
3 is report valid(delta.content.inspectionPerformed.report)
4
5 is vp valid(delta.content.inspectionPerformed.report.inspectionBody)
```

5.2 Integration into the TTSM Prototype

This section shows how to integrate our Rule Evaluator prototype into a decentralized BPE prototype. The prototype for Time-Traveling State Machines (TTSMs) was implemented by Kleebinder [Kle23] and extended by Navratil [Nav23]. Our implementation is based on the extended version from Navratil, which is available on the code hosting website GitHub[1]. We have published our prototype on Zenodo[2] and GitHub[3]. We briefly explain the TTSM prototype and present our contribution in the following.

The TTSM prototype allows the verifiable execution of business processes. It leverages DLTs as a single source of trust. Changes to the process state are sent off-chain to the participants, but commitments are stored on the blockchain. The name-giving time-traveling aspect is reached by using an event-sourcing approach. The system keeps track of the current state as a sequence of changes to this state (events). By specifying a particular timestamp, this approach makes it possible to “travel back in time” and

[1] https://github.com/alexnavratil/ttsm-prototype
[2] https://doi.org/10.5281/zenodo.8374110
[3] https://github.com/lietho/ttsm-prototype

observe the system's state as of this time. The architecture of the prototype is split into four loosely-coupled modules:

- The **Workflow Module** supports the definition, instantiation, and execution of workflows by exposing REST endpoints which an external system can call.
- The **Consistency Module** is responsible for the communication between the system participants and thus keeping their state consistent. It does so by forwarding relevant events to the other participants via peer-to-peer communication technologies.
- The **Rules Module** allows the integration of rule-checking engines. Ultimately, this module is responsible for acknowledging or rejecting a state change. If a state change is performed or received from another participant, it calls the registered rule checkers. If all validation results are positive, a special acknowledgment event is persisted.
- All the modules mentioned above are connected via the **Persistence Module**. It serves as a service bus for the other modules to react to specific events and stores all workflow-related events in an event log. The prototype from Navratil uses OrbitDB for persistence.

5.2.1 Contribution to the TTSM Prototype

5.2.1.1 Extension of the Workflow Definition Format

Although the original prototype design already provides an infrastructure to integrate a rule engine, there was no possibility to enrich a workflow definition with the rules to be verified. Therefore, we extended the custom workflow definition format. It is now possible to define global constraints and state transition constraints. Global constraints can be specified using the `globalConstraints` property of a `WorkflowDefinition` and state transition constraints by using the `when` property of an `EventObject`. Both properties are string arrays. This design allows to define multiple rules using the FEEL expression language as shown in Section 4.2. Currently, the constraints are not validated when the workflow is created. If one of them has an invalid syntax, uses variables of the evaluation context that do not exist, or does not evaluate to a boolean value, this will cause runtime errors during the execution of the workflow. Listing 5.13 shows the modified interfaces. The other interfaces for the workflow definition format were not modified.

Listing 5.13: The modified `WorkflowDefinition` and `EventObject` interfaces

```
// other interfaces were not modified
interface WorkflowDefinition {
  id: string;                    // Identifier of Workflow
  activities: Record<string, ActivityObject>; // Activities
  initial: string;               // Initial Activity
  globalConstraints?: string[]; // Global constraints get validated
      for all state transitions
}
interface EventObject {
  target: string;              // Target Activity
  external?: boolean;          // Externally triggered Transition?
  schema?: object;             // JSON Schema Payload Validation
  assign?: ObjectDefinition;   // Context Assignment Definition
  when?: string[];             // State transition constraints
}
```

5.2.1.2 Rule Evaluator

The next step to integrate our Rule Evaluator prototype into the TTSM prototype was to enhance the internal data structures for the on-chain commitments. The original code only stored the hash of the blockchain transaction. However, we also need the transaction's timestamp for our rule validation. We adapted the relevant classes and interfaces to make the timestamp available.

The rule validation of the TTSM prototype requires specific REST endpoints. `check-new-workflow` checks if a workflow can be created, `check-new-instance` validates if an instance of a workflow can be created, and `check-state-transition` similarly verifies if the given state transition can be performed. This interface differs quite significantly from our Rule Evaluator, which only supports validating state transitions.

To finally integrate the Rule Evaluator, we created an adapter that translates between the different interfaces. The adapter is a REST controller that implements the interface as the TTSM prototype requires. The operations `check-new-workflow` and `check-new-instance` are not implemented and return a positive validation result. `check-state-transition` retrieves the relevant rules from the Persistence Module and builds the request for the Rule Evaluator. Most fields can be easily mapped, but there are also some problems. For example, the TTSM prototype does not support an environment state, so we must leave this field empty for all requests. Finally, the adapter must be registered with the Rule Module to be called when the system performs state transitions.

5.2.2 Discussion

The well-defined interfaces of the two prototypes facilitate a straightforward integration of the Rule Evaluator into the TTSM prototype. Implementing an adapter that translates

between the two interfaces with minimal effort is possible. Nevertheless, some problems must be mentioned. The Rule Module of the TTSM prototype stipulates that the rule engine also checks workflow definitions and instantiations of workflows. This is currently not supported by our Rule Evaluator. It would be conceivable that the syntax of the rules is checked when validating workflow definitions or whether variables not defined in the event payloads or the process state are used in the expressions.

The TTSM prototype does not fulfill all properties for a BPE we have defined in Section 3.3.1.1. This means that not all functionalities of the Rule Evaluator can be used as intended by our concept. The TTSM prototype does not support an environment state. Therefore, it can not be used in the constraints. This affects the rules for resource allocation and if data from external systems is required for rule evaluation. In addition, the messages between the participants are not signed in the TTSM prototype. Only the blockchain transaction is signed by the sender. This implies that there is only one signer per message. However, our Rule Evaluator would support multiple signers. Thus, certain functionalities of resource allocation, e.g., separation of duties, cannot be used.

5.3 Fulfilment of the Requirements

In this section, we evaluate whether our prototype meets the identified requirements. We perform a qualitative evaluation in a descriptive form using informed arguments as proposed by Hevner et al. [HMPR04]. We also compare our solution to approaches from related works, where applicable.

R1 Termination Termination of the rule evaluation largely depends on the power of the used expression language. We use the FEEL expression language in our prototype, which is not fully Turing-complete. In general, there are two possible reasons for the non-termination of programs: loops and recursive functions [GHM+08]. The DMN specification defines the usage of *for loops* over lists and ranges of numbers. The list type is immutable and may only contain a finite number of items. A range must be fully specified via an upper and a lower bound [Obj23]. Thus, it is not possible to implement infinite loops.

Further, it is not possible to define recursive functions. FEEL allows the definition of *user-defined functions* in a context (although this is currently not possible in our prototype), but it is not possible to reference the function itself or another function that is defined later in the context within its body. The DMN standard explicitly specifies this circumstance: "An expression in a context entry may not reference the key of the same context entry but may reference keys [...] from previous context entries in the same context, as well as other values [...] in scope. These references SHALL be acyclic and form a partial order." [Obj23, p. 107]

Based on this premiss, we argue that the rule evaluation will succeed or fail within a finite time and thus terminate. As discussed later, this property may be weakened by *externally-defined functions*.

R2 Determinism As discussed in Section 3.3.3 there are several challenges to reach a deterministic rule evaluation. Referencing the current date and time in the constraints can be made deterministic by using the timestamp of the blockchain transaction. This is entirely rational in practice since this is the point in time at which the state transition is carried out for the other participants of the system.

Another problem for determinism are side effects. These can happen during the evaluation of constraints and when external systems are accessed. FEEL is side-effect free by specification [Obj23]. Accessing external systems can also be side-effect-free if only read operations are performed on systems with the same or stricter guarantees for availability and verifiability as our system (e.g., blockchains).

We argue that the rule evaluation is deterministic as long as the constraints only use built-in FEEL concepts. Again, *externally-defined functions* may cause the rule evaluation to be non-deterministic. An unsolved problem in our concept is randomness. We do not expect this to be relevant to the execution of collaborative business processes. However, our concept is not restricted to this category of processes.

R3 Verifiability Several properties of our concept and requirements to the BPE executing the business processes ensure verifiability. The BPE must retain all changes to the process and environment states. The deterministic rule evaluation helps to verify whether the process execution complies with all constraints even after the process finishes.

However, our concept only guarantees private verification. Only the participants of a particular process can verify its compliant execution. An un-involved third party (e.g., an arbitrary blockchain node) cannot make any statements whatsoever about the execution of the process. If an external auditor wants to verify the correct execution, they must request the data from one of the participants. Then, they can verify the integrity of the data by recalculating the commitment and comparing the result to the commitment stored on the blockchain. The compliant execution can be verified by replaying the state transitions and re-validating the constraints.

R4 Soundness We do not provide a formal soundness proof for our solution as this would go beyond the scope of this thesis. This is a limitation of this work and needs to be addressed in future work.

R5 Completeness We also do not present a proof of completeness here. This problem is left open for future research.

R6 Syntax and Semantics The FEEL expression language we use for our prototype implementation has a well-defined syntax and semantics thoroughly specified by the DMN standard [Obj23]. Every standard-compliant implementation will yield the same evaluation result. Since the standard cannot cover the semantics of *externally-defined functions*, problems can arise here again. The collaborating parties must ensure that

these functions are specified in a similarly thorough manner to avoid misunderstandings and differences in behavior.

R7 Ease of Use FEEL was explicitly designed to have a simple syntax and a simple data model with only a few pre-defined data types [Obj23]. We argue that the language is easy for developers and business professionals to use and understand.

R8 Well-Known Language FEEL is specified as part of DMN to represent and automate decisions in business processes. As such, BPMN allows the seamless integration of DMN via *business rule tasks* [Obj23, Obj14]. DMN has been adopted by major process automation engines such as Camunda Zeebee[4], Oracle Integration Platform[5], or Red Hat Decision Manager[6]. The way FEEL is used in our concept differs from DMN decision models. Nonetheless, we argue that the language itself is well-known in the area of business process modeling.

R9 Constraints FEEL, in general, is not restricted to expressions resolving to a boolean value. However, our concept restrains constraints to be of that type. This requirement is thus fulfilled. At this point, however, we do not validate this property in any way. This would be part of the Rule Creator component, which is out of the scope of this thesis.

R10 Global Integrity Constraints As presented by this work, the Rule Evaluator makes no difference with what kind of constraints it is called. It is the task of the BPE is to determine all relevant rules and send them to the Rule Evaluator for evaluation.

R10.1 Global Integrity Constraints on the Shared Process State As described in the concept and the implementation, one part that is required for evaluating the rules is the shared process state. It can be referenced by the rules to restrict the possible values of its variables. This can happen through global integrity constraints and state transition constraints.

R11 State Transition Constraints The BPE can call the Rule Evaluator with different constraint types without any difference.

R11.1 State Transition Constraints on State Transition Metadata As shown in the implementation, requests to the Rule Evaluator also contain the state transition metadata of the message causing the evaluation. Constraints can restrict the possible values of the variables therein. We use such constraints to realize resource allocation, for example.

[4] https://camunda.com/platform/decision-engine/

[5] https://docs.oracle.com/en/cloud/paas/integration-cloud/user-processes/work-decision-models.html

[6] https://access.redhat.com/products/red-hat-decision-manager/

R11.2 State Transition Constraints on State Transition Data Another part of validation requests to the Rule Evaluator is the state transition data itself. It can also be subject to constraints restricting its variable values. For example, our externally-defined function `has allowed changes` allows us to specify which variables can be contained in the state transition data.

R12 Simple Data Types FEEL supports the following basic data types: number (based on IEEE 754-2008 Decimal128), string, date, time, date-time, days and time duration, years and months duration, and boolean. It can also represent unknown values via the value *null* [Obj23]. Thus, FEEL provides all of the required simple data types.

R13 Complex Data Types The DMN standard specifies two complex data types for FEEL: context and list. A context is an ordered collection of key-expression pairs. The key is a name (which is mapped to a string in the semantic domain of FEEL) or a string. The expression can be any valid FEEL expression, i.e., values of simple or complex data types or further expressions. The context type of FEEL is thus an extension to the required complex type 'object' [Obj23].

Lists in FEEL are similar to the required complex type 'array'. The difference is that lists in FEEL can contain values of different data types. This is also an extension of the requirement.

R14 Expressions Expression in FEEL can resolve to any of the specified data types. Thus, list, context, and temporal expressions are also available besides the required arithmetic and boolean expressions. Conjunction and disjunction are total functions over the semantic domain of FEEL [Obj23].

R15 Control Structures FEEL supports the required control structures `if` and a `for` loop to iterate over lists [Obj23].

R16 Extensibility In Section 4.2.3, we showed how we realized extensibility for our prototype. It is possible to extend FEEL with *externally-defined functions*. These functions are implemented in Java and can be called from FEEL expressions like any other function. We used this functionality in our prototype and implemented use-case-specific functions that could not be implemented using pure FEEL. Currently, the functions are hard-coded into the Rule Evaluator. When a function needs to be modified or a new one is required, the Rule Evaluator must be re-compiled. A production-ready implementation of the Rule Evaluator could provide a plugin mechanism to add or remove custom functions during runtime. Nonetheless, using externally-defined functions raises termination, determinism, and trust problems, as discussed later.

R17 Direct Resource Allocation Direct resource allocation can be achieved using our concept by comparing the sender of the state transition message to a constant value.

Depending on how the process is implemented, this value could be the address of a blockchain account or some other unique user identifier. Our solution coincides with other concepts implementing direct resource allocation such as Sturm et al. [SSSJ18] and Madsen et al. [MGH+18]. Since these approaches are purely on-chain, the checks are performed in the smart contracts. Our validation happens completely off-chain.

R18 Deferred Resource Allocation Deferred resource allocation works almost the same as direct resource allocation in our concept. The difference is that the message's sender is compared to a variable from the process or environment state. Our approach enables binding an actor to a role via setting the respective variable and releasing an actor by resetting the variable. We do not explicitly implement a *vote* operation like López-Pintado et al. [LPDGBW19a] that allows an actor to accept or reject the assignment. However, this could be realized by adapting the business process accordingly.

R19 Role-Based Resource Allocation In our concept, the role assignments are stored in the process or environment state. To validate if the sender of a message has a specific role assigned, we check if it is contained in the respective role assignment list. The advantage of storing the role assignments in the environment state is that their lifetime is not bound to the process execution. The same assignments can be used for multiple process enactments or even different processes. Other solutions for role-based allocation, such as those implemented by Weber et al. [WXR+16] or Ladleif et al. [LWW19], require assigning participants to roles for each new process or choreography execution.

R20 Separation of Duties Compared to blockchain-based solutions, the benefit of our approach is that we can also implement more complex resource allocation schemes such as separation of duties. Pure blockchain-based implementations are bound to the fact that only a single participant can sign and send a transaction. In contrast, the off-chain messages in our concept can be signed by an arbitrary number of users. Thereby, we can implement sophisticated schemes such as m-out-of-n combined with role-based allocation.

5.4 Limitations and Future Work

In the previous sections, we empirically proved the practical utility of our proposed concept for artifact-centric process modeling and process-oriented collaborations and choreographies. We further argumentatively described how well the requirements are met. We now conclude the evaluation by discussing the presented results. We especially point out shortcomings of the concept and prototype design and mention possible future research directions.

Externally-defined Functions Termination and determinism are guaranteed as long as only built-in FEEL functionality is used in the constraints. The syntax and semantics of FEEL are explicitly specified to be free from side effects and possible non-terminating

constructs such as infinite loops and recursions. However, these fundamental properties of our concept can be weakened by using *externally-defined functions*. There are two possibilities for creating an expression language suitable to implement arbitrary constraints in business processes. The language would either need to be Turing-complete, which violates the termination requirement, or it needs to be extensible. For FEEL, the latter approach was chosen by supporting externally-defined functions. Our implementation of the use cases in Section 5.1 highlights that many situations require additional functions. Still, caution is advised when implementing them. The DMN standard does not specify a concrete programming language for them to implement. This depends on the FEEL implementation that is used. Our prototype uses the `FEEL-Scala` library, which allows the implementation of such functions with Java, but theoretically, any other programming language could be used. Java and most other modern programming languages are Turing-complete. For this category of programming languages, we can, in general, not prove if a function call will terminate or execute forever. Moreover, Java is not side-effect-free and allows operations that are prohibited by our concept to ensure determinism, e.g., network calls to retrieve data from an external system. These possible problems must be taken into account if externally-defined functions are implemented.

Additional concerns may emerge depending on if and how the implementation of the functions is shared between the participants. If a single participant is responsible for implementing the function, the other participants must either trust this implementation to be correct or verify its correctness. Another variant to reduce centralization is that each participant implements the function themselves. In this case, the collaborating parties must ensure a comprehensive function specification to avoid misunderstandings and differences in behavior.

Reachability and Dead-Ends As soon as process states are guarded by constraints, as in our concept, problems such as reachability and dead-ends must be considered. Reachability in this context means it is possible to reach a specific state during the execution of the process. Usually, a reachability analysis is performed for the process's final state(s). If no final state is reachable, the process execution will get stuck indefinitely. Such a scenario can arise when the final state, or a state on the path to the final state, is guarded by a constraint that can never be fulfilled. A closely related problem are dead-ends. These occur if the process reaches a valid state that can not be left anymore [BGH+07, HNN09]. In our concept, this can happen when all outgoing transitions are guarded by constraints that can not be fulfilled at the current state. A manual analysis to avoid such problems is only feasible for small process models. As the process models become more complex, a tool for automatic analysis is needed.

Bhattacharya et al. [BGH+07] show that these problems are generally undecidable for artifact-centric process modeling but become decidable under some restrictions. Hull et al. [HNN09] propose LTL-FO, a first-order extension for linear-time temporal logic, to verify such restricted artifact-centric process models automatically. Also BPMN process model verification has been studied. Takemura [Tak08] and more recently Rachdi et

al. [REND16] propose reachability and liveness analysis techniques by transforming the process models into variants of Petri nets.

Correctness and Redundancy of Constraints Another open problem of our concept is verifying the constraints for correctness. As with any other program, the implemented rules can diverge from their specification. However, verifying the correctness, even in edge cases, would further increase the trust in the system. It is also desirable to remove redundant constraints to streamline the process model, making it easier to understand for all stakeholders. A constraint is redundant if it can be removed without increasing the set of possible values for a variable of the shared process state. Further research is necessary on how correctness verification and redundancy detection can be realized with our concept.

Data Perspective Our results align with other research in that a shared understanding of data is essential [SW22, MPB+13, MPB+14]. This work shows that defining constraints is only possible with a comprehensive data model of the process state and the exchanged messages. Artifact-centric modeling is advantageous because creating a data model is fundamental in such process modeling approaches. Approaches such as that of Meyer et al. [MPB+14] can help tackle the challenges of creating uniform data models for process-oriented modeling.

Privacy Our concept for resource allocation relies on access control lists and user identifiers stored in the environment or process states. Even if this approach does not require more information about a specific user, the user identifier only guarantees pseudonymity. Participants of a certain process can track user activities over multiple process enactments. More research is necessary to find solutions to improve the privacy of the users.

Accessing External Systems In Section 3.3.3, we argue that eliminating external calls in the rule evaluation can be achieved by using *oracles* that retrieve the data from the outside world and store it in the process state or environment state of the BPE. However, the use of oracles increases centralization and trust issues. *Centralized* oracles rely on a single, trusted data source or entity to provide information. While they offer simplicity and convenience, they also introduce a single point of failure and potential manipulation. In contrast, *decentralized* oracles source data from multiple independent providers, enhancing reliability and security by reducing the risk of data manipulation or system failures. Decentralized oracles align more closely with the principles of DLTs, fostering trust and transparency. However, they require some conflict resolution mechanism, making them less efficient than centralized oracles [ABRSS20].

We also argue that accessing a blockchain while evaluating the constraints does not negatively affect determinism. Here, further research is necessary to determine which external systems are safe to access and how this affects the main properties of our concept: termination, determinism, and verifiability.

CHAPTER 6

Conclusion

Since the emergence of Distributed Ledger Technologies, many new application areas have been studied. One such area with increasing scientific interest is the management and execution of collaborative business processes. These processes involve multiple external partners working together seamlessly to achieve common goals. The classical approach for realizing such a process is to use a central third party responsible for hosting the required services and serving as a mediator in the case of disputes. DLTs can provide significant advantages here. By providing a secure and transparent way to record, verify, and share data among the parties, these technologies enable a trust-based ecosystem that reduces the need for intermediaries. Moreover, the immutable nature of distributed ledgers enhances data integrity and auditability, making it easier for organizations to comply with terms and regulations.

The research community so far has focused on process-level compliance and mostly ignored the data perspective. Most existing approaches for blockchain-based business process execution require writing code in Solidity, for example, to automatically enforce business rules. This is not only a complex task but also causes increased costs. This work aimed to close this gap by introducing a holistic approach to enforcing business rules or other terms and regulations by also considering the data that is shared by the collaborating parties. These constraints can be specified with an expression language that is easy for business professionals and developers alike, thus fostering a common understanding.

We used the Design Science methodology as proposed by Hevner et al. [HMPR04]. We conducted a narrative literature review to gain background knowledge on the relevant fields *compliance-by-design* and *blockchain-based business process execution*. Based on the literature review, we were able to define the main properties of the rule engine that are relevant for a decentralized evaluation of constraints. We used three real-world business processes to derive further requirements for the rule engine and the used expression

language. Based on these requirements, we proposed a theoretical concept that we implemented as a prototype for evaluation.

We identified *termination*, *determinism*, and *verifiability* as the main properties a rule evaluation has to fulfill in a decentralized system. Constraints containing non-terminating constructs, such as infinite loops, could block other system nodes. It would be possible to perform Denial of Service attacks with specifically crafted rules. The determinism of the rule evaluation is required so that every non-faulty system node eventually reaches the same state when the same input messages are applied. Verifiability is vital for increasing trust in the system. It is not only possible for collaborating participants to verify a compliant process execution of the other participants but also for an external stakeholder, such as an auditor, to verify compliance even after the process has finished. Another important property is *soundness* for reporting the correctness of the current state of the system, i.e., the evaluation logic is never wrong when it claims a rule to be fulfilled.

In order for the properties mentioned above to be met, the BPE into which the rule evaluation is integrated must also meet specific requirements. The most important is ensuring a *causal message ordering* of the received state transition messages. If the messages were not processed in the correct order, reaching determinism for the rule evaluation would not be possible. This includes the messages for a particular process but also changes to the environment state which are relevant to this process. To ensure verifiability for external stakeholders, the BPE must provide a *traceable change history* for all data within the process state and the environment state. This guarantees that it is possible to verify past process executions for compliance.

We have implemented the prototype of the Rule Evaluator as REST API, which allows an easy integration with an existing BPE. The prototype uses the FEEL expression language for defining the constraints. This language is part of the DMN standard and is already established in business process modeling. By integrating the Rule Evaluator into the TTSM prototype and by implementing the constraints from three business processes, we showed the practical utility of our approach. Nonetheless, there is also room for future research. Most notably, it is worth investigating how the extensibility of our prototype can be improved. We make use of *externally-defined functions* as specified by the DMN standard to implement functionality that would not be possible with pure FEEL alone. As we have argued, these functions can significantly weaken our concept's termination and determinism properties. Further research is also needed on how liveness and reachability analysis can be integrated into our concept. With the rapid advancement in the area of zero-knowledge proofs, it could be interesting to analyze if and how such proof systems could enhance the rule evaluation to increase data privacy further.

Our work builds upon the unique properties of DLTs. While the distributed ledger ensures data integrity, the constraints ensure compliance with predefined business rules. Our approach fosters trust by reducing uncertainties about the compliant process execution. This could be a first step towards uniformly representing relevant terms and regulations and their underlying data in decentralized business process execution to reach compliance-by-design.

List of Figures

List of Algorithms

Acronyms

BIM Building Information Modeling. 2, 33, 37

BPE Business Process Engine. 42, 43, 45–50, 53–56, 60, 61, 63, 64, 66, 71, 72, 76, 78, 81–83, 87, 90, 91

BPEL Business Process Execution Language. 27

BPI Baseline Protocol Implementation. 16–19

BPM Business Process Management. 20, 24, 26

BPMN Business Process Model and Notation. 2, 20, 21, 28–30, 36, 38, 86, 91

BPMS Business Process Management System. 21, 23–25, 29, 43–45, 47–49, 91

CCSM Consensus-Controlled State Machine. 17, 19

CDW Construction and Demolition Waste. 33–36, 40, 52, 72, 73

CRM Customer Relationship Management. 49

DCR Dynamic Condition Response. 29

DID Decentralized Identifier. 77, 78

DLT Distributed Ledger Technology. xvii, xix, 1, 2, 4, 5, 7, 8, 12, 16, 17, 23–26, 29, 31, 37, 44, 53, 78, 87, 89, 90

DMN Decision Model and Notation. xv, xvii, 27, 62, 63, 66, 68, 81–84, 86, 90

ECA Event Condition Action. 31

ERP Enterprise Resource Planning. 16, 49

EVM Ethereum Virtual Machine. 10, 11, 14, 53, 59

FCL Formal Contract Language. 27

FEEL Friendly Enough Expression Language. xv, xvii, 62–68, 71–76, 79, 81–86, 90

JSON JavaScript Object Notation. 60–64, 66, 76, 77

LTL-FO Linear Temporal Logic with First Order quantification. 23, 86

MSA Master Service Agreement. 37

NFT Non-Fungible Token. 12

OMG Object Management Group. 20, 62

PBFT Practical Byzantine Fault Tolerance. 12

REST Representational State Transfer. 19, 61, 64, 65, 79, 80, 90

TTSM Time-Traveling State Machine. xix, 78–81, 90

UML Unified Modeling Language. 5, 64

UTXO Unspent Transaction Output. 9

VC Verifiable Credential. 77, 78

VP Verifiable Presentation. 77, 78

VSM Virtual State Machine. 18

Bibliography

[ABRSS20] Hamda Al-Breiki, Muhammad Habib Ur Rehman, Khaled Salah, and Davor Svetinovic. Trustworthy blockchain oracles: Review, comparison, and open research challenges. *IEEE Access*, 8:85675–85685, 2020.

[Aca22] Glassnode Academy. Utxo vs. account-based chains. `https://academy.glassnode.com/concepts/utxo`, 2022. (Accessed on 2022-03-26).

[ACC+18] Tara Astigarraga, Xiaoyan Chen, Yaoliang Chen, Jingxiao Gu, Richard Hull, Limei Jiao, Yuliang Li, and Petr Novotny. Empowering business-level blockchain users with a rules framework for smart contracts. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11236 LNCS:111–128, 11 2018.

[AS19] Monika Di Angelo and Gernot Salzer. A survey of tools for analyzing ethereum smart contracts. *Proceedings - 2019 IEEE International Conference on Decentralized Applications and Infrastructures, DAPPCON 2019*, pages 69–78, 4 2019.

[Bas] Baseline protocol introduction. `https://docs.baseline-protocol.org/baseline-basics/the-baseline-protocol`. (Accessed on 2022-04-15).

[BGH+07] Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4714 LNCS:288–304, 2007.

[BHS09] Kamal Bhattacharya, Richard Hull, and Jianwen Su. A data-centric design methodology for business processes. In *Handbook of research on business process modeling*, pages 503–531. IGI Global, 2009.

[BSA14] Alysson Bessani, João Sousa, and Eduardo E.P. Alchieri. State machine replication for the masses with bft-smart. 2014.

[BSS21] Dominik Breitfuß, Goran Sibenik, and Marijana Sreckovic. Digital traceability for planning processes. *SSRN Electronic Journal*, 2021.

[But20] Vitalik Buterin. Ethereum whitepaper | ethereum.org. *Ethereum.org*, 2020.

[CCD+19] Claudio Di Ciccio, Alessio Cecconi, Marlon Dumas, Luciano García-Bañuelos, Orlenys López-Pintado, Qinghua Lu, Jan Mendling, Alexander Ponomarev, An Binh Tran, and Ingo Weber. Blockchain support for collaborative business processes. *Informatik Spektrum 2019 42:3*, 42:182–190, 5 2019.

[CDW] Construction and demolition waste: challenges and opportunities in a circular economy — european environment agency.

[CH09] David Cohn and Richard Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull*, 32, 2009.

[Cona] Ethereum Contributors. Proof-of-stake (pos) | ethereum.org. `https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/#finality`. (Accessed on 2023-07-23).

[Conb] Ethereum Contributors. Scaling | ethereum.org. `https://ethereum.org/en/developers/docs/scaling/`. (Accessed on 2023-07-23).

[Conc] Wikipedia Contributors. Datalog - wikipedia. `https://en.wikipedia.org/wiki/Datalog`. (Accessed on 2023-06-20).

[cond] Wikipedia contributors. Sarbanes–oxley act. `https://en.wikipedia.org/wiki/Sarbanes%E2%80%93Oxley_Act`. (Accessed on 2023-09-16).

[CPNX20] Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. A survey on ethereum systems security. *ACM Computing Surveys (CSUR)*, 53, 6 2020.

[DHPV09] Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic verification of data-centric business processes. *ACM International Conference Proceeding Series*, 361:252–267, 2009. formal verification of data-centric processes; sehr theoretisch; wahrscheinlich eher für future research interessant.

[DRMR18] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. Fundamentals of business process management: Second edition. *Fundamentals of Business Process Management: Second Edition*, pages 1–527, 3 2018.

[EH18] Jacob Eberhardt and Jonathan Heiss. Off-chaining models and approaches to off-chain computations. 2018.

[ET17] Jacob Eberhardt and Stefan Tai. On or off the blockchain? insights on off-chaining computation and data. volume 10465 LNCS, 2017.

[FHBL19] Ghareeb Falazi, Michael Hahn, Uwe Breitenbücher, and Frank Leymann. Modeling and execution of blockchain-aware business processes. *Software-Intensive Cyber-Physical Systems*, 34:105–116, 6 2019.

[Fin16] Klint Finley. A $50 million hack just showed that the dao was all too human. https://www.wired.com/2016/06/50-million-hack-just-showed-dao-human/, 2016. (Accessed on 2022-03-29).

[FKRMR22] Walid Fdhila, David Knuplesch, Stefanie Rinderle-Ma, and Manfred Reichert. Verifying compliance in process choreographies: Foundations, algorithms, and implementation. *Information Systems*, page 101983, 1 2022.

[FOT21a] Andreas Freund, Anais Ofranc, and Kyle Thomas. Baseline CCSM Requiements Version 1.0. https://github.com/eea-oasis/baseline-standard/blob/main/ccsm/baseline-ccsm-v1.0-psd01.md, 09 2021. (Accessed on 2022-04-22).

[FOT21b] Andreas Freund, Anais Ofranc, and Kyle Thomas. Baseline Core Specification Version 1.0. https://github.com/eea-oasis/baseline-standard/blob/main/core/baseline-core-v1.0-psd01.md, 09 2021. (Accessed on 2022-04-22).

[Fou19] CertiK Foundation. The blockchain trilemma: Decentralized, scalable, and secure? https://medium.com/certik/the-blockchain-trilemma-decentralized-scalable-and-secure-e9d8c41a87 10 2019. (Accessed on 2022-04-01).

[FRMKR15] Walid Fdhila, Stefanie Rinderle-Ma, David Knuplesch, and Manfred Reichert. Change and compliance in collaborative processes. *Proceedings - 2015 IEEE International Conference on Services Computing, SCC 2015*, pages 162–169, 8 2015.

[GGSGL+20] Julian Alberto Garcia-Garcia, Nicolas Sanchez-Gomez, David Lizcano, M. J. Escalona, and Tomas Wojdynski. Using blockchain to improve collaborative business process management: Systematic literature review. *IEEE Access*, 8:142312–142336, 2020.

[GHM+08] Ashutosh Gupta, Thomas A Henzinger, Rupak Majumdar, Andrey Rybalchenko, and Ru-Gang Xu. Proving non-termination. In *Proceedings*

of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 147–158, 2008.

[GMR19] Shafi Goldwasser, Silvio Micali, and Chales Rackoff. The knowledge complexity of interactive proof-systems. *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 10 2019.

[GV06a] Stijn Goedertier and Jan Vanthienen. Compliant and flexible business processes with business rules. volume 236, pages 94–103, 2006.

[GV06b] Stijn Goedertier and Jan Vanthienen. Designing compliant business processes with obligations and permissions. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4103 LNCS:5–14, 2006.

[HBC+16] Richard Hull, Vishal S. Batra, Yi Min Chen, Alin Deutsch, Fenno F.Terry Heath, and Victor Vianu. Towards a shared ledger business collaboration language based on data-aware processes. volume 9936 LNCS, pages 18–36. Springer Verlag, 2016. vorteile von BC.

[HHS20] Abdelatif Hafid, Abdelhakim Senhaji Hafid, and Mustapha Samih. Scaling blockchains: A comprehensive survey. *IEEE Access*, 8:125244–125262, 2020.

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly: Management Information Systems*, 28:75–105, 2004.

[HNN09] Richard Hull, Nanjangud C. Narendra, and Anil Nigam. Facilitating workflow interoperation using artifact-centric hubs. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5900 LNCS:1–18, 2009.

[Hul17] Richard Hull. Blockchain: Distributed event-based processing in a data-centric world. 2017.

[JP14] Paul Johannesson and Erik Perjons. A method framework for design science research. *An Introduction to Design Science*, pages 75–89, 2014.

[Kle23] Daniel Kleebinder. Time-travelling state machines for verifiable bpm. Master's thesis, TU Wien, 2023.

[Kwo14] Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1(11):1–11, 2014.

[Lam78a] Leslie Lamport. The implementation of reliable distributed multiprocess systems. *Computer Networks (1976)*, 2, 1978.

[Lam78b] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21, 1978.

[LM03] Ninghui Li and John C. Mitchell. Datalog with constraints: A foundation for trust management languages. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2562:58–73, 2003.

[Loh13] Niels Lohmann. Compliance by design for artifact-centric business processes. *Information Systems*, 38:606–618, 6 2013.

[LPDGBW19a] Orlenys López-Pintado, Marlon Dumas, Luciano García-Bañuelos, and Ingo Weber. Dynamic role binding in blockchain-based collaborative business processes. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11483 LNCS:399–414, 6 2019. RBAC.

[LPDGBW19b] Orlenys López-Pintado, Marlon Dumas, Luciano García-Bañuelos, and Ingo Weber. Interpreted execution of business process models on blockchain. *Proceedings - 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference, EDOC 2019*, pages 206–215, 10 2019.

[LPDGBW22] Orlenys López-Pintado, Marlon Dumas, Luciano García-Bañuelos, and Ingo Weber. Controlled flexibility in blockchain-based collaborative business processes. *Information Systems*, 104, 2 2022.

[LPGBD+19] Orlenys López-Pintado, Luciano García-Bañuelos, Marlon Dumas, Ingo Weber, and Alexander Ponomarev. Caterpillar: A business process execution engine on the ethereum blockchain. *Software - Practice and Experience*, 49, 2019.

[LSNW20] Tom Lichtenstein, Simon Siegert, Adriatik Nikaj, and Mathias Weske. Data-driven process choreography execution on the blockchain: A focus on blockchain data reusability. *Lecture Notes in Business Information Processing*, 389 LNBIP:224–235, 2020.

[LWW19] Jan Ladleif, Mathias Weske, and Ingo Weber. Modeling and enforcing blockchain-based choreographies. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11675 LNCS:69–85, 2019.

[Men18] Jan Mendling. Towards blockchain support for business processes. *Lecture Notes in Business Information Processing*, 319:243–248, 2018.

[MGH+18] M F Madsen, M Gaub, T Høgnason, M E Kirkbro, T Slaats, and S Debois. Collaboration among adversaries: Distributed workflow execution on a blockchain. volume 7, 2018.

[MOR20] Marcel Müller, Nadine Ostern, and Michael Rosemann. Silver bullet for all trust issues? blockchain-based trust patterns for collaborative business processes. *Lecture Notes in Business Information Processing*, 393 LNBIP:3–18, 2020.

[MPB+13] Andreas Meyer, Luise Pufahl, Kimon Batoulis, Sebastian Kruse, Thomas Stoff, Dirk Fahland, and Mathias Weske. Data perspective in process choreographies : Modeling and execution. *Tech. Rep. BPM-13-29, BPMcenter. org.*, 2013.

[MPB+14] Andreas Meyer, Luise Pufahl, Kimon Batoulis, Sebastian Kruse, Thorben Lindhauer, Thomas Stoff, Dirk Fahland, and Mathias Weske. Automating data exchange in process choreographies. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8484 LNCS:316–331, 2014.

[MWA+18] Jan Mendling, Ingo Weber, Wil Van Der Aalst, Jan Vom Brocke, Cristina Cabanillas, Florian Daniel, Søren Debois, Claudio Di Ciccio, Marlon Dumas, Schahram Dustdar, Avigdor Gal, Luciano García-Bañuelos, Guido Governatori, Richard Hull, Marcello La Rosa, Henrik Leopold, Frank Leymann, Jan Recker, Manfred Reichert, Hajo A. Reijers, Stefanie Rinderlema, Andreas Solti, Michael Rosemann, Stefan Schulte, Munindar P. Singh, Tijs Slaats, Mark Staples, Barbara Weber, Matthias Weidlich, Mathias Weske, Xiwei Xu, and Liming Zhu. Blockchains for business process management - challenges and opportunities. *ACM Transactions on Management Information Systems (TMIS)*, 9, 2 2018.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[Nav23] Alexander Navratil. Distributed off-chain storage for inter-organizational business process execution. Master's thesis, TU Wien, 2023.

[Obj14] Object Management Group. *Business Process Model and Notation*, 01 2014. Version 2.0.2.

[Obj23] Object Management Group. *Decision Model and Notation*, 04 2023. Version 1.4.

[Pro21] Chainlink Project. What is an oracle in blockchain? » explained. `https://chain.link/education/blockchain-oracles`, 2021. (Accessed on 2022-04-09).

[PSHW20] Christoph Prybila, Stefan Schulte, Christoph Hochreiner, and Ingo Weber. Runtime verification for business processes utilizing the bitcoin blockchain. *Future Generation Computer Systems*, 107:816–831, 6 2020.

[RAHE05] Nick Russell, Wil M.P. Van Der Aalst, Arthur H.M. Ter Hofstede, and David Edmond. Workflow resource patterns: Identification, representation and tool support. *Lecture Notes in Computer Science*, 3520:216–232, 2005.

[RD05] Florian Rosenberg and Schahram Dustdar. Business rules integration in bpel - a service-oriented approach. *Proceedings - Seventh IEEE International Conference on E-Commerce Technology, CEC 2005*, 2005:476–479, 2005.

[REND16] Anass Rachdi, Abdeslam En-Nouaary, and Mohamed Dahchour. Liveness and reachability analysis of bpmn process models. *Journal of computing and information technology*, 24:195–207, 6 2016.

[RGG+18] Michel Rauchs, Andrew Glidden, Brian Gordon, Gina C. Pieters, Martino Recanatini, François Rostand, Kathryn Vagneur, and Bryan Zheng Zhang. Distributed ledger technology systems: A conceptual framework. *SSRN Electronic Journal*, 2018.

[Rho11] Ellen A Rhoades. Literature reviews. *Volta review*, 111(3):353–368, 2011.

[Sch90] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22, 1990.

[SGN07] Shazia Sadiq, Guido Governatori, and Kioumars Namiri. Modeling control objectives for business process compliance. volume 4714 LNCS, 2007.

[SKGL08] Stefan Sackmann, Martin Kähmer, Maike Gilliot, and Lutz Lowis. A classification model for automating compliance. *Proceedings - 10th IEEE Joint Conference on E-Commerce Technology and the 5th Enterprise Computing, E-Commerce and E-Services, CEC 2008 and EEE 2008*, pages 79–86, 2008.

[SSSJ18] Christian Sturm, Jonas Szalanczi, Stefan Schönig, and Stefan Jablonski. A lean architecture for blockchain based decentralized process execution. *Lecture Notes in Business Information Processing*, 342:361–373, 9 2018.

[SSSJ19] Christian Sturm, Jonas Scalanczi, Stefan Schönig, and Stefan Jablonski. A blockchain-based and resource-aware process execution engine. *Future Generation Computer Systems*, 100:19–34, 11 2019.

[SW22] Fabian Stiehle and Ingo Weber. Blockchain for business process enactment: A taxonomy and systematic literature review. *Lecture Notes in Business Information Processing*, 459 LNBIP:5–20, 2022.

[Swa15] Melanie Swan. *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.", 2015.

[Sza96] Nick Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 16:50–53, 1996.

[Tak08] Tsukasa Takemura. Formal semantics and verification of bpmn transaction and compensation. *Proceedings of the 3rd IEEE Asia-Pacific Services Computing Conference, APSCC 2008*, pages 284–290, 2008.

[TJFO21] Kyle Thomas, Daven Jones, Andreas Freund, and Anais Ofranc. Baseline API and Data Model Version 1.0. `https://github.com/eea-oasis/baseline-standard/blob/main/api/baseline-api-v1.0-psd01.md`, 09 2021. (Accessed on 2022-04-22).

[TLW18] An Binh Tran, Qinghua Lu, and Ingo Weber. Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management. volume 2196, pages 56–60. CEUR-WS, 2018.

[TXW+17] An Binh Tran, Xiwei Xu, Ingo Weber, Mark Staples, and Paul Rimba. Regerator: a registry generator for blockchain. In *CAiSE-Forum-DC*, pages 81–88, 2017.

[VXBP19] Wattana Viriyasitavat, Li Da Xu, Zhuming Bi, and Vitara Pungpapong. Blockchain and internet of things for modern business process in digital economy - the state of the art. *IEEE Transactions on Computational Social Systems*, 6:1420–1432, 12 2019.

[W3C22a] W3C. Decentralized identifiers (dids) v1.0. `https://www.w3.org/TR/did-core/`, 07 2022. (Accessed on 2023-08-25).

[W3C22b] W3C. Verifiable credentials data model v1.1. `https://www.w3.org/TR/vc-data-model/`, 03 2022. (Accessed on 2023-08-25).

[Wag02] Gerd Wagner. How to design a general rule markup language? In Robert Tolksdorf and Rainer Eckstein, editors, *XML Technologien für das Semantic Web – XSW 2002*, pages 19–37, Bonn, 2002. Gesellschaft für Informatik e.V.

[Wes19] Mathias Weske. Business process management. 2019.

[WG18] Karl Wust and Arthur Gervais. Do you need a blockchain? *Proceedings - 2018 Crypto Valley Conference on Blockchain Technology, CVCBT 2018*, pages 45–54, 11 2018.

[WHH+19] Wenbo Wang, Dinh Thai Hoang, Peizhao Hu, Zehui Xiong, Dusit Niyato, Ping Wang, Yonggang Wen, and Dong In Kim. A survey on consensus mechanisms and mining strategy management in blockchain networks. *IEEE Access*, 7, 2019.

[Woc] Jeremy Wadhams and other contributors. Jsonlogic. `https://jsonlogic.com/`. (Accessed on 2023-07-15).

[Woo22] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. 2 2022.

[WXR+16] Ingo Weber, Xiwei Xu, Régis Riveret, Guido Governatori, Alexander Ponomarev, and Jan Mendling. Untrusted business process monitoring and execution using blockchain. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9850 LNCS:329–347, 2016.