Informatics

Distributed off-chain storage for inter-organizational Business Process Execution

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering/Internet Computing

eingereicht von

Alexander Navratil, BSc.
Matrikelnummer 11776829

an der Fakultät für Informatik

der Technischen Universität Wien

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

Wien, 27. April 2023

Alexander Navratil   Wolfgang Kastner

TU WIEN Informatics

Distributed off-chain storage for inter-organizational Business Process Execution

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering/Internet Computing

by

Alexander Navratil, BSc.
Registration Number 11776829

to the Faculty of Informatics

at the TU Wien

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

Vienna, 27th April, 2023

Alexander Navratil | Wolfgang Kastner

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

Erklärung zur Verfassung der Arbeit

Alexander Navratil, 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, 27. April 2023

Alexander Navratil

Danksagung

Ich möchte mich allen voran bei meinem Betreuer Wolfgang Kastner bedanken, dass er mir die Möglichkeit gegeben hat, zum Zweck dieser Forschungsarbeit der Automation Systems Group an der TU Wien beizutreten. Dadurch konnte ich wertvolle Erfahrung zur akademischen Forschungsarbeit aus erster Hand sammeln. Weiters möchte ich mich bei Thomas Preindl und Martin Kjäer für die unzähligen Ratschläge und wegweisenden Diskussionen bedanken. Ich konnte durch diese Arbeit nicht nur sehr viel thematisches Wissen akquirieren, sondern mir auch viele methodische Fertigkeiten und Soft Skills aneignen.

Acknowledgements

I would like to thank my advisor Wolfgang Kastner for allowing me to join the Automation Systems Group at TU Vienna for the purpose of this research work. Through this opportunity, I have gained valuable first-hand academic research experience. Furthermore, I would like to thank Thomas Preindl and Martin Kjäer for their countless advice and groundbreaking discussions. I was not only able to acquire lots of thematic knowledge, but also many methodological and soft skills.

Kurzfassung

Geschäftsprozesse, welche über die Unternehmensgrenzen hinweg gehen, sind heutzutage ein wichtiges Instrument von Unternehmen. Speziell bei Lieferketten-Prozessen, wie zum Beispiel dem Bestellen von Produkten von externen Herstellern, sind mehrere Unternehmen in den Geschäftsprozess involviert. Diese Unternehmen übernehmen die Verantwortung für spezifische Aufgaben, wie zum Beispiel die Spedition oder das Liefern gewisser Bauteile. Die große Anzahl an beteiligten Unternehmen macht den Prozess jedoch komplexer und intransparent. Aus diesen Gründen gibt es in solchen Geschäftsprozessen eine vertrauenswürdige Partei, welche im Falle von Streitigkeiten die Steuerung des Prozesses übernimmt.

Das Anstellen und Übergeben der Verantwortung über den Geschäftsprozess an diese vertrauenswürdige Partei setzt allerdings das Vertrauen in diese Partei voraus, welche von dieser auch negativ ausgenutzt werden könnte. Aus diesem Grund ist es durchaus sinnvoll, nach alternativen Ansätzen zu suchen, um Geschäftsprozesse ohne eine vertrauenswürdige Partei ausführen zu können und gleichzeitig das Vertrauen aller am Prozess beteiligten Parteien sicherzustellen. Ein interessanter alternativer Ansatz ist es nun, diese vertrauenswürdige Partei mithilfe der Eigenschaften von Blockchains zu ersetzen, um den Prozess transparenter und öffentlich verifizierbar zu machen.

In dieser Arbeit führen wir daher eine Anforderungsanalyse durch, um wichtige Anforderungen für das dezentrale Ausführen von unternehmensübergreifenden Geschäftsprozessen zu identifizieren. Basierend auf den identifizierten Anforderungen, schlagen wir ein Konzept zur Modellierung und Ausführung lose-gekoppelter Geschäftsprozesse vor. Unser Konzept umfasst ein Workflow-Definitionsformat, Mechanismen zur Validierung der Eingabedaten und zur Implementierung von datenbasierter Prozess-Navigation.

Das vorgeschlagene Workflow-Definitionsformat erlaubt auch das Deklarieren eines externen Vertrages für die Zusammenarbeit mit anderen Geschäftspartnern. Wir verwenden dafür eine Peer-to-Peer Technologie, um eine Event-Log-Datenstruktur zu replizieren, welche auf einem sogenannten Conflict-Free Replicated Data Type (CRDT) basiert. Zusätzlich verwenden wir eine Blockchain, um gesendete Events festzuschreiben und damit auch die Überprüfbarkeit aller kollaborativer Aufgaben im Prozess sicherstellen. Weiters umfasst diese Arbeit die Implementierung eines Prototypen des vorgestellten Konzepts, welchen wir verwenden, um den realen Nutzen dieser Forschungsarbeit zu zeigen.

Abstract

Inter-organizational business processes are an essential component of organizations nowadays. Specifically, in supply chain scenarios, like ordering of products from an external manufacturer, many intermediate parties are involved in the process. These intermediate parties are responsible for tasks like shipping or supplying specific parts. The large number of involved parties makes the process complex and non-transparent. Thus, having a trusted intermediate party that mediates the process in case of a dispute is essential.

However, relying on a trusted intermediate party incurs a trust relationship, representing a potential single point of failure. Therefore, researching ways to eliminate the intermediate party while ensuring not to introduce a lack of trust is essential. An interesting approach to replace the trusted intermediate party is using a blockchain's properties to make the business process more transparent and publicly verifiable.

In this research work, we conduct a requirements analysis to identify essential requirements for executing inter-organizational business processes in a decentralized way. Based on the identified requirements, we propose a concept for modeling and executing inter-organizational business processes following a loosely coupled architecture. Our concept includes a workflow definition format, mechanisms for validating input data, and implementing data-based routing.

Additionally, our workflow definition format allows for defining an external contract for collaborating with other business partners. Therefore, we rely on peer-to-peer technology to replicate an event log data structure, which is based on a Conflict-Free Replicated Data Type (CRDT). We use a blockchain for committing exchanged events and enforcing the verifiability of collaborative tasks in the process. Our contribution also includes a prototypical implementation of the proposed concept, which shows the real-world utility of our research work.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

Business Process Management (BPM) is indispensable in companies nowadays. With the progressing globalization, companies need to collaborate even more to keep up with the market [1]. The increasing demands for collaboration lead to the introduction of inter-organizational BPM systems in companies. These inter-organizational BPM systems connect a company's internal business processes with other parties such as suppliers or manufacturers. In inter-organizational business processes, it is usual that there are multiple parties involved. All of them are interested in being successful, but that depends upon every involved business partner. Thus, it is natural that uncertainties might come up over time, leading to a lack of trust [2, 3]. This lack of trust in collaboration makes trusted intermediate parties a vital part of every inter-organizational business process. In case of a dispute, the intermediate party can take over the control and mediate the process [4].

In the past, several research works have put efforts into finding ways to mediate business processes without the need for an intermediary. Executing business processes without an intermediary reduces the dependency on a central entity. To get rid of the intermediate party without introducing a lack of trust, Weber et al. [5] suggest using Blockchain Technology (BCT). Due to the fact that the blockchain data structure is immutable, BCT is hard to compromise and it can replace a trusted intermediate party [6, 7].

Similarly, the Caterpillar [8] research project proposes a BPM solution with BCT by generating so-called smart contracts. Smart contracts are programs that are executed on a blockchain. The execution of smart contracts can be verified later on, thus allowing anyone with access to the BCT to verify the process. However, this also means that anyone can look up all the details consisting of input and output data. This form of verifiability is specifically problematic when the data is sensitive.

Sturm et al. [9, 10] evaluated the execution of business processes with BCT and proposed to use BCT only for its core properties, i.e., to guarantee the integrity of data and for time-stamped commitments. Instead, peer-to-peer networks with tailored cryptographic data structures have the potential to improve on existing BCT-based approaches.

Executing Business Processes fully on a blockchain comes with a few issues. On one hand, BCT aims to improve fail-safeness by decentralizing all processes. On the other hand, fully relying on a blockchain again introduces a dependency and a potential single point of failure, e.g., fixing an urgent bug in a deployed smart contract is harder than applying a fix to an application hosted on a centralized server. Additionally, preserving the privacy of sensitive business processes, when executing them solely on a blockchain, is hard to achieve. However, BCT contains many interesting concepts like decentralization, peer-to-peer networks or data structures. Therefore, in this research work, we analyze requirements regarding BPM, BCT and related topics and propose a concept for executing business processes in a decentralized way.

Based on existing decentralized storage networks, business processes can be executed asynchronously. Hence, two collaborating business partners do not need to be reachable at the same time. Instead, complementary concepts like mailboxing and off-chain communication take over the role of an intermediate storage node without compromising any trust properties. By using off-chain communication, we keep all communication off a blockchain. Additionally, cryptographic data structures can help to ensure a secure execution of business processes, i.e., all collaborating organizations need to reach a consistent state of their business process.

Collaboration in business processes specifically reaches limits with an increasing number of participating organizations. Moving centrally-controlled business processes into a more decentralized form brings the before-mentioned advantages. However, we are at the same time losing the overview of the global process spanning all participating organizations, because every organization is operating the process independently. Additionally, collaboration only happens at specific activities in the process and primarily only between two business partners, which makes it hard to get an overview of the overall business process. Overall, our proposed concept strives to take BCT as a component instead of as a fully independent system and consequently improve BPM by architecting a decentralized system composed of different mechanisms, data structures and algorithms.

1.2 Problem Relevance

Previous research work solely relied on BCT as an independent system. Following this system architecture brings along many drawbacks that were mentioned above. Depending on a single system introduces a single source of failure, although blockchains are often considered as being fail-safe without additional measures. Therefore, this research aims to explore existing research work, identify requirements and take centralized BPM into a decentralized direction.

As BPM is a complex problem in general, we are mostly focusing on Supply Chain Management (SCM) as a reference business process. SCM processes typically consist of a variable-sized chain of intermediary organizations like suppliers, manufacturers or shipping carriers. Throughout this complex process, data is exchanged between the participants. Based on the exchanged data, decisions need to be taken transparently as decisions can influence routes inside the business process.

Based on the defined aim, the following research questions are defined:

RQ1 What is the current state of the art for distributed off-chain storage technologies?

RQ2 Which requirements does a BPM system need to provide for the decentralized execution of inter-organizational processes?

RQ3 Which technologies and architecture can be leveraged to implement the defined requirements?

1.3 Method

This section describes the chosen methodological approach to gain the necessary background knowledge and related work, identify requirements and implement the proposed concept in the form of a prototype.

1.3.1 Design Science

The research is based on the Design Science research method by Hevner et al. [11]. Design Science, as the overall research method, employs different methods. The following sections explain the used methods in more detail.

1.3.1.1 Literature Review

A narrative literature review is used to build a knowledge base about the covered topics. It specifically serves to identify related work and state-of-the-art. The literature review covers theoretical information about distributed off-chain storage protocols and their internals like data structures and algorithms. Additionally, we explore and compare related work to incorporate the beneficial properties of BCT into this research. The primary sources for this task are previous scientific work or technical documentation. In more detail, a set of concepts and properties are reviewed for each identified distributed off-chain storage solution. Overall, the literature review answers the research question **RQ1**.

1.3.1.2 Requirements Engineering

Throughout the requirements engineering process, we study an inter-organizational SCM business process that assists to define the required properties of the final artifact.

Requirements engineering is used to set up a minimum set of requirements that a distributed off-chain storage solution has to fulfill to work in the defined environment of decentralized inter-organizational BPM. To conduct the requirements analysis, we first define the vision we want to achieve. The vision is based on previously identified literature and a reference business process. Based on the defined vision, we split the topic into different related aspects that help to set boundaries and focus on the vision. This structured approach later helps to identify important requirements and to evaluate the result.

Finally, to evaluate the results of this research, a prototypical implementation is built using software engineering principles. The prototype uses a distributed network that consists of decentralized peers for exchanging information. We conduct an empirical evaluation based on qualitative properties. By cross-checking the proposed concept and the implemented prototype against the identified requirements, we evaluate the practical utility of the final artifact. Throughout this research, and specifically during the evaluation, we identify limitations and future work. Overall, the requirements engineering phase answers the research question **RQ2**.

1.3.1.3 Software Engineering

Using the defined requirements and the proposed concept, we conduct a prototypical implementation to show the practical utility. Our implemented prototype incorporates several software design patterns and components. We aim to use different design patterns, like the strategy design pattern by Gamma et al. [12] and inversion of control, to allow for extending the prototype with exchangeable components. Additionally, our prototype is split into loosely coupled modules that interact via clearly defined interfaces. These interfaces assist in having a transparent event and data flow. The implementation of the prototype answers the research question **RQ3**.

1.4 Structure of the Work

The remainder of this work is structured as follows: Chapter 2 gives necessary background information for the following chapters. Chapter 3 continues with a literature review on off-chain storage technologies. Next, we incorporate the result of the literature review in Chapter 4 to conduct a requirements analysis. Based on the identified requirements, Chapter 5 presents our proposed concept and describes the prototypical implementation. The resulting requirements, the concept, and the prototype get evaluated in Chapter 6, before concluding the research work in Chapter 7.

CHAPTER 2

Background

In this chapter, we introduce necessary concepts and technologies, which are required for the upcoming chapters. The chapter also covers basic concepts of BPM followed by necessary background knowledge on data structures required for the remaining chapters.

2.1 Blockchain Technology in Business Process Management

To get an overview of existing research on BCT in BPM, we conducted a narrative literature review [13]. This literature review targets previous research work about inter-organizational business processes on the blockchain with a focus on different storage approaches. More details on how the literature review was conducted can be found in Chapter 3 and Appendix A.

2.1.1 Definition of Business Process Management

Business Process Management covers the management of internal and external processes in an organization. Internal Processes satisfy an inner need, like assuring the quality of a factory process. On the other hand, external processes interact with other business parties, like ordering a product from a supplier. Both types of processes consist of events, activities and decisions that are chained together and belong or serve some *actor* (e.g., humans, organizations), *physical object* (e.g., equipment, material, paper document) or *informational object* (e.g., electronic records or electronic documents).

Figure 2.1 shows an inter-organizational SCM process using the Business Process Model and Notation (BPMN) standard. The BPMN is used for modeling and visually describing business processes. The BPMN standard models activities as rounded rectangles, decision points as diamond shapes, and events as circles. Events signal atomic happenings, like receiving an order from a customer. Activities usually consist of multiple tasks that need to be done to fulfill the activity. Simple activities consisting of a single task are just called tasks. Decision points (gateways) are used to split the process into multiple paths. In the end, a process has either a positive or negative outcome that depends on whether the actual process instance delivered value to the actors. [14, 9]

Processes are often visually grouped in so-called lanes representing the participants or organizational unit (e.g., departments). Lanes can, in turn, be grouped into pools that represent whole organizational units. Pools may also represent external organizational units. A process that groups together multiple pools from different organizations is called inter-organizational, as opposed to intra-organizational processes.

Business Processes can be categorized into process orchestrations and choreographies. Orchestrations are centralized business processes that are usually internal, as they always need a central mediator that controls the process. Orchestrations are relatively simple to manage, but the downside of orchestrations is that for large processes, centralized coordination is prone to becoming a bottleneck [7]. On the other hand, choreography processes do not have a central controlling component. Instead, the participants operate their activities independently and communicate with the business partners via messages. Due to this decentralized behavior, choreography processes are used to describe inter-organizational business processes.

Figure 2.1 shows a choreography business process (illustrated using BPMN 1.1) indicated by the dashed connections with empty arrows between the lanes. The event circles that contain envelopes signal that they are waiting to receive messages. [15, 7] In this thesis, we primarily focus on orchestration business processes.

Choreography business processes can connect processes from different organizations. Agreeing on standard message interfaces to connect organizations is still an issue as their internal processes are quite complex. In addition to that, a poorly designed interaction might cause deadlocks or similar problems. For standard processes like SCM or financial transactions, there are pre-defined collaboration protocols like RosettaNet [1] and SWIFTNet [2]. For non-standard processes, the agreement on a common interface remains a big problem. This problem gets even more significant when considering the evolution of business processes and their interactions. As choreographies do not have a central controlling component, no organization can ever be aware of the global state to mediate disputes of the collaborating organizations. This situation leads to a trust issue between the organizations. [15, 17, 2, 3, 10]

A solution to this issue would be an intermediate party, as Chapter 1 introduced. Instead of having to trust this intermediate party, Weber et al. [5] suggest using BCT. The blockchain data structure in BCT is immutable, and every interaction between the

[1] `https://docs.oracle.com/cd/B10463_01/integrate.904/b12121/b2bstandards.htm`

[2] `https://www.swift.com/our-solutions/interfaces-and-integration/swiftnet-link`

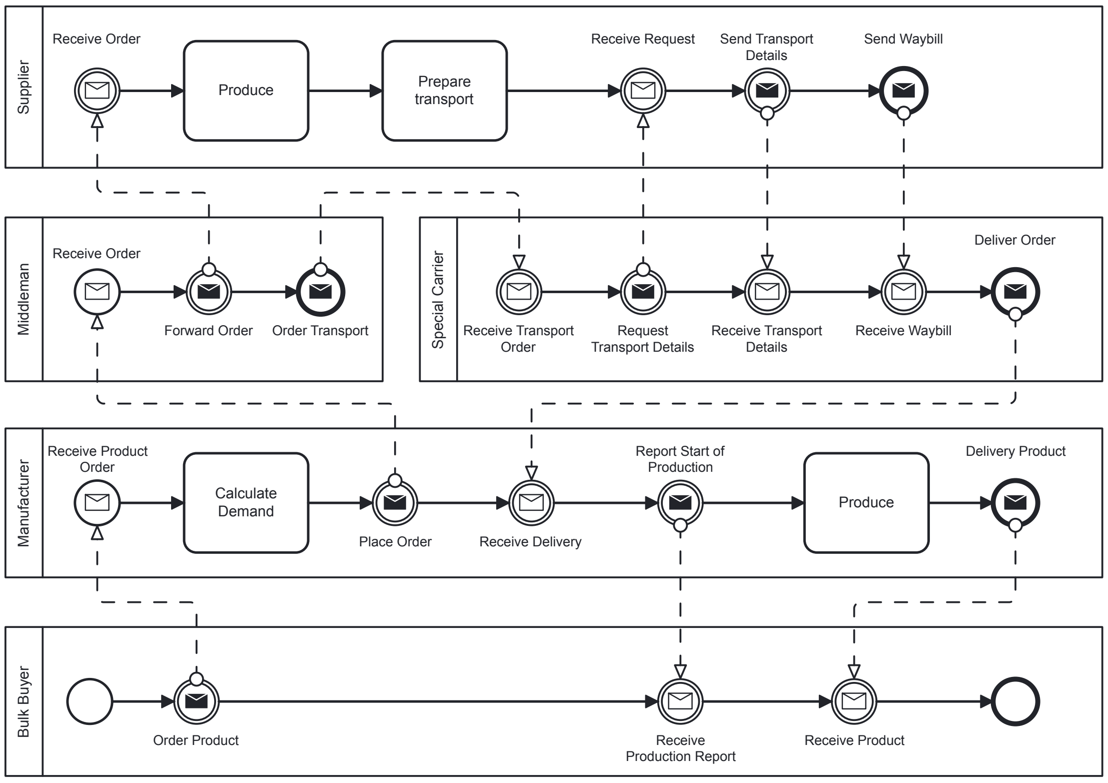


Figure 2.1: SCM Business Process from [5] and [16] in BPMN notation.

choreography business processes is recorded. Therefore, all pre-defined rules in the process are publicly verifiable. Through smart contracts, even the enforcement of penalties is possible. In a typical SCM scenario, a delayed delivery could be compensated with a pre-defined discount. [6, 7, 18]

BCT as a mediator in business processes can also serve other purposes. Weber et al. [5] describe the implementation of a choreography monitor which observes all message exchanges between the organizations. The resulting audit trail allows to analyze and optimize the business process. In addition, the authors introduced so-called triggers that keep the organizations' local Business Process Management System (BPMS) and the business process on the blockchain in sync. Furthermore, the triggers can move any attached payload to a separate storage because storing data on the blockchain is costly. The attached payload gets then replaced by a URI that links to the payload in the separate storage. Since this separate storage is placed outside of the blockchain, it is called off-chain storage. The trigger can additionally take care of encrypting sensitive payloads.

Alternatively to triggers, Carminati et al. [6] use oracles, which are off-chain components that communicate with external sources and send data to the blockchain. From a functionality point of view, oracles are comparable to triggers, but oracles are typically not in possession of the organization. Therefore, the usage of oracles introduces another trust issue that one can avoid by using triggers. In addition, Carminati et al. [6] further brought attention to security issues that need to be addressed with BCT. BCT only guarantees data integrity for data included in blockchain transactions, but keeping all relevant data on the blockchain breaks the confidentiality property, specifically for sensitive data. The data confidentiality issue further splits up into keeping transaction data confidential (i.e., transaction confidentiality) and keeping data in oracles confidential (i.e., Oracle data confidentiality). Additionally, the whole business process might be sensitive. The Lightstreams[3] project follows a promising approach to tackle data confidentiality issues, by using the Permissioned Blocks protocol[4] which combines a confidential data store with BCT to selectively grant access to transaction data.

Mendling et al. [19] even predict that research around BPM choreographies might get influenced by BCT. Based on that assumption, Ladleif et al. [20] proposed extensions to BPMN choreography diagrams like global data objects or sub-choreographies to keep data objects private to a specified set of participants.

López-Pintado et al. [8] implemented a BPMS for the Ethereum blockchain with a compiled approach. Their prototype takes a business process in BPMN format as input and compiles them to smart contracts. After deploying them on the blockchain, a client can execute the given business process on the blockchain. Also, this concept faces challenges such as business processes are typically not fixed and change over time, which makes the deployed smart contracts obsolete, and new ones must get deployed, which is a costly task in the long term. Therefore, as opposed to the compiled approach, López-Pintado et al. [21] extended this prototype in another research work with an interpreted approach. This approach works by deploying a generic interpreter smart contract that uses a business process in BPMN format as input and interprets that during the runtime. When a business process gets updated, an administrator must update the deployed model and not the entire contract. A similar approach, which focuses more on the verification than the execution, was followed by Prybila et al. [7]. They used the Bitcoin blockchain to store cryptographic hashes of local BPMS executions to trace and verify the business process.

Based on the work of López-Pintado et al. [8], another research work by Di Ciccio et al. [22] evaluated the traceability of the Caterpillar BPMS. Their approach reverse-engineered the transactions from the deployed BPM smart contracts to get insights into the executed operations and the supplied input data. However, this approach only works because all the data on the Ethereum blockchain is public, which is not suitable for most organizations.

Köpke et al. [23] compared different patterns to execute choreography processes on Ethereum confidentially. Since choreographies mainly rely on message exchanges, they tried to encrypt messages or keep them off-chain. On top of that, selective access control

[3]https://docs.lightstreams.network/
[4]https://github.com/lightstreams-network/permissioned-blocks

and encryption mechanism need to be designed, as not every participating organization needs to have insights on all exchanged messages [23, 24]. Solving confidentiality and privity issues needs more tailored solutions than using state-of-the-art blockchains like Ethereum. Previous research work focusing on data confidentiality issues or privity proposes that it needs tailored solutions to efficiently manage sensitive inter-organizational business processes with BCT [23, 9, 10].

2.2 Merkle trees

Merkle trees are essential to almost all off-chain storage technologies to encode data efficiently. They are similar to binary trees in that they are a tree-like data structure, typically with a branching factor of two. A branching factor of two means that every node has at most two children. They differentiate from simple binary trees in a few points. Merkle trees only store the actual payload in the leaf nodes, and Merkle trees can only be built bottom-up. All intermediate nodes, including the root node, authenticate their child nodes with a cryptographic hash function.

Figure 2.2 depicts the theoretical construction of a Merkle tree, which stores a file split into chunks. Any change in a leaf node leads to a change in the leaf node's hash, which implicitly propagates up to the root node. Therefore, committing to the Merkle root hash ensures that any data change can be detected by re-building the Merkle tree and comparing the root hashes.

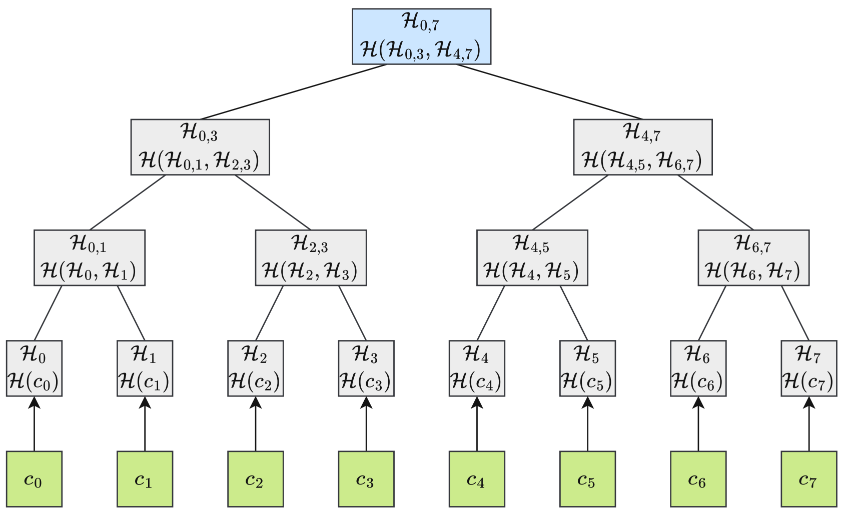

$\mathcal{H}_{0,7}$
$\mathcal{H}(\mathcal{H}_{0,3}, \mathcal{H}_{4,7})$
$\mathcal{H}_{0,3}$
$\mathcal{H}(\mathcal{H}_{0,1}, \mathcal{H}_{2,3})$
$\mathcal{H}_{4,7}$
$\mathcal{H}(\mathcal{H}_{4,5}, \mathcal{H}_{6,7})$
$\mathcal{H}_{0,1}$
$\mathcal{H}(\mathcal{H}_0, \mathcal{H}_1)$
$\mathcal{H}_{2,3}$
$\mathcal{H}(\mathcal{H}_2, \mathcal{H}_3)$
$\mathcal{H}_{4,5}$
$\mathcal{H}(\mathcal{H}_4, \mathcal{H}_5)$
$\mathcal{H}_{6,7}$
$\mathcal{H}(\mathcal{H}_6, \mathcal{H}_7)$
$\mathcal{H}_0$ $\mathcal{H}(c_0)$
$\mathcal{H}_1$ $\mathcal{H}(c_1)$
$\mathcal{H}_2$ $\mathcal{H}(c_2)$
$\mathcal{H}_3$ $\mathcal{H}(c_3)$
$\mathcal{H}_4$ $\mathcal{H}(c_4)$
$\mathcal{H}_5$ $\mathcal{H}(c_5)$
$\mathcal{H}_6$ $\mathcal{H}(c_6)$
$\mathcal{H}_7$ $\mathcal{H}(c_7)$
c_0 c_1 c_2 c_3 c_4 c_5 c_6 c_7

Figure 2.2: Theoretical construction of a Merkle tree that stores chunks of a file

Additionally, proving that a specific leaf node is included in a Merkle tree with an inclusion proof is a crucial property of Merkle trees. For example, to prove the inclusion of the chunk c_4, a proofer needs to provide $\mathcal{H}_5$, $\mathcal{H}_{6,7}$ and $\mathcal{H}_{0,3}$. Given these values, a verifier can verify the inclusion of the chunk c_4 by only knowing the root hash of the

Merkle tree. The verification works by hashing the chunk c_4 and re-building the Merkle tree up to its root node, which can then be verified by comparing the root hash. In case the root hash is equal to the previously committed root hash, we have a proof that the chunk c_4 is included in the Merkle tree. Inclusion proofs are a widespread method in BCT to prove the inclusion of blockchain transactions in a block. [25] The practical example in Appendix B.2 gives more details on Merkle tree inclusion proofs.

2.3 Merkle DAGs

Instead of encoding information with Merkle trees, data can be structured with Directed Acyclic Graphs (DAGs). Merkle DAGs follow a similar concept as Merkle trees in that they authenticate the stored information with cryptographic hashes. The data can be stored throughout the graph, not just in leaf nodes, which makes it possible to store metadata within the data structure. Due to the linkage between the nodes with cryptographic hashes, the DAG can again only be built bottom-up. Additionally, the hashes ensure that there cannot be any cycle in the graph. For creating a cycle in the graph, the hashes of a higher-level node would need to be known when creating the lower-level node. But since predicting hashes is cryptographically impossible, Merkle DAGs are acyclic by design. Like Merkle trees, a change in a referenced file, metadata, or directory propagates to the root node. Thus, any change to the data structure leads to a changed Merkle root hash. A practical example of Merkle DAGs are mutable InterPlanetary File System (IPFS) directories, where whole directory structures are encoded as DAGs. The usage of Merkle DAGs in data networks, as well as the storage of directories in IPFS are explained in more detail in the Sections 3.4.3 and 3.4.5.

Figure 2.3 shows a Merkle DAG that visualizes a simplified data structure of the Git versioning system. This Merkle DAG example contains three object types: Commits, Trees, and Blobs. The initial commit on the right contains a reference to a tree with three blobs: `file1.txt`, `file2.txt`, `file3.txt`. The user then deletes `file3.txt` and adds `file4.txt`. After committing these changes, a new commit object is created in the graph that references the previous commit. The resulting commit object (middle) references a new tree that still references the remaining files `file1.txt` and `file2.txt` and adds `file4.txt` (highlighted in blue). The last commit (on the left) introduces a new directory structure. The root tree of this commit references a new file `readme.md` and a tree that references the existing `file4.txt`. The remaining files from the previous commits (i.e., `file1.txt`, `file2.txt` and `file3.txt`) were deleted in that commit. Along with the objects, Git stores metadata like commit messages or authors of changes. The Merkle DAG is ideally suited for such data structures, as any change is tracked directly via the hash-linked data structure. [26]

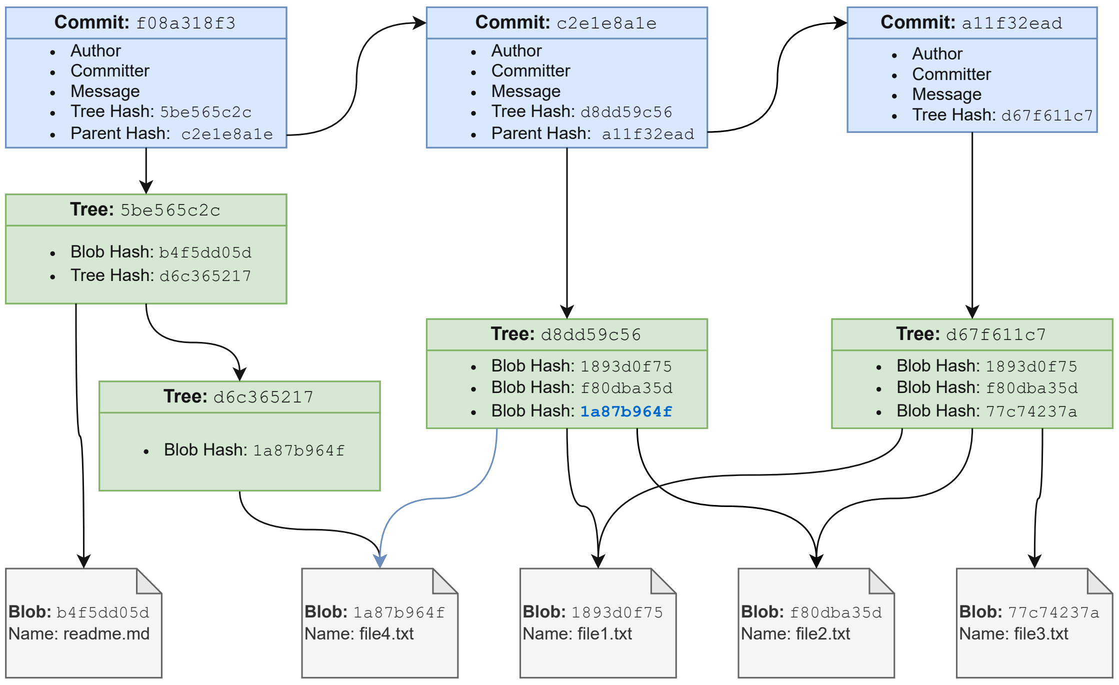


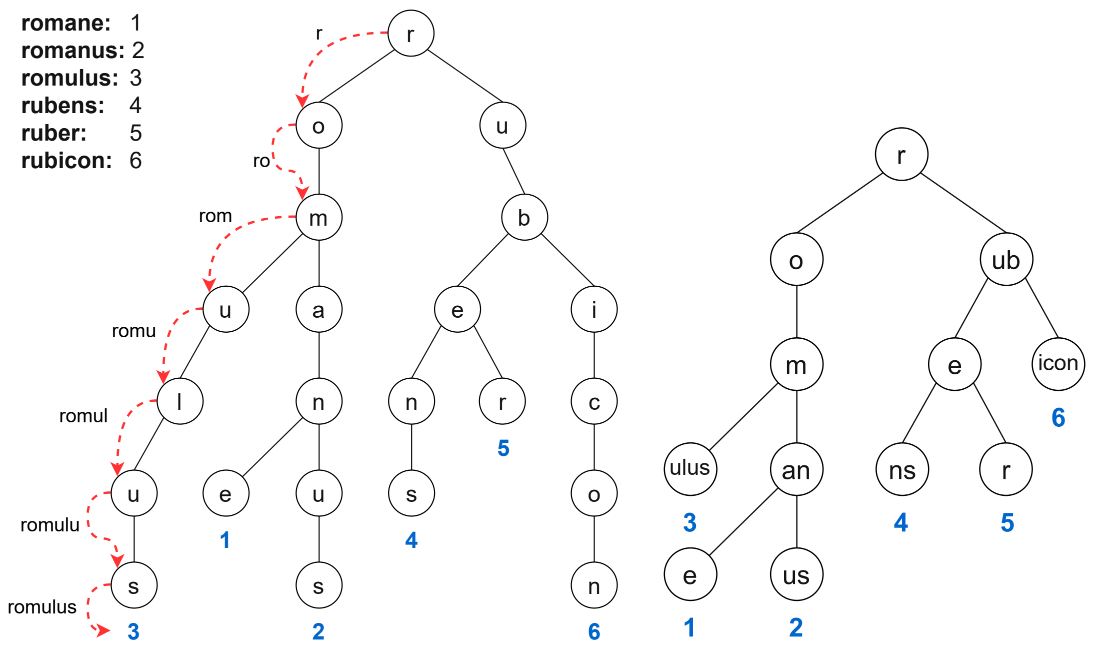
Figure 2.3: Simplified Merkle DAG showing a Git versioning system data structure, inspired by [26]

2.4 Tries

A Trie is a tree data structure for storing key-value mappings. The name originates from the word *retrieval* because Tries are optimized for retrieving information based on prefixes. Tries are different from trees in that they encode the information along a path over multiple nodes. The keys of the key-value pairs are split by character, and each character forms a node. Keys with the same prefix share the same path for that particular prefix, which makes retrieving key-value pairs with the same prefix a fast operation.

Figure 2.4a illustrates the principle of the Trie data structure. The figure shows the simple form of a Trie, where every character in the key forms a separate node, although a more compact form would be possible. The red arrows show the retrieval of the key *romulus* with a depth-first search. Patricia Merkle Tries are a more compact form of Tries, where unambiguous parts of the graph are stored in a single node instead of separate nodes for each character. The compact storage makes graph operations more efficient, as the depth is typically lower. Specifically for unambiguous keys, the graph traversal is faster because there are fewer nodes to access. The same principles as in Merkle trees and Merkle DAGs for linking nodes also apply to Patricia Merkle Tries. Thus, nodes are linked via a cryptographic hash function, which authenticates the content.

Figure 2.4b shows a Patricia Merkle Trie without hashes for simplicity reasons. The retrieval of a key with many unambiguous parts, like *rubicon*, is accessible by reading just three instead of seven nodes.



(a) A Trie data structure based on [27]. The key-value mapping on the top left is encoded in the Trie, with the values being highlighted in blue.

(b) A Patricia Merkle Trie based on the expanded Trie from Figure 2.4a. The hash linkage has been omitted for simplicity but applies equally as in Merkle DAGs from Section 2.3.

Figure 2.4: Comparison of a Trie and a Patricia Merkle Trie data structure

Patricia Merkle Tries are used in Ethereum for storing states, transactions, receipts, or even data of smart contracts. The peer-to-peer data network Swarm uses Patricia Merkle Tries for efficiently storing manifest documents, which are essentially just key-value pairs. More details on storing Swarm manifests in Patricia Merkle Tries can be found in Section 3.4.5. [28, 29]

2.5 Distributed Hash Tables

Distributed Hash Tables (DHTs) are the heart of many distributed data networks, like BitTorrent, IPFS or Swarm. A DHT is a key-value storage that is distributed among a network of collaborating nodes. None of the nodes stores all key-value pairs, they are rather distributed and queried on demand. Furthermore, there is no central registry knowing all keys and their location, as that is the responsibility of the routing algorithm to find the requested key-value pair in the network. Therefore, a DHT's main functionality is based on employing a protocol for finding the route to any key — a Key-Based Routing

(KBR) service [30]. The most popular variant of DHTs is called Kademlia, and apart from minor differences, all data networks presented in this work are based on it. Thus, this work solely concentrates on the Kademlia DHT. [31]

2.5.1 Distance Metric

Kademlia is a DHT proposed by Maymounkov et al. [31] which uses an XOR-based distance metric as its core mechanism. Every peer and every key in the DHT is addressed by its hash, thus every addressable identifier uses the same key space. While in the original paper, the authors use a 160-bit hash, almost all known data networks use 256-bit hashes. The distance metric is important, as every key in the DHT is stored on peers with an address that is close to the hash of the key. In the context of the Kademlia DHT, Closeness is defined as the binary XOR between two addresses. Closeness means that two addresses are logically close, not geographically. For example, the XOR distance between a node with `110` as hash and a node with `010` as hash is:

$$distance(110, 010) = 100 = 4 \text{ in decimal}$$

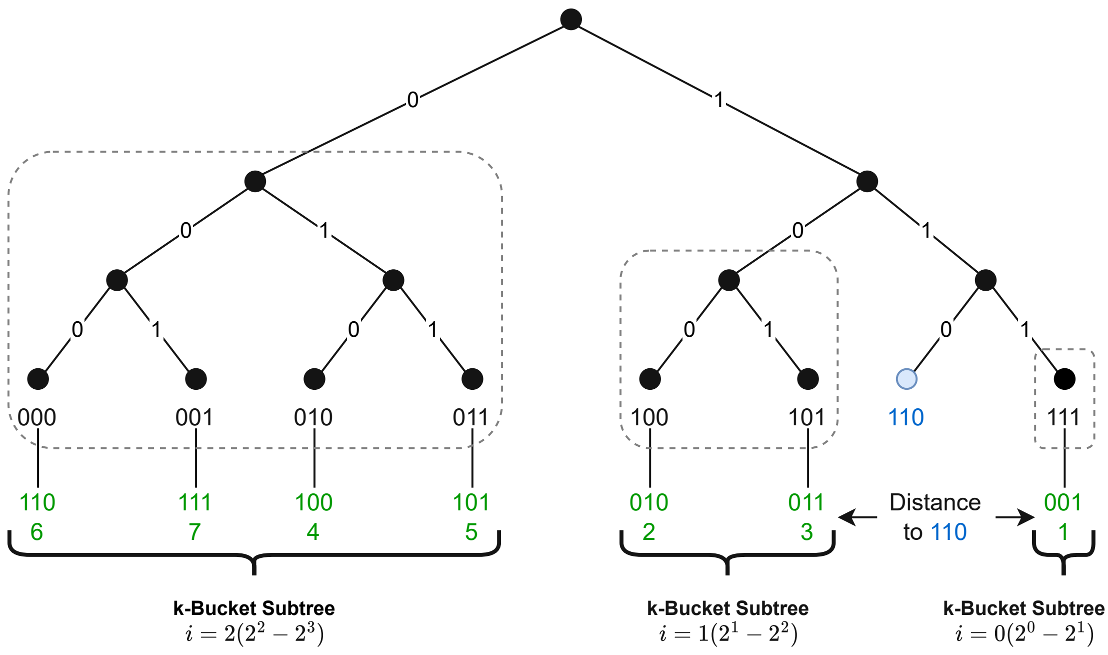


Figure 2.5: Shows a Kademlia network as a binary tree with each peer being a leaf node in the tree. The XOR distance metric is shown relative to the node `110`. The dashed rectangles group all peers in a subtree that could belong to the corresponding k-bucket. The node `110` must have a connection to at least one node in every k-bucket, otherwise, Kademlia does not work efficiently. This visualization is inspired by the Kademlia binary tree in [31].

Figure 2.5 represents a Kademlia network as a binary tree. The leaf nodes represent a peer (although not all peers must exist) and the path from the root node to a peer builds up the peer's binary hash. For instance, when following the path `1` $\rightarrow$ `1` $\rightarrow$ `0`, we arrive at node `110`. The numbers in green below each peer show the XOR distance in binary and decimal relative to the peer `110`. [31]

2.5.2 Routing Table

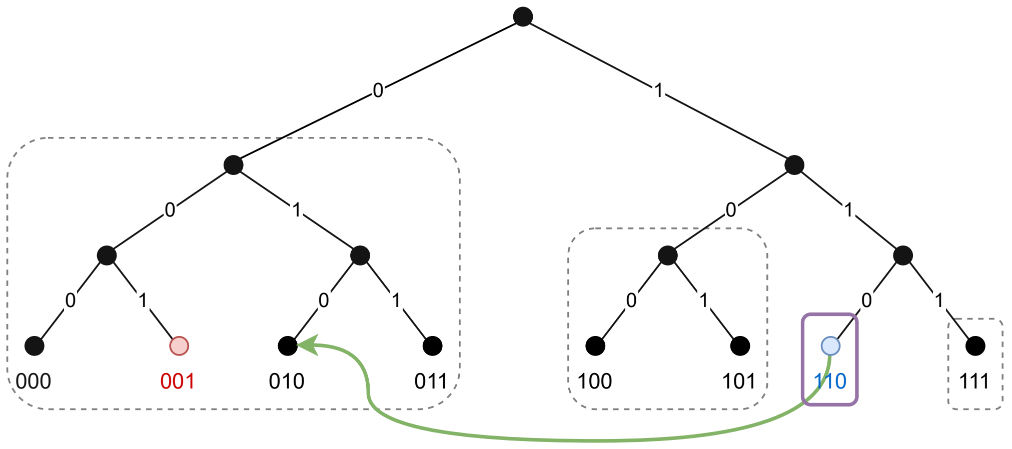
For Kademlia's routing to work correctly, each peer maintains a routing table. For a 160-bit key space, this routing table consists of 160 so-called k-buckets. For each $0 \leq i < 160$, the i^{th} k-bucket maintains a list of k nodes with an XOR distance between 2^i and 2^{i+1} relative to the node hosting the routing table. A k-bucket contains at most k references to other nodes, with k having typically a value of 20. The k-bucket size was chosen by the researchers such that the probability is high enough so that k nodes will remain online for another hour. This is because, from a statistical point of view, nodes that are already online for a longer time are more likely to remain online. Each k-bucket entry consists of a triple containing the IP address, UDP port and the node ID of the referenced node.

Figure 2.5 depicts these potential k-buckets from the perspective of node `110`. In this example, we have a 3-bit key space and thus three k-buckets. The node `110` must hold a reference to at least one node in each k-bucket subtree. As long as a Kademlia network complies with this rule, it is possible to find any node in logarithmically many steps. Additionally, due to the XOR metric, all lookup paths for any key-value pair will converge toward the target node. Therefore, caching key-value pairs on nodes that are close to the target node sustainably improves the performance of lookups.

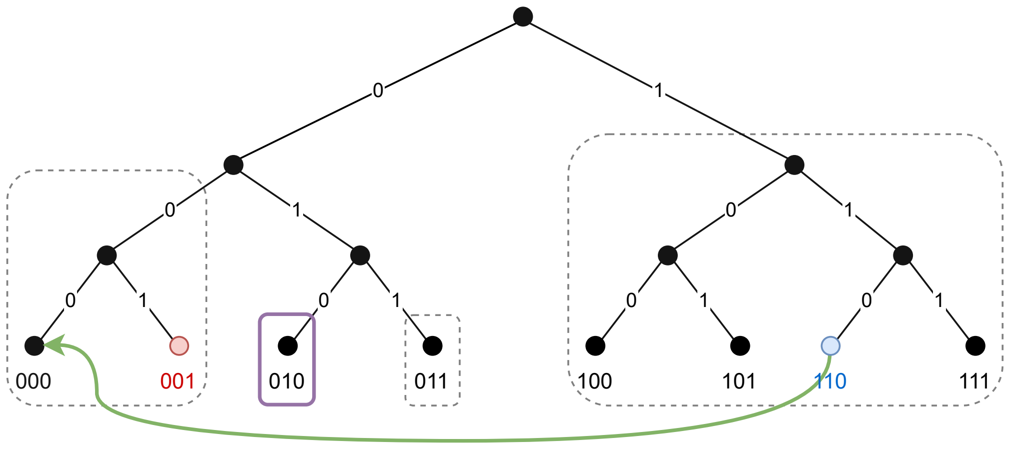
Figure 2.6 visually describes how the process of finding the target node `001` starting from the source node `110` works. With each step of the algorithm, the source node queries a node from a k-bucket subtree that is closer to the target node.

For a node to join the Kademlia network, it must know at least one node that is already part of the network. IPFS maintains so-called bootstrap nodes that are operated by the company behind IPFS. This might make a more centralized impression as these bootstrap nodes could be a single point of failure. It has to be mentioned, though, that it is up to the joining node to know another node that is already participating in the network and the bootstrap nodes are just improving the usability of joining the network. Upon connecting to a bootstrap node, the joining node performs a lookup of its peer identifier. The reason for this mechanism is, that any node, that is approached by another node must consider adding the contacting node to its routing table. Thus, by looking up its peer identifier, the node advertises its existence to other nodes.

As we have seen in Figure 2.6, the lookup path converges towards the target node. Therefore, when a node looks up itself, close neighboring nodes are getting identified and a connection to those neighboring nodes is established. The joining node gets added in the appropriate k-bucket of the neighbor nodes as soon as the respective k-bucket is not full. In case the k-bucket is full, the node with the least-recent communication gets pinged. When the node does not answer, it gets evicted from the k-bucket and the joining node gets added. Otherwise, the joining node gets discarded and the node that is already in the k-bucket is kept. This mechanism also reflects that nodes that we know of for a longer time, will most likely remain online in the future, which makes the Kademlia network more stable against peer churn. [31]



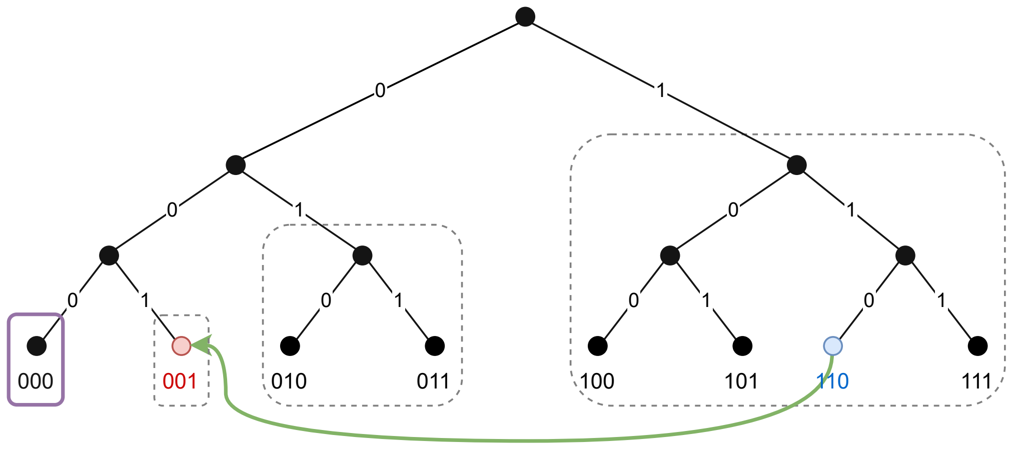
(a) **Step 1:** The source node queries a node from the k-bucket that is closest to the target. Here we assume, that the source node does not directly know the target node, but another node (`010`) that is in the closest k-bucket subtree. Node `010` will respond with a node that is even closer to the target node.



(b) **Step 2:** This figure shows the perspective of node `010`. In the previous step, node `010` returned node `000` as a closer node. Therefore, in this step, the source node queries the node `000` and asks for an even closer node.

2.5.3 Storing a Key-Value Pair

Storing a key-value pair is similar to looking up a node with a specific identifier. Remember that all identifiers are hashes from the same key space. Instead of looking for a specific



(c) **Step 3:** This figure shows the perspective of node `000`. As node `000` already returned the target node `001`, the process of looking up the target node is finished. In this step, the source node can already communicate with the target node and ask for the value of some key or store a value.

Figure 2.6: Kademlia's process of looking up a previously unknown node. With every step, the source node gets closer to the target node, based on the XOR distance metric. Each step shows the k-bucket subtrees from the perspective of the node enclosed in the purple rectangle. The green arrow shows which node is queried in this step. The node marked in red is the target node, which is going to be looked up eventually. The Kademlia binary trees are inspired by the visualization in [31].

peer, we are looking for a node with an identifier that is close to the hash of the key. Therefore, we are again looking for the k closest peers and upon finding them, we send them a dedicated Remote Procedure Call (RPC) to store the key-value pair.

Figure 2.7 shows an example of storing the key `key`. We assume that the binary hash of the key is `001`, $k = 3$ and the concurrency parameter $\alpha = 2$. The concurrency parameter defines how many peers are queried in each iteration. The algorithm for finding the k closest nodes works as follows:

1. In the first round, we pick the closest α nodes and ask them for nodes that are close to our key hash. Peer `101` will return Peer `010` with a distance of 3 and Peer `011` will return Peer `100` with a distance of 5.
2. In the next round, we pick again the closest α nodes that we know of, including the recently identified nodes from the previous iteration. Thus, the α closest nodes are Peer `010` and Peer `011`. After asking them for nodes that are close to the key hash, we get back Peer `000`.
3. As we have already queried enough nodes and received k responses with closer nodes, we pick these k nodes and send them the RPC to store our key-value pair.

In this example, we pick the nodes `000`, `011` and `010` with a distance of 1, 2 and 3, respectively.

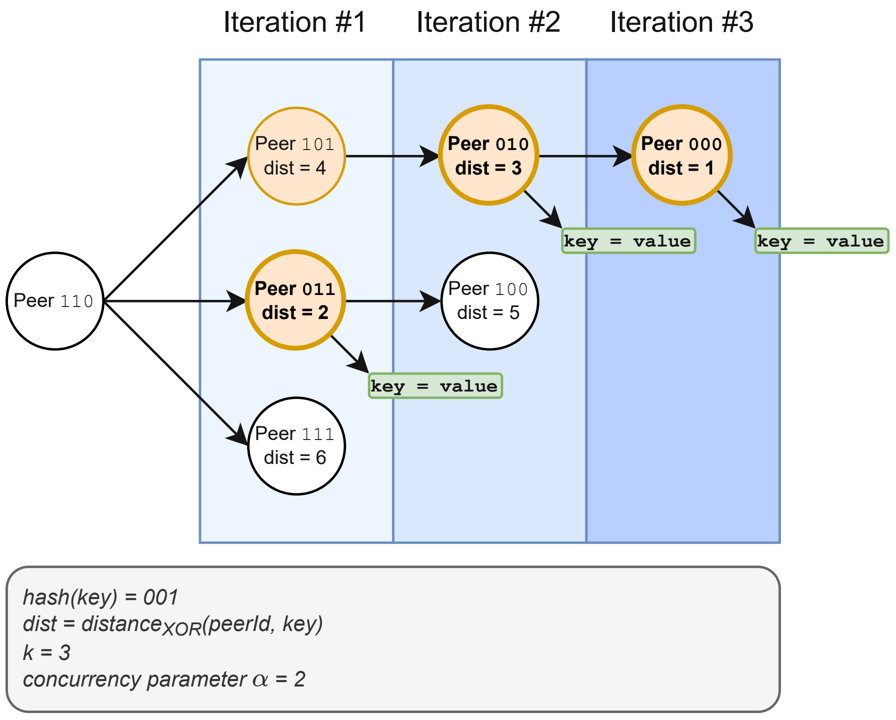


Figure 2.7: This figure depicts the value retrieval process in a Kademlia-based network. The visualization is inspired by [32].

2.5.4 Retrieving a Key-Value Pair

Retrieving a key-value pair works similarly to the storage procedure, which was described previously. We need to find nodes that are close to the hash of the key using the XOR distance metric. In practice, different termination mechanisms exist, because there are some edge cases where key-value pairs are mutable and thus multiple versions could exist. In that case, a lookup algorithm might query more than k nodes in total. A typical mechanism is based on quorums where the algorithm tries to find enough equal answers such that it has enough evidence that the returned answer is the current one [32]. The mechanism presented in the original Kademlia paper is much simpler and terminates when stumbling upon the first occurrence of the key-value pair.

Instead of sending RPCs to identify close nodes, the requesting node sends a RPC to query for the key directly. In case a node does not know the key but knows a closer node, it returns the closer node so the requesting node can ask there in the upcoming iteration. [31]

2.5.5 Kademlia Extensions

Over the years, many researchers analyzed the concepts of Kademlia, security, and suitability for different use cases. Kademlia became the de-facto standard DHT for peer-to-peer networks, although it also comes with drawbacks, like ignoring the geographical topology of the underlying network. The rest of this section, we describe extensions to Kademlia that improve the investigated aspects.

2.5.5.1 S/Kademlia

The research work of Baumgart et al. [33] focuses on the security of the Kademlia protocol. Previous research has identified several conceptual security issues in Kademlia. The following list briefly summarizes the most important security findings together with possible countermeasures proposed by Baumgart et al. [33]:

- **Eclipse attack:** During an Eclipse attack, an attacker creates adversarial nodes that cut off a victim node from the network. This works by having all messages from the victim node routed through adversarial nodes that can censor or manipulate all messages.

  A countermeasure would be to restrict the generation of arbitrary peer identifiers, as routing tables could be easily manipulated with specially forged peer identifiers. Since Kademlia prefers long-lived nodes when maintaining its routing table, the attack is more likely to succeed during the bootstrap process. Thus, a node is more vulnerable to this attack when it joins the network. [33]

  A practical example of this attack can be found in Section 3.7.

- **Sybil attack:** Similar to an Eclipse attack, during a Sybil attack, an attacker places a large number of peers with so-called Sybil identities. These Sybil identities are artificially created fake identities. The goal is to flood the network with many Sybil nodes, so the adversary controls a specific fraction of the network.

  A possible impediment to the Sybil attack is hardening the creation of fake identities through cryptographical puzzles. Additionally, limiting the number of identities per host in the routing table makes it harder to operate Sybil nodes. Limiting the number of Sybil identities per physical host (e.g., by checking the IPv4 or IPv6 address) works because these Sybil identities often run on the same physical machine. [33]

  A practical example of this attack can be found in Section 3.7.

- **Churn attack:** Based on a Sybil attack, an adversary might abruptly take a large portion of its Sybil peers down, such that the network gets unstable, i.e., lots of queries fail. As this attack involves having lots of references to adversarial nodes in the victim's routing table, the attack is most likely to succeed when a node joins the network with an almost empty routing table. [33]

- **Adversarial routing:** In this attack, an adversary routes queries for keys or peers into a subnet of adversarial nodes. This works by answering with a list of closer adversarial nodes to any query. These adversarial nodes control every query and can even make all queries fail.

  Baumgart et al. [33] propose implementing a lookup algorithm that uses multiple disjoint paths. The query can still succeed if one of the disjoint paths is not infested with adversarial nodes. Executing the adapted lookup algorithm has the drawback of using more bandwidth and possibly being slower as more nodes have to be asked during the lookup process.

2.5.5.2 Coral DSHT

The Coral Distributed Sloppy Hash Table (DSHT) improves the efficiency of the lookup algorithm in DHTs. In Kademlia, the XOR distance metric brought along many improvements compared to previous DHTs. One of the advantages is the simple lookup process with a logarithmically bounded maximum hop count. Even though the number of hops is limited, each hop could travel across continents back and forth until the target node is reached. The reason for this inefficiency is that the XOR distance metric does not consider the nodes' physical locality.

Freedman et al. [34] propose introducing clusters in Kademlia to improve the efficiency of lookup paths. Clusters structure the overlay network into nodes that share a specific maximum Round-Trip Time (RTT), which they also call the cluster's diameter. The Coral DSHT uses three levels of clusters that each node joins, and each level has a predefined diameter. When looking up a query, the Coral DSHT first sends the query to its cluster with the lowest diameter, i.e., the cluster with the lowest RTT. As in Kademlia, the lookup algorithm walks toward the target key. In case of a hit, the target node or value is returned. Otherwise, the requesting node continues the lookup in the next level cluster of the closest node in the previous level. Therefore, the lookup does not restart from the beginning when switching levels. Instead, the lookup starts at a closer node and continues the lookup in a larger diameter cluster. This process continues until the global cluster is reached, which essentially falls back to the original Kademlia lookup. [34]

Another improvement proposed by the Coral DSHT is the "sloppiness" of the DHT. Instead of storing actual data in the DHT, Coral proposes to hold a list of pointers to nodes that store the data locally. For a key, there can be multiple nodes storing the data. Thus a key can have multiple values, i.e., references to nodes. Coral limits the number of values per key on every node. Once the limit when storing a key-value pair is reached, the storing node steps back one hop and tries storing the key-value pair again. This process is repeated until a node is found that stores the pair. In the long term, this adapted algorithm eliminates overloaded nodes and improves the distribution of keys in the network. [34]

2.5.5.3 Other Extensions

Kademlia's lookup algorithm *iteratively* walks toward the target node. Alternatively, R/Kademlia by Heep [30] switches this lookup algorithm into a *recursive* mode. Instead of the requesting node doing all the lookup queries on the individual nodes, R/Kademlia recursively queries all nodes. The requesting node sends a query to a closer node which recursively sends the request to an even closer node in its routing table. This recursive lookup process continues until the target node is found. A recursive lookup algorithm reduces latencies as less time is spent sending the requests and responses back and forth. Additionally, prioritizing nodes with a low RTT in the routing table is easier as long as every node tracks the RTT in the routing table out of the box. Thus, a recursive approach is more efficient compared to the iterative lookup. The downside of the recursive lookup algorithm is that the querying node completely loses control over the lookup process. Improving the performance of recursive lookups with parallel queries is also harder. As parallel queries potentially converge toward the same path, additional undesirable complexity in the algorithm would be needed to circumvent this issue. [30]

KadRTT by Kanemitsu et al. [35] propose an adapted lookup algorithm that takes the RTT into account without increasing the maximum number of hops needed. Additionally, KadRTT introduces an adapted metric for maintaining k-buckets. The efficiency of k-buckets decreases when the node identifiers aren't distributed uniformly. Over time, the node identifiers inside a k-bucket could converge into some range that generates inefficient k-buckets. Therefore, additionally checking the variance with the new candidate peer is essential upon adding or replacing a peer in a k-bucket. [35]

2.6 JSON Schema

JSON Schema is a JavaScript Object Notation (JSON)-based format for schematically defining the structure of JSON documents. Additionally, the format supports validating values in a JSON document. JSON Schema is supported by a wide variety of programming languages via libraries. These libraries often support extending the set of validation rules of JSON schema with custom validations.

Listing 1 provides a JSON Schema example that includes multiple validation rules. Based on this example, we describe the fields of the JSON Schema format:

- `type`: The type specifies whether the root object is an object or a simple primitive value, e.g., a number or string.
- `properties`: Similar to the `ObjectDefinition` interface defined above, the `properties` field specifies the fields, their types and validation rules of a JSON document. Using the `properties` field, we can also build nested structures.
- `required`: This field defines which properties are required on the root level of the object.

- `dependentRequired`: This field allows to define dependencies between optional fields. In this example, the field `articleName` is optional, but if the field is set, the property `articleNameCode` is required.

```
{
    "type": "object",
    "properties": {
        "quantity": {
            "type": "number",
            "exclusiveMinimum": 0
        },
        "articleName": {
            "type": "string",
            "maxLength": 60,
            "minLength": 2
        },
        "articleNameCode": {
            "type": "string",
            "pattern": "^.{4}-.{6}-\\d{4}-.{2}$"
        },
        "addons": {
            "type": "array",
            "items": { "type": "number" },
            "uniqueItems": true
        }
    },
    "required": [ "quantity" ],
    "dependentRequired": {
        "articleName": ["articleNameCode"]
}}
```

Listing 1: A JSON Schema definition for a fictional order model to demonstrate validation rules.

JSON Schema supports different validation rules depending on the type of a value. We introduce the following basic validation rules that we consider important for defining schemas of event payloads:

- `exclusiveMinimum` and `minimum`: These validation rules allow to set an exclusive minimum (i.e., strictly greater than) or a minimum (i.e., greater than or equal to) value for a number.

- `exclusiveMaximum` and `maximum`: Analogously to the minimum number validation rules, these validations allow to set an upper limit for a number.

- `minLength` and `maxLength`: These validation rules specify minimum and maximum length constraints for a string value.

- `pattern`: The `pattern` validation rule checks whether a string value matches a given regular expression.
- `uniqueItems`: This validation rule specifies that an array must contain only unique items, i.e., no duplicate values are allowed.

Listing 2 provides two examples that either validate against the JSON Schema or contain validation errors as described in the comments of the listing.

```
// CORRECT: Successfully validates against the given JSON Schema
{
  "quantity": 3,
  "articleNumber": "500081002",
  "articleName": "16GB Memory DDR4",
  "articleNameCode": "fk2h-lpp3rt-9918-aa",
  "addons": [1, 2, 3]
}

// WRONG: The field articleNameCode does not match the pre-defined RegEx
{
  "quantity": 3,
  "articleNumber": "500081002",
  "articleName": "16GB Memory DDR4",
  "articleNameCode": "!\textbf{fk2h}!",
}
```

Listing 2: A JSON file containing examples that successfully validate or contain validation errors.

CHAPTER 3

Related Work

The idea of storing data on a network of distributed peers dates back to the last century. One of those networks is the popular BitTorrent network, which is still used nowadays. In recent years, these kinds of data networks gained more popularity again, mainly due to the emergence of blockchain networks. As storing larger amounts of data on a blockchain is costly, many research projects have experimented with off-loading the data to a distributed file system based on an off-chain peer-to-peer data network. Therefore, we conduct a literature review in this chapter to research the related work on distributed file systems.

Weber et al. [5] suggested using BCT for BPM in 2016. Since then, much research has been carried out, intending to integrate BPM with BCT. In a literature review on blockchain-based cyber-security applications by Taylor et al. [36], "Data Storage and Sharing" was the second most popular topic among all selected research papers. As BPM is always connected to storing data, like documents, we conduct a narrative literature review to get an overview of the state of the art [13]. We made use of the Scopus search engine, which consults various scientific data sources. The applied search query is documented in Listing 3 and only includes the title, abstract, and keywords. It excludes non-English and conference review papers as they often target many keywords but do not directly contain any research results. We also excluded papers targeting the topics "hyperledger" and "Internet of Things" as both terms mainly result in research papers using private BCTs that we do not want to focus on. Since the research papers use many different synonyms for the same topic, the search query targets common synonyms explicitly, to not exclude any important research papers. Additionally, the search query restricts the publication year to include only the recent research work. The publication year limit was aligned such that our search query targets the last years with increasing research activity in the defined research areas. After running the search query, the 25 most relevant papers are filtered manually by reading their abstracts and skimming through the papers. Table 3.1 lists the included and excluded papers of the literature review. The result of the literature review answers **RQ1:** *What is the current state of the art for distributed off-chain storage technologies?* In addition to the literature review, we take complementary information from transitively referenced papers or papers that specialize in a specific topic but were not part of the search query.

```
TITLE-ABS-KEY (
    blockchain  AND  ( storage  OR  "file system" )  AND
    (
        "off-chain"  OR  "off-chaining"  OR  "off-the-chain"  OR
        "peer-to-peer"  OR  decentralized  OR  distributed
    )  AND NOT (
        hyperledger  OR  "Internet of Things"
    )
)  AND
( PUBYEAR  =  2018  OR  PUBYEAR  >  2018 )  AND
( EXCLUDE ( DOCTYPE ,  "cr" ))  AND
( LIMIT-TO ( LANGUAGE ,  "English" ))
```

Listing 3: Search query targeting research on peer-to-peer storage networks

3.1 Peer-to-Peer Data Networks

Peer-to-Peer data networks gradually evolved over the last two decades. Their motivation was the self-organized and scalable management and sharing of storage without the need for a central server. Like Virtual Private Networks (VPNs), peer-to-peer data networks build a so-called overlay network. This overlay network is a layer on an existing network, like the internet, that structures and connects a set of participating peers. This virtual layer makes it hard for governments or organizations to influence the network. Following that, this overlay architecture implicitly features resistance to censorship and centralized control. [62, 43] Androutsellis-Theotokis et al. [62] conducted a survey targetting different application scenarios of peer-to-peer data networks. Based on this survey, the authors classified the following categories:

- **Communication and Collaboration:** These are peer-to-peer networks for directly communicating with other peers, like instant messaging applications.
- **Distributed Computation:** Using a set of peers to compute processing-intensive tasks collaboratively.
- **Database Systems:** Utilizing a peer-to-peer network for the distributed storage of data records.
- **Content Distribution:** Sharing general purpose data between all participating users in a peer-to-peer network.

#	Title	Included
1	A Blockchain-Based Decentralized Data Storage and Access Framework for PingER [37]	Yes
2	Distributed Off-Chain Storage of Patient Diagnostic Reports in Healthcare System Using IPFS and Blockchain [38]	Yes
3	Peer-to-Peer Distributed Storage Using InterPlanetary File System [39]	No
4	Decentralized Cloud Storage Using Blockchain [40]	Yes
5	Filetribe: Blockchain-based Secure File Sharing on IPFS [41]	Yes
6	A Review on Blockchain-Based Systems and Applications [42]	No
7	IPFS and Friends: A Qualitative Comparison of Next Generation Peer-to-Peer Data Networks [43]	Yes
8	A secure and distributed framework for sharing COVID-19 patient reports using consortium blockchain and IPFS [44]	No
9	Design and implementation of web system based on blockchain [45]	No
10	When Blockchain Meets Distributed File Systems: An Overview, Challenges, and Open Issues [46]	Yes
11	Photograph Ownership and Authorization using Blockchain [47]	No
12	Proposed framework for blockchain technology in a decentralised energy network [48]	No
13	MTFS: Merkle-Tree-Based File System [49]	No
14	Decentralized Data Access with IPFS and Smart Contract Permission Management for Electronic Health Records [50]	Yes
15	Secure Data Management Using IPFS and Ethereum [51]	No
16	Content Addressed P2P File System for the Web with Blockchain-Based Meta-Data Integrity [52]	No
17	Overview of the main challenges and threats for implementation of the advanced concept for decentralized trading in microgrids [53]	No
18	Live video streaming service with pay-as-you-use model on Ethereum Blockchain and InterPlanetary file system [54]	Yes
19	Blockchain-based decentralized storage networks: A survey [55]	Yes
20	Secure Digital Transactions in The Education Sector Using Blockchain [56]	No
21	Toward Privacy-Preserving Shared Storage in Untrusted Blockchain P2P Networks [57]	No
22	Immutable and democratic data in permissionless peer-to-peer systems [58]	No
23	Leveraging node heterogeneity to improve content discovery and content retrieval in peer-to-peer networks [59]	Yes
24	A new cluster P2P file sharing system based on IPFS and blockchain technology [60]	No
25	On the exploitation of blockchain for distributed file storage [61]	No

Table 3.1: Search results from 9.7.2022 of query from Listing 3

One of the most popular networks was Napster, a digital music-sharing network. Examples of general-purpose file-sharing networks are Gnutella, Freenet, and BitTorrent. Daniel et al. [43] identify this era of data networks as *First Generation of Data Networks*. These systems have laid the building blocks for state-of-the-art data networks, like the IPFS. Nearly every first-generation network tried different approaches for structuring and incentivizing peers, which is covered in more detail in Appendix B. Specifically, Kademlia [31], a distributed hash table algorithm, influenced the history of peer-to-peer networks. Many peer-to-peer networks are building upon Kademlia and its routing and structuring algorithms even up until now. Although their initial motivation is still valid, their scale and application have evolved significantly. [62, 43]

The Bitcoin peer-to-peer network, launched in 2008, started a new era called the *Transition Phase*. The invention of cryptocurrencies revolutionized the problem of incentivizing peers to store data. Along with Bitcoin, content integrity gained attention, which resulted in the introduction of Information-Centric Networking (ICN) and self-certifying names into data networks.

In contrast to HTTP, which uses location-based addressing, ICN introduces content-based addressing. Instead of querying data by entering its location, the content is requested directly via an identifying string. In practice, ICN can be compared to looking for a book. In location-based addressing, one can either ask for the bookstore and its detailed location in the store. Content-based addressing would work by looking for a book via its identifying ISBN. ICNs also have a positive impact on replicating content, as the book might be available in more than one bookstore. [63]

In addition to addressing data by its content, self-certifying names implicitly ensure the integrity of the requested content. A typical self-certifying name is the cryptographic hash of the content. The hash can be used for locating the data via content-based addressing and to check the integrity of the received content by comparing the requested cryptographic hash with the hash of the received content. [43]

New data networks were developed with the evolution of cryptocurrency-based incentives, ICN, and self-certifying names. The more prominent ones are IPFS and Swarm, which are also the ones considered in this work. Other identified but less researched next-generation data networks are Storj, Arweave, SAFE, Sia, PPIO, and Hypercore. Daniel et al. [43] classifies this era of data networks as *Next Generation of Data Networks*. Figure 3.1 depicts the peer-to-peer data network epochs together with known examples.

3.1.1 Aspects of Peer-to-Peer Data Networks

When comparing traditional peer-to-peer data networks, they primarily share the aspect of storing data. The decentralized design of peer-to-peer data networks makes it inevitable for all network peers to collaborate. As the intrinsic motivation for collaboration does not suffice, the nodes also get extrinsic motivation by providing incentives for behaving honestly and collaborating according to the protocol.

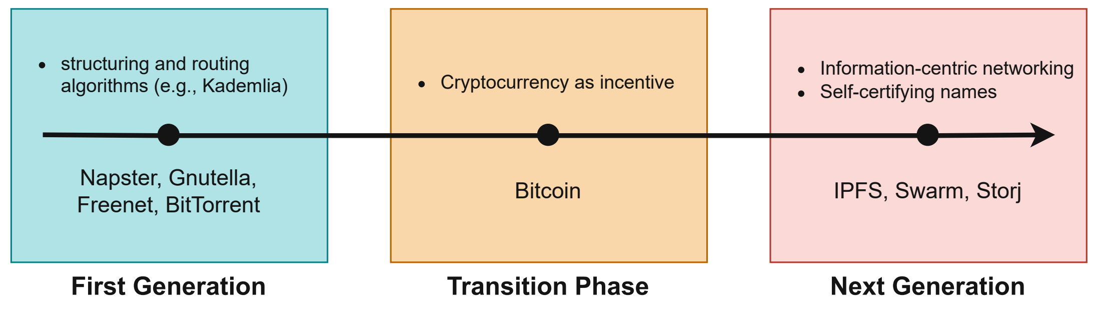

structuring and routing algorithms (e.g., Kademlia)
Napster, Gnutella, Freenet, BitTorrent
First Generation
Cryptocurrency as incentive
Bitcoin
Transition Phase
Information-centric networking
Self-certifying names
IPFS, Swarm, Storj
Next Generation

Figure 3.1: Epochs of peer-to-peer data networks by Daniel et al. [43]

Generally, peer-to-peer data networks cover the following aspects:

- **Trustless Storage:** The primary goal of peer-to-peer data networks is the trustless storage of data. In traditional Cloud storage solutions, the user always needs trust in the Cloud provider because the provider can manipulate the data or pull the plug anytime. Therefore, having a storage that verifiably stores arbitrary data without incurring any trust relationships is an advantage to the end user.

- **Communication:** As any centralized design would harm the aspect of trustless storage, ideally, every peer has to collaborate with the other peers to keep the network functional. In a peer-to-peer data structure, all data gets distributed over the network, and every peer only stores a limited amount of data. To get the data distributed or to request specific data, the peers communicate in a mesh-based network until the target node is reached. Following this approach as a peer is not obvious, as peers benefit only indirectly when collaborating.

- **Incentivization:** To keep the peers motivated to follow the protocol, the network incentivizes peers with built-in mechanisms. These incentives range from remunerating nodes for their bandwidth efforts or running costs to continuously organizing lotteries for all honestly behaving peers.

3.2 Motivation for Peer-to-Peer Off-Chain Storage

Today, peer-to-peer networks are often tightly connected to blockchain-based applications, and they also inherit similar properties as blockchains (e.g., data integrity or immutability). On the other hand, they can fix the bloating issue in blockchains. Storing data on a blockchain is costly and inefficient. Keeping them on a peer-to-peer data storage saves costs and reduces the size of the blockchain. Additionally, more prominent blockchains (e.g., Bitcoin or Ethereum) also have the problem of centralization due to the vast amount of storage needed to host a full node. Therefore, only powerful nodes with enough storage and computational resources can join the blockchain network.

Norvill et al. [64] proposed the storage of smart contract creation transactions to off-chain storage. The blockchain would instead hold a reference to the off-chain file as a cryptographic hash. Storing this kind of transaction on a blockchain would reduce its size by over 90% compared to a regular on-chain transaction. [37, 55]

In addition to efficient storage, many peer-to-peer data networks include replication mechanisms in their design. Data replication also implies a more efficient retrieval, as data chunks can be read from multiple distributed nodes simultaneously. Simultaneously downloading different chunks of data better utilizes the network connection and makes retrievals or uploads faster [55, 40]. Some networks even incentivize data replication by paying nodes to serve clients' chunks. If there is a popular file with many downloads, other peers will likely replicate this file to get paid too for offering this file [43, 65].

With the next generation of data networks, Information-Centric Networking (ICN), together with self-certifying names, became two widely used concepts that improve censorship resistance in data networks. The requested content cannot be modified along the way, as the verification based on the file's cryptographic identifier will fail. Additionally, the concept of immutability helps applications to audit all data flows.

3.3 Layered Structure

Huang et al. [46] proposed a layered structure of peer-to-peer data networks. We present the layered structure without the consensus layer, as this is not part of typical off-chain storage systems. The following list shows the layers and shortly summarizes the functionality of each layer:

- **Identity Layer:** Every peer in the network needs a unique identifier, which is used to connect to other peers. Since most off-chain storage systems use a Kademlia-based DHT, the node identifier plays an important role in storing and finding files.
- **Routing Layer:** This layer is responsible for finding the correct target storage node to store a file and also retrieve it later on. The DHT is the component that is responsible for this layer.
- **Network Layer:** The network layer makes sure that the peers form an overlay network and thus find neighboring peers to connect to.
- **Data Layer:** The data layer cares about how the data storage is structured. Typically, files are split into chunks and organized into Merkle Trees. The nodes of the Merkle Tree are then stored and distributed in the data network.
- **Incentive Layer:** The incentive layer provides mechanisms that incentivize the nodes to behave honestly and store or provide data. These incentive mechanisms are described later in Section 3.5.

3.4 Features of a Peer-to-Peer Off-Chain Storage

Apart from storing data, peer-to-peer data networks offer many other features compared to using the blockchain as storage directly. The underlying peer-to-peer connections exchange data between nodes. Additionally, these peer-to-peer connections allow real-time message exchanges with other nodes. If a node is offline, messages can be cached on the storage layer for delayed retrieval by the target node. The ICN-based design makes all content immutable, thus making mutable files and directories a non-obvious feature. Immutability is enforced by storing the data in Merkle trees, which were introduced in Section 2.2. Some peer-to-peer networks even allow low-level access to the Merkle tree to store custom tree-based data structures. In addition to raw data storage, using different cryptographic algorithms, encrypting data or enforcing access control is possible. In the following sections, these features, among others, are introduced in more detail.

3.4.1 Storage

Storing arbitrary data is the core feature of all peer-to-peer data networks. The storage mechanism in IPFS and Swarm, among many other data networks, is based on a DHT. The IPFS protocol specifies that files are stored on a node whose identifier is similar to the cryptographic hash of the file. Section 2.5 describes the peer-to-peer storage mechanisms in more detail.

Before files are stored on nodes, the dedicated storage client splits them locally and aligns them into the final data structure, which is typically a Merkle DAG. An advantage of the Merkle DAG is that all files are made immutable by referencing their content-hash in the DAG. Additionally, duplicate parts of a file do not need to be stored separately, as the Merkle DAG de-duplicates data by design. Instead of persisting the file directly on a storage node, every Merkle DAG node is pushed into the data network. Accessing the file works by requesting the root node of the Merkle DAG and subsequently requesting all other nodes of the DAG. Since all nodes are distributed over the network, downloading the data in parallel better utilizes the available bandwidth, thus making the retrieval faster.

3.4.2 Encryption and Access Control

Although encrypting data stored on an untrusted node is essential, IPFS does not encrypt content due to its modular design. Instead, users are encouraged to encrypt sensitive files manually before uploading them to an IPFS node. Steichen et al. [66] proposed an extension to IPFS called acl-IPFS, which integrates an access control list-based mechanism into IPFS. The access control list is entirely managed on a smart contract, and the forked IPFS version enforces these access rights by checking the smart contract when a user requests a file. Although this solution might work, it does not consider malicious acl-IPFS nodes that do not behave honestly and wrongly grant access to unauthorized users.

Swarm and Storj support encryption in their protocol, although it is disabled on public nodes. To use encryption, Swarm recommends using a local Swarm client and enabling encryption manually. Storj also proposes to encrypt files as early as possible in the storage pipeline. Otherwise, the unencrypted payload would be shared with a foreign node in the data network.

Specific use cases are in need of more sophisticated encryption schemes than symmetric encryption. Verdonck et al. [50] propose the usage of Attribute-Based Encryption (ABE) for securing electronic health records. With ABE, it is possible to encrypt data with a private key that allows for decryption with any of multiple issued private keys. A private key in ABE has multiple associated attributes, which can be used for any property that should allow decryption. In the case of electronic health records, it is possible to encrypt certain documents just for doctors. This kind of access control comes with the drawback of inflexible revocation. Once a particular attribute gets revoked from an encrypted file, the file needs to be re-encrypted. Revoking an attribute from a user's private key is even harder, as every user with the revoked attribute needs a new private key. Additionally, all files encrypted with the revoked attribute need to be re-encrypted to block the usage of obsolete private keys.

Sári et al. [41] propose FileTribe, a data-sharing platform based on IPFS, which allows the confidential sharing of files to a group of users. FileTribe manages different contracts for managing users, files, and groups and coordinating an off-chain group key change. Once a file is changed, FileTribe only stores its difference from the previous version, similar to version control systems like Git. Since FileTribe uses ABE, the key needs to be changed when revoking some users from accessing the group. This results in a computationally expensive process of re-encrypting all files. For simple use cases, it suffices to locally encrypt the data using a symmetric encryption algorithm like AES, as Shash et al. [40] propose in their system. More complex platforms involving shared access for multiple users could still fall back to symmetric encryption, although securely exchanging the key is challenging.

3.4.3 Structured Data

IPFS and Swarm store files in fixed-size chunks. These chunks are stored as leave nodes of a Merkle DAG, with the tree's root node being the entry point for the file. Thus, each link to a file is a reference to the root node of the Merkle DAG that stores the files' chunks in its leaf nodes. The reference to a file is called the Content Identifier (CID) in IPFS, and Swarm hash in the Swarm data network. Structuring data into Merkle DAGs also includes deduplication and versioning as a by-product. As the hash of its content identifies a Merkle DAGs node, existing content in the network can be referenced without storing it again. Versioning is implicitly achieved, as changes in a leaf node are propagated to all ancestor nodes up to the root of the file. IPFS takes structuring even a step further by enabling the storage of any custom Merkle DAG-based data structure through its InterPlanetary Linked Data (IPLD) project. [43, 67]

3.4.4 Naming System

IPFS comes with its name resolution component, the InterPlanetary Name System (IPNS). IPNS works similar to the Domain Name System (DNS), whereby it manages a set of IPNS records that link a public key to an IPFS CID. Instead of a domain in DNS, IPNS uses a public key. A public key makes it possible to update the IPNS record by proving the ownership of the record with the corresponding private key. Immutable files can be accessed via `/ipfs/<CID>`, while mutable IPNS links can be accessed via `/ipns/<PUBKEY>` in the IPFS network or via an IPFS HTTP Gateway, that enables HTTP-based access to IPFS. As the public key in IPNS-links is hard to remember, the public key can be replaced with a domain. IPNS makes this possible through DNSLink that links a human-readable DNS domain to an IPNS public key. Thereby, IPNS-links can also be written as `/ipns/<DOMAIN>`.

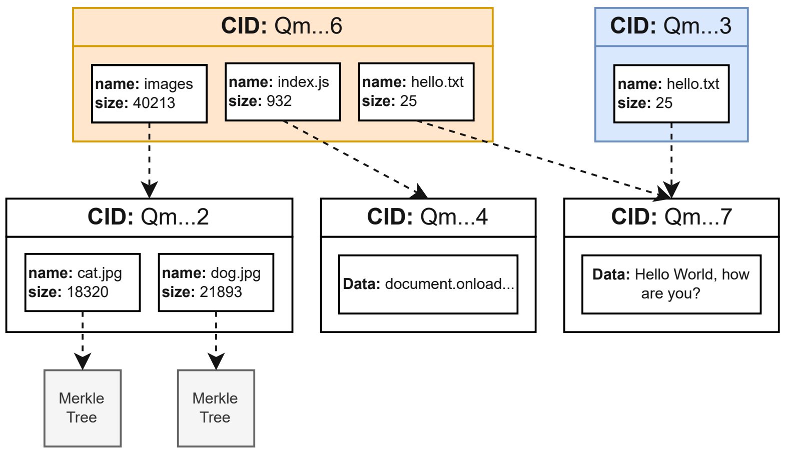
IPNS distributes the records either via a DHT, or via peer-to-peer messaging. The DHT approach offers the advantage of keeping the record alive for a maximum lifetime of 24 hours, after which the record must be renewed. The peer-to-peer messaging approach distributes the record much faster and allows peers to subscribe to record updates. Still, the record is only distributed temporarily and cannot be resolved after propagation. [43, 68]

Alternatively to IPNS, Ethereum offers the Ethereum Name Service (ENS) with similar functionality, but it solely relies on smart contracts that act as domain registrars. It follows the same principle as DNS by hierarchically structuring the domain into a top-level, second-level, and sub-domain. Each domain in each level has its registrar smart contract deployed that resolves the name to some content (e.g., in IPFS or Swarm) or another registrar contract. [43, 69]

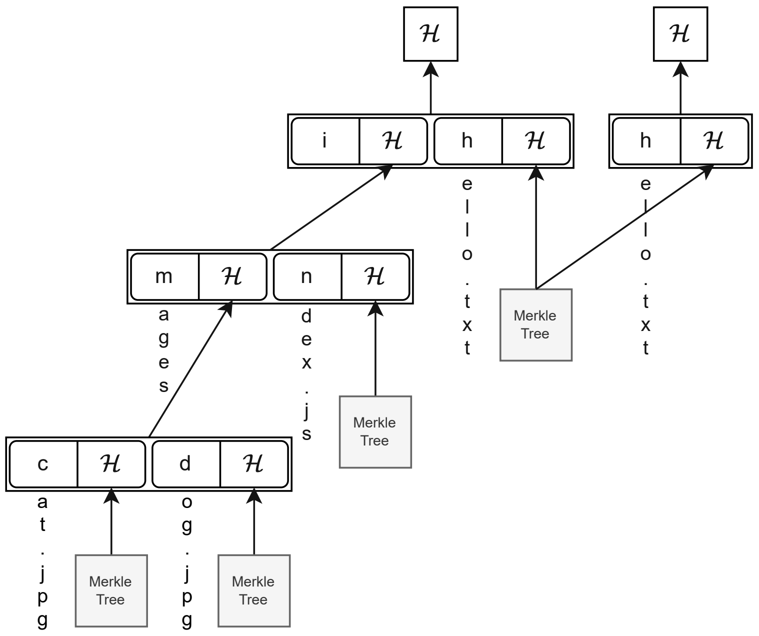
3.4.5 Mutable Files or Directories

Peer-to-peer data networks only offer immutable files due to their ICN concept. ICN makes it hard to update files as the new CID needs to be propagated to all users that had access to the previous version of the file. Otherwise, they would still reference the old content. The solution for the tedious process of propagating the new CID can be solved with mutable files in IPFS. Mutable files use IPNS by generating a public key for each file and linking the record to the file's CID. Upon updating the file, the record is re-linked to the CID of the updated file. Users can still rely on accessing the file through the IPNS record and getting the up-to-date content. Additionally, propagating the IPNS record with peer-to-peer messaging also notifies users when a record gets updated, thus making the user aware of the updated file. [67, 68]

Through IPLD, it is possible to store whole directory structures in IPFS. A node in a Merkle DAG references other files or directories with their filenames and cumulative content sizes. When updating, adding, or removing a file from a directory, the change is propagated up to the root node of the root directory in the Merkle DAG. Therefore, any change also changes the CID of the root directory. The old CID would still link to the directory before the change. To make directories mutable, their root CID can be referenced in an IPNS record. After any change, this IPNS record must be updated with the new root CID, thus forming a mutable file system. Figure 3.2a depicts the structure of a directory in IPFS. The blue node with the CID `Qm...3` is the new root node after deleting the directory *images* and the file *index.html* from the root directory. File deduplication is an important feature here, as the remaining files can be referenced without storing them separately.



(a) IPLD DAG visualization of a directory in IPFS



(b) Compacted Trie data structure of a directory in Swarm from [65]

Figure 3.2: Directory data structures in IPFS and Swarm

Similar to IPFS, Swarm uses a particular type of chunk called single-owner chunks. As the name tells, single-owner chunks have only a single owner. They are similar to IPNS records in that they are signed with a key pair. As in IPNS, single-owner chunks have a fixed address with mutable references. These mutable references allow a user to provide virtually mutable files. Directories are realized with so-called manifests in Swarm. A manifest is a compacted trie data structure containing metadata about a file or a directory and included files. Mutable directories are again possible by updating the reference of a single-owner chunk to the updated directory. [65] Figure 3.2b shows the compacted trie that stores a directory.

3.4.6 Streaming of Data

Even consecutive streaming of data to a data network is possible. For example, Putz et al. [70] use so-called feeds in Swarm to periodically push sensor data to the Swarm network. Swarm feeds work similarly to message queues, as the data updates on the feed are based on a topic. The topic is combined with an indexing scheme, which indexes the consecutive updates on the feed. By hashing together the topic with the current index, it generates a unique identifier that can be mapped to the updated chunk. Depending on the use case, Swarm offers different indexing schemes to cover sporadic, periodic or serial and partitioned data (e.g., for video streams where a single update is not standalone, but always based on a previous update).[65]

3.4.7 Messaging

Based on data streams, also messaging and communication protocols can be implemented. Due to the indexing schemes in Swarm, it is even possible to implement mailboxed communication (i.e., asynchronous communication where the receiver is offline at that time) because all messages are preserved in the history of a feed. However, an important communication requirement is confidentiality, which can be solved with encryption in Swarm. [65]

IPFS supports messaging through its underlying peer-to-peer messaging component called libp2p. It supports basic publish-subscribe operations out of the box. However, a mailboxing mechanism is not supported in the publish-subscribe implementation, which would need to be integrated separately. Such a mechanism could also be based on a distributed database, which gets introduced in the following section.

3.4.8 Distributed Database

Both IPFS and Swarm support the storage of key-value pairs through their data structures like the DHT or the Swarm manifest trie, respectively. For more advanced data structures, there are community projects like OrbitDB[1]. OrbitDB is a database based on IPFS that supports various data structures like key-value pairs or document storage. It implements an append-only log on top of IPFS, which makes use of the underlying storage for all

[1]`https://github.com/orbitdb/orbit-db`

other data structures. This append-only log is a Conflict-Free Replicated Data Type (CRDT), meaning multiple peers can work on the same copy simultaneously. Merging all changes into one common state is possible at any point in time without conflicts. Based on the append-only log, OrbitDB implements other data structures like JSON documents or key-value storages that all benefit from the properties that the underlying CRDT layer offers. More information on OrbitDB and its algorithms is given in Section 5.1.2 and 5.2.

Textile[2] is another project that builds on IPFS, libp2p and Filecoin. [54] Its core entity are Buckets which are essentially folders with all the content in it being pushed into the IPFS network. The data then gets automatically pinned on Textile nodes. On top of Buckets, it provides ThreadDB, which works similarly to OrbitDB. Based on ThreadDB, Textile provides mailboxes that work similarly to Swarms messaging with mailboxing. Textile's user mailboxes are automatically encrypted, thus, only the recipient can read all incoming messages. The encryption of the mailboxes is secured via the recipient's public-private key pair.

3.5 Incentivization

As previously mentioned, some networks incentivize the replication of data. This concept works out for popular data, as nodes get paid for serving chunks of data. On the other hand, long-term storage of unpopular data is not backed by this incentive, as the storage costs for nodes do not amortize this way. Therefore, peer-to-peer data networks came up with a solution called pinning. Pinning services use existing networks by regularly compensating storage nodes for storing pinned data. Clients can then request some data to be pinned and pay a certain amount. The pinning service then ensures that the data is safely stored and pay the respective node for its honest behavior.

Filecoin is a peer-to-peer off-chain data storage and pinning service based on IPFS that incentivizes nodes to store and provide data. The incentivization mechanisms of Filecoin are split into two markets. The storage market ensures the long-term storage of data and compensates nodes for behaving honestly and not losing data. On the other hand, there is the retrieval market for compensating data-providing nodes for their bandwidth efforts.

The storage market employs a Proof-of-Replication algorithm that is based on Proof-of-Space. Proof-of-Space is typically used as a consensus algorithm in blockchain networks like Sia[3]. The basic Proof-of-Space consensus algorithm ensures that some node is verifiably "wasting" a pre-defined amount of space on its disk. Proof-of-Replication extends this by making sure that the node stores useful data instead. For long-term proofs, Filecoin expects its storage nodes to sequentially create Proof-of-Replications for the pinned data. Since continuously submitting these proofs creates a lot of overhead, these proofs are instead sequentially chained in the form of Spacetime proofs.

[2]`https://docs.textile.io/`
[3]`https://sia.tech/`

In addition to incentivizing nodes to store data, nodes are also compensated for their bandwidth efforts on the retrieval market. Requesting files on the retrieval market starts by negotiating a price. Then, the peers start to exchange the file whilst continuously paying for the retrieved chunks. More details on the Filecoin incentivization mechanisms can be found in Appendix B.1.

Swarm's primary incentive mechanism reimburses nodes for their bandwidth or retrieval efforts, by employing three core mechanisms that power the incentive layer of the network:

1. **SWAP (Swarm Accounting Protocol)**
   SWAP handles compensation for the retrieval of chunks. Therefore, the requestor of a chunk continuously sends cheques to the chunk provider. SWAP offers service-for-service and service-for-money compensation. Service-for-service compensation can be used if chunks are exchanged in both directions. This might be the case when the storage provider needs chunks that the requestor possesses. Therefore, redeeming the cheques at a later point can save transaction costs. Swarm even designed SWAP to settle any debts over time, which allows any user to access Swarm files for free in a limited fashion.

2. **SWEAR (SWarm Enforcement And Registration)**
   SWEAR is the mechanism handling long-term storage incentivization. Storage nodes can commit to behaving honestly and storing files for longer periods by offering storage promises on the market. These storage promises are stored in the SWEAR smart contract together with a collateral deposit. As soon as a node misbehaves, it loses the deposit.

3. **SWINDLE (Secured With INsurance Deposit Litigation and Escrow)**
   SWINDLE is the mechanism that enforces litigation when a node misbehaves. Anyone who bought storage promises can challenge the responsible storage nodes to prove the existence of the stored chunks. In case the responsible storage node lost the chunk, it loses the collateral which is deposited on the SWEAR contract.

Swarm also employs a concept called postage stamps for preventing spam in the network. Uploaders of content first need to buy postage stamps so they are incentivized to not spam the data network with junk data. These postage stamps are coincidentally used as an additional storage incentive in the form of postage lotteries. The postage lottery randomly challenges the existence of chunks analogously to the SWEAR mechanism. In return for proving the existence, the responsible storage nodes are then remunerated with the revenue generated from the postage stamps.

More information on Swarm's incentive mechanism can be found in Appendix B.2.

3.6 Applications and Usages

Ali et al. [37] investigated the data storage architecture of a global internet performance monitoring project called PingER. Multiple monitoring agents regularly scan pre-defined hosts and publish the data into a centralized repository. They propose decentralizing the storage by connecting the agents in a peer-to-peer mesh form. Each monitoring agent further distributes their results to other agents for increased data security. A DHT ensures that all files can be found and retrieved later.

Kumar et al. [38] propose storing electronic health records in an off-chain peer-to-peer data storage, like IPFS. The medical sector is often targeted with Denial-of-Service attacks. Having the data distributed over the Internet makes it hard to attack, as there is ideally no single point of failure in decentralized systems.

Lopes et al. [54] built a decentralized live video streaming service called Livepeer. Content creators broadcast their video into the Livepeer network, and peers encode the source video into all necessary target formats. These encoding peers are getting paid for doing their work. The research project from Lopes et al. additionally features a chat functionality that works via IPFS's messaging capabilities.

Allouch et al. [71] implemented a blockchain-based auditing platform for Unmanned Aerial Systems (UAVs) like drones. These UAVs are controllable via a ground control station that sends commands. To audit all commands and the remote sensor data, they use a permissioned blockchain. As some specific data, like flight status details and sensor data is too large to be stored in the blockchain, they use OrbitDB as a decentralized off-chain database. The hash of the data in the database is then committed to the permissioned blockchain for auditing purposes.

In general, IPFS sees a lot of adoption in community projects. The implemented projects range from real-time text editing apps to photo libraries[4].

3.7 Privacy and Security

Peer-to-peer data networks are designed to be more secure due to their decentralized mechanisms. However, these decentralized mechanisms could be reused for Distributed Denial-of-Service (DDOS) attacks against other systems. Cloudflare's quarterly report on DDoS threats from Q3 2022 showed an over 1.200% increase in attacks launched via the decentralized BitTorrent network. [72] A recent report by Cisco Talos also shows a tremendous increase in criminal abuse using the IPFS network [73].

Anomali Labs [74] reported ransomware attacks that utilized IPFS for their command-and-control channel. Similar to this attack, Karapapas et al. [75] did a research project showing a decentralized networks' suitability for building a ransomware as a service platform. As IPFS is hard to take down by authorities, there will probably be future attacks based on IPFS.

[4]`https://github.com/ipfs/awesome-ipfs`

The idea of hosting ransomware on IPFS nodes raises another complicated topic, as it is still unclear who is legally responsible for unencrypted data on foreign computers, as is the case in peer-to-peer data networks. Additionally, there is a need for more research on GDPR compliance of popular data networks, which might also be problematic in many decentralized applications. Politou et al. [76] investigated the GDPR compliance of IPFS, specifically the right to be forgotten. They proposed an anonymous protocol, which distributes deletion requests for a specific file from a verified owner. Their implementation also showed the downsides of decentralized networks, as no incentives encourage a node to comply with the proposed protocol.

Even though many decentralized applications use pseudonyms (e.g., random strings), it is often possible to reveal someone's real identity by monitoring their traffic on decentralized applications, as was shown by Balduf et al. [77]. Since there's no central point where all data requests come together, it seems hard to monitor the network. Balduf et al. [77] proved that by simply placing a monitoring node (i.e., a modified IPFS node that logs all incoming data requests), it was possible to surveil more than 45% of the IPFS network over a longer period. They were able to gather data like popular files, or interests of a specific node by checking all incoming data requests.

Prünster et al. [78] investigated an attack on the IPFS network. By exploiting the connection handling implementation in IPFS, they separated a victim node from the remaining network using an Eclipse attack, which are described in Section 2.5.5.

3.8 Future Challenges

Huang et al. [46] point out that scalability, privacy, and security issues need to be tackled. Trautwein et al. [59] raises the light centralization issue in current peer-to-peer data networks, which becomes more relevant with a more significant adoption. As pinning services (e.g., Filecoin or Pinata) grow, the tendency to off-load resources to these centrally managed nodes will increase. Offloading work to more powerful nodes raises a similar issue to mining-pools in cryptocurrency networks. Prünster et al [78] did a brief analysis of the IPFS network and could only connect to about 6% of all nodes, meaning that all other nodes are likely to be operated behind a firewall. Therefore, the remaining 94% run their DHT in client mode, which means that other nodes do not add them to their routing table. Thus, only 6% fully participate and contribute to the network. This relation needs to change drastically in the future, as the network will not scale otherwise.

3.9 Summary

Peer-to-peer data networks have evolved over the last two decades. The first generation of data networks was tied to specific use cases, while the next generation of data networks is designed to be more flexible. When Bitcoin was introduced in 2008, the research activity towards incentivization algorithms peaked, which greatly influenced the next generation of data networks. The increased flexibility leads to the integration into many different research projects, although the practical adoption still lags behind. Reasons for the low adoption of production-grade applications range from bad usability to missing documentation, even though the set of features is extensive. It is still necessary to gain implementation-specific knowledge to integrate peer-to-peer data networks into applications, as essential details differ from cloud storage technologies.

Our review also shows that decentralized networks suffer from privacy and security issues. As nodes have to collaborate, they exchange lots of information, which can also be abused for tracking purposes. Additionally, their properties, like censorship resistance, make decentralized networks abusable for malware or other criminal activities.

Decentralized storage networks bring many new tools, like peer-to-peer messaging, content-addressed storage, and immutability. The different networks, like IPFS or Swarm, follow different approaches. While all common data networks use a similar DHT, they use different ways to incentivize nodes. The majority of the community still focuses mainly on IPFS, and the adoption of other networks is relatively low.

During our literature review, we did not find research targeting the integration of decentralized networks in mobile applications. However, according to Statista, about 60% of all internet traffic originates from mobile devices [79]. In case more mobile applications are going to adopt decentralized storage networks in the future, the network operators will need to improve the scalability of the decentralized network protocols.

Implementing decentralized storage networks will only be the beginning of the decentralization era. New networks are forming, like Lit Protocol [5], that offer computation as a decentralized network.

[5]`https://litprotocol.com/`

CHAPTER 4

Requirements for Decentralized BPM

In this chapter, we conduct a requirements analysis as defined in the methodology section. Before analyzing requirements, we briefly summarize the current state in BPM with BCT. The current state helps to define a vision in Section 4.1 that we use as our target during the requirements engineering process. We conclude the requirements engineering phase by answering **RQ2:** *Which requirements does a BPM system need to provide for the decentralized execution of inter-organizational processes?* in Section 4.2.3.

4.1 Vision

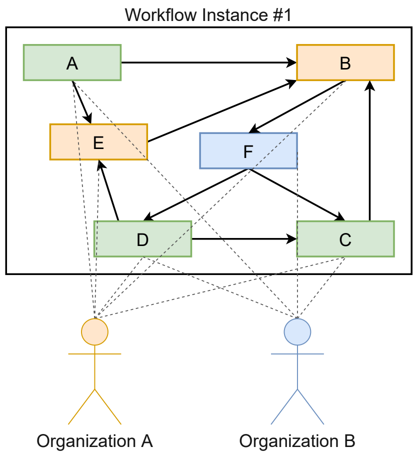
Before defining requirements, we must determine an idealized system we want to achieve. In inter-organizational business processes, all participating organizations pursue a common business goal that is modeled by a workflow containing different activities. Therefore, we introduce a model used in this research work for executing inter-organizational business processes.

The modeled workflows are used as a template and specify how a business process is executed. When executing a workflow, a corresponding workflow instance is created, i.e., the workflow is reusable by instantiating it. Workflows and workflow instances are often called processes and process instances.

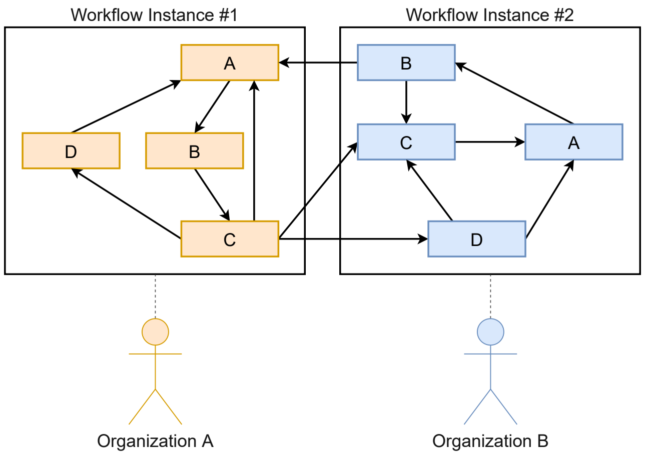
A workflow can be structured and executed in different ways. There could be one global workflow or multiple smaller workflows that collaborate to reach a common business goal. Van der Aalst et al. [80] differentiate between Case Transfer Architecture (CTA) and Loosely Coupled Architecture (LCA). In CTA, there is a global process known to all participants where each business partner takes over specific activities, i.e., the workflow instance is shared among all participants. Figure 4.1a shows a workflow following the

Capacity Sharing scheme. The Capacity Sharing scheme is an example of the CTA, where activities are shared between multiple organizations, and all organizations work on the same workflow instance. Due to only having public processes and activities, centrally managing the process is possible. The CTA structures the business process vertically, i.e., the workflow instances are shared among the participants.

In LCA, there is a global process composed of multiple local processes of each participant. Figure 4.1b depicts such a loosely coupled workflow following the LCA. The process is distributed among all participants making it hard to control the overall process without employing a trusted third party. In other words, all organizations have their internal workflow and communicate at specific points. LCA structures the business process horizontally, i.e., the workflow is split, with each participant having its internal process.



(a) Capacity Sharing (CTA)



(b) Loosely Coupled Workflow (LCA)

Figure 4.1: Workflow Architectures, inspired by [80]

In this research work, we concentrate on loosely coupled workflows, because preserving the privacy of business processes is hard to achieve with centralized models like the CTA. Additionally, we raised other important topics in the motivation of this research work, which are better aligned with a LCA. Therefore, every organization's internal workflow operates mostly independently and communicates with other organizations' workflows at specific activities. The communication between the organizations is asynchronous, i.e., the other organization must not respond directly. An organization could even be offline during the message exchange, depending on the message exchange technology used.

In addition to the communication aspect in the LCA, there is also coordination needed between the collaborating nodes. Every organization manages its internal workflow independently, i.e., no central coordinating entity exists. Message exchanges are often outsourced to trusted third parties that BCT-based models do not need. Instead of relying on trusted third-party approaches, Weber et al. [5] propose to rely on executing the business process fully on a blockchain. López-Pintado et al. [8] implemented Caterpillar, a BPM system that entirely relies on BCT. Mendling et al. [19] summarized several issues with BCT that specifically target approaches that solely rely on a blockchain to execute business processes. The issues range from historical incidents (e.g., DAO or replay attacks) to fundamental studies that challenge the game-theoretic models which are often used to verify BCT-based assumptions [81]. The requirement of having the whole business logic executed on-chain was challenged in various research works, e.g., [82, 2, 23], due to privacy and transparency reasons. Following this challenge, Sturm et al. [9, 10] proposed to use BCT only for its core properties, i.e., to guarantee the integrity of data and for time-stamped commitments. The revised approach manages the execution of a business process locally instead of on a blockchain. Coordination between the collaborating nodes is needed when they exchange messages. By posting commitments on a blockchain, all business partners can verify the global business process which leads to a transparent message exchange. Commitments ensure that the message is authenticated, non-repudiable and verifiable.

Previous research still relies on web services that every business partner needs to host [9, 10]. As pointed out in Section 1.1, this form of message exchange is not feasible generally. Furthermore, using web services for inter-organizational message exchange is prone to system outages when a partner is offline. The resilience against system failures is therefore not given. A system outage can be a worst-case scenario as businesses lose valuable resources like orders and potential opportunities for making revenue.

Given the current state, we identify requirements for implementing the following vision:

> *Executing inter-organizational business process workflows asynchronous and decentralized without relying on a trusted third party such that global tracking of activities is possible.*

We focus on specific aspects of the proposed vision to identify more precise requirements. Therefore, we define a basic construction of our proposed vision to derive the main aspects. The basic construction of an inter-organizational and loosely coupled workflow consists of a local process engine that executes the organization's local business process. By connecting the local process engines with an asynchronous messaging component, a global business process gets formed. Figure 4.2 depicts the basic construction of our starting point. Organization A communicates uni-directional with Organization B, which in turn communicates bi-directional with Organization C. Organization C also communicates with Organization A. We can think of Organization A as a Buyer, Organization B as a Supplier, and Organization C as the shipping provider.

Based on this starting point, we pick the following aspects to focus on during the requirements analysis:

- **Workflow Design and Execution:** We identify requirements that target the design and execution of the local workflow. Additionally, requirements for integrating the message exchange component with the local process engine are defined.

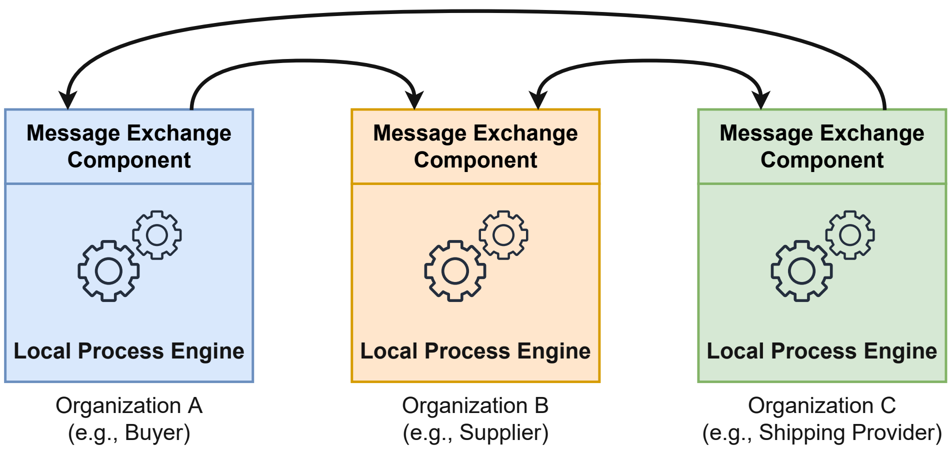


Figure 4.2: Example of a basic construction for an inter-organizational and loosely-coupled workflow architecture spanning three organizations.

- **Inter-Organizational Communication:** We identify requirements for the message exchange between the organizations.

4.2 Requirements Analysis

Based on the vision and the aspects defined in the previous section, we take a bottom-up approach by looking at a typical SCM process as a reference. Figure 4.3 shows the reference SCM workflow. This type of business process was used heavily in previous research, e.g., see [2]. It contains many elements and representative action sequences of typical business processes. In the following sections, we identify requirements for modeling and executing the given SCM process. The background information collected in Chapter 2 assists in finding suitable requirements. Section 4.2.1 describes the *Workflow Design and Execution* aspect, whereas Section 4.2.2 describes the *Inter-Organizational Communication* aspect.

4.2.1 Workflow Design and Execution

Having a workflow that crosses the borders of an organization does not mean that the whole workflow is visible to external parties. Organizations are investing a lot of resources to keep their internal workflows and processes private. Nevertheless, these internal processes might have external interfaces or exit points where communication with other organizations is possible. Therefore, we propose to split a process into internal and external workflows, with the internal workflow hidden from the outside world. In the big picture, a decentralized inter-organizational BPM system should be modeled as a loosely coupled workflow process. Modeling workflows based on a CTA is centralized by design which makes protecting the privacy of businesses hard to achieve.

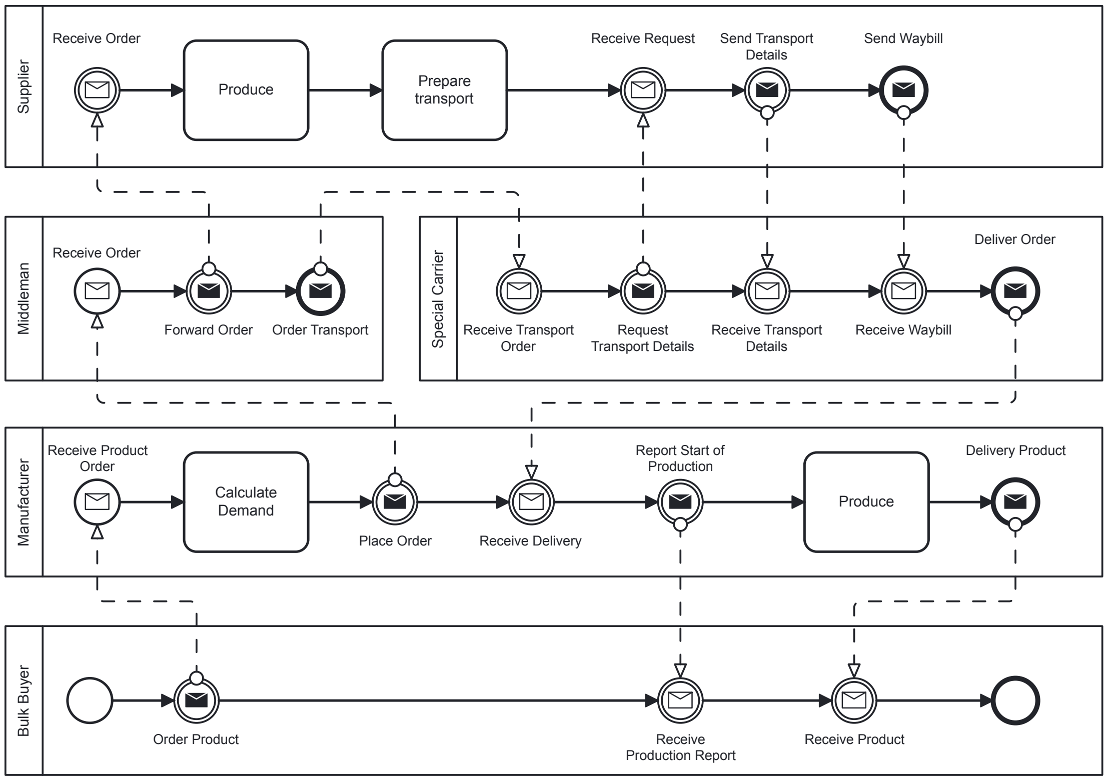


Figure 4.3: SCM Business Process from [5] and [16] in BPMN notation.

According to Van der Aalst et al. [80], a loosely coupled workflow typically spans multiple organizations. Each business partner has a private workflow process that operates independently. The internal workflow transitions between activities without reacting to any externally incoming events. At specific points, it is necessary to communicate with the other business partners' workflows to ensure that the overall inter-organizational business process is followed accordingly. All connection points to other business partners should be explicitly defined and documented. Otherwise, processes would be hard to trace and could potentially lead to indeterministic behavior, which consequently leads to a lack of trust in the system. In other words, workflows must be executed in a transparent way, i.e., the workflow model must provide a deterministic way that determines which action gets executed next. We conclude with the requirements defined in WFL-REQ 1.

WFL-REQ 1: Preservation of Organization Internals

- An inter-organizational business process **must** be modeled as a loosely coupled workflow, i.e., the process **must** be split into internal and external workflows.
- The internal workflow **must** be hidden from the outside world.

- The internal workflow **must** operate independently.
- The inter-organizational business process **should** be modeled as a loosely coupled workflow.
- All connection points **should** be explicitly defined.

When business processes evolve, they tend to get complex. None of the other defined requirements should ever restrict the flexibility of the workflow design. Therefore, we require that the number of actions, decisions, users, and involved external organizations is not limited. Flexibility also means that it must be possible to trigger new but pre-defined workflow instances at other organizations. For example, when ordering a product from a supplier, the supplier provides a pre-defined workflow. Whenever a buyer orders articles, a new workflow instance is created. Ordering articles should, therefore, not be dependent on an existing workflow instance. It must instead be possible to trigger a new instance just in time. These arguments lead us to the requirements defined in WFL-REQ 2.

WFL-REQ 2: Flexibility in the Complexity of the Global Business Process

- The number of actions, decision, users and involved external organizations **must not** be limited.
- It **must** be possible that a new local workflow instance is triggered by another business partner.

Furthermore, when designing software systems, a good fundamental principle is to follow the KISS[1] — Keep It Short and Simple — pattern. Therefore, we aim to design a workflow definition format that is versatile but simple. We expect it to be able to model a wide variety of workflows, from simple to more complex. The KISS principle should help to focus on the essentials and not inflate the format with too many features. Simplicity and minimalism in the definition make it easier for the user to design a workflow and reduces the likelihood of encountering bugs.

As described in Section 2.1, business processes can be classified as orchestrations or choreographies. Due to their centralized control, business process orchestrations are unsuited for inter-organizational workflows. On the other hand, in choreography business processes, the execution of the workflow happens in a decentralized form. Choreography business processes can be implemented using the loosely coupled workflow design defined in the requirement WFL-REQ 1. In loosely coupled workflows, the owner (i.e., the responsible party) changes throughout the global business process based on the current activity. Although modeling the overall inter-organizational business process is not the goal of our concept, we have to incorporate the communication points with the other business partners. We conclude with a recommendation defined in WFL-REQ 3.

[1]`https://people.apache.org/~fhanik/kiss.html`

WFL-REQ 3: Extensible and Flexible Workflow Definition

- The concept **should** follow the KISS pattern.

In BPMN, activities and transitions between activities are the fundamental components for modeling a business process. The requirements WFL-REQ 4, WFL-REQ 5, WFL-REQ 6, and WFL-REQ 7 further describe these fundamental components of a business process.

Similar to how activities and messages are defined and behave according to BPMN, we define the following terminology and behavior for our requirements: Activities are action points in the business process executed by a human worker or a software service. An activity can be a simple *local* activity or an *external* activity. External activities must have one or more external participants defined. An external event is sent to each defined participant upon entering an external activity. The sent external event possibly triggers a transition at the receiving participant when all pre-conditions are met, i.e., the participant expects and accepts the event.

Local activities are fulfilled immediately, i.e., the activity can be left as soon as an expected event occurs. On the other hand, a consensus must be reached for external activities. Therefore, an external activity expects an acceptance acknowledgment from *all* defined participants by default. The default strategy should be overridable with other conditions, like a lower limit or a list of specific participants that must accept the event. For example, we could extend our reference SCM process to require a minimum amount of offers to be obtained before ordering new parts from a supplier. Therefore, we need to model our workflow such that this minimum amount of responses is incorporated into the external activity. Only when the lower limit of responses is reached, an external activity can be declared as finished and the process can move on to the next activity.

In addition to the definition of finishing an activity, every workflow needs at least one *initial* activity that is automatically activated once a workflow instance is started. Additionally, any activity can be classified as a *final* activity that cannot be left anymore and implicitly closes the workflow instance.

WFL-REQ 4 summarizes all requirements targeting the *local* and *external* activities.

WFL-REQ 4: Activities

- External activities **must** have one or more external participants defined.
- External activities **must** trigger events that are sent to the defined business partner.
- Received events **must** trigger a transition if all preconditions are met.
- An external activity **must** receive an acceptance acknowledgment from all defined participants by default.

- Every workflow **must** have at least one initial activity.
- An activity classified as final **must not** be left and **must** close the workflow instance.

The requirement WFL-REQ 4 introduced the concept of events that trigger transitions. Events are simple messages with a sender, recipient, payload, and the event's name. The event payload is not required in all events. When an event gets accepted by the recipient, it triggers a transition from one activity to another. Due to the requirement WFL-REQ 1, we have to differentiate between *local* and *external* transitions. Local transitions can only be triggered by events inside an organization. On the other hand, external transitions can only be triggered by events outside the receiving organization. Thus, an external and potentially malicious organization can never trigger transitions only intended within the internal workflow. WFL-REQ 5 collects all requirements targeting events and transitions that are triggered by events.

WFL-REQ 5: Events and Transitions

- Events **must** have a sender, recipient, payload, and name.
- Local transitions **must** only be triggerable by organization-internal events.
- External transitions **must** only be triggerable by organization-external events.

Jumping from activity to activity without storing any data (e.g., storing a document or additional information given by the activities' human worker) does not make sense in BPM. Even for events without an additional payload object, information is generated by simply having an event applied to the workflow. Sending documents and other metadata is often used when executing business processes. Having the payload on events ensures that all data inputs are incorporated into the workflow design. Payloads on events typically originate from a human worker that fills out some form or a software service that generates a document and attaches it to an event. Event payloads needed for future decisions are processed and stored in the workflow instance upon entering an activity. We refer to this type of storage as workflow instance context or simply context.

To ensure a structured context, every transition must define which of the payloads' properties it should assign to the context. The context is accessible anytime and has the same lifetime as the workflow instance. By aggregating all occurred transitions, the context can be reproduced anytime, even when the workflow instance is finalized. Defining a schema with simple validation rules must be possible to prevent malformed or schematically wrong payloads. We expect at least to have the following validation options:

- **Schematic correctness:** The payload should follow a pre-defined schema, i.e., correct property names and object structure.

- **Required and Optional properties:** The schema should also define whether a property must be filled or can be omitted, as not every property is required.
- **Type of property:** To ensure that a correct value is passed, it should be possible to define a type, e.g., string, number or array.
- **Simple validation checks:** Depending on the type, simple validation rules should be provided, e.g., minimum and maximum string length and minimum and maximum limit for numbers.

WFL-REQ 6 lists all requirements regarding event payloads.

WFL-REQ 6: Event Payloads

- It **must** be possible to process and store event payloads.
- Transitions **must** define which properties of the payload should be assigned to the context.
- The context **must** have the same lifetime as the workflow instance, and **must** be accessible anytime.
- It **must** be possible to validate the schema of event payloads.
- It **must** be possible to define validation rules for event payloads to check the data type of a property.
- It **must** be possible to define validation rules for the following data types:

Numbers:
It **must** be possible to define a valid value range for a number, i.e., minimum and maximum limits.

Strings:
It **must** be possible to define a string's minimum and maximum allowed length.
It **must** be possible to define a RegEx pattern that a string must match.

Objects:
It **must** be possible to define required and optional properties.

Arrays:
It **must** be possible to define the type of an array's items.
It **must** be possible to define an array's minimum and maximum allowed number of items.
It **must** be possible to limit an array to contain unique items only.

The requirement WFL-REQ 6 specified that events can have payloads that can be assigned to the context. Storing properties in the context can be especially useful when a business process requires passing the properties to another business partner's workflow. Therefore, the workflow definition must allow dynamically building external event payloads using either constant values or properties from the context. Furthermore, even the external participant could be unknown when defining the workflow. Thus, it must be possible to dynamically inject the recipient information during the runtime of a workflow instance. A previous event could store the recipient in the context such that it can be used to determine the recipient of an external event later on. These requirements lead us to the definition of WFL-REQ 7.

WFL-REQ 7: External Event Payloads

- It **must** be possible to dynamically build external event payloads with constant values or properties from the context.
- It **must** be possible to evaluate the recipient of an external event dynamically.

Data-based routing is a common feature in traditional workflow engines. Workflows often contain decision gateways, i.e., the workflow splits at a junction based on predefined conditions. For example, in the reference SCM business process, the process might split at the special carrier when the parcel contains hazardous substances. The requirement WFL-REQ 6 specifies that events can also carry a payload and that the payload is stored in the workflow instances' context. A boolean expression specified in the workflow can be evaluated by reading values from the context. The corresponding route in the workflow is then taken based on the evaluation result. In the SCM example, a human worker enters whether the parcel contains hazardous substances and needs special carriage. This decision is stored in the context so the workflow engine can choose the correct route. Data-based routing also affects the route to external participants as mentioned in the requirement WFL-REQ 7. The recipient information (i.e., the external business partner, its workflow, or workflow instance) can be dynamically retrieved from the workflow instance when the recipient is not known at the time of defining the workflow. WFL-REQ 8 summarizes the requirements regarding data-based routing.

WFL-REQ 8: Data-based Routing

- Junctions in the workflow **must** allow dynamic routing by evaluating simple expressions based on the workflow instance's context.

Since we want to achieve a flexible workflow design, a workflow engine also needs an extensible approach for checking custom rules. The variety of necessary checks makes an extensible rule-checking approach inevitable. Every action that manipulates the currently active activity of the business process should be checked to determine whether it passes all rules. An example of a custom rule could be to check incoming events to determine whether the originating organization is allowed or if there is a currently valid business contract. When all rule checks are completed for an activity, a corresponding acceptance or rejection acknowledgment event is emitted. WFL-REQ 9 defines the requirements regarding custom rule validation which are further specified in COM-REQ 7.

WFL-REQ 9: Custom Rule Validation for every Action

- A workflow engine **must** allow the definition of custom rule-based checks on any activity.
- The custom rule checks **must** be triggered on every activity transition, i.e., the currently active activity changes.

4.2.2 Inter-Organizational Communication

Currently, standardized business processes that allow multiple organizations to collaborate are often tightly constrained to a specific business area or type of exchange, i.e., interoperability in the BPM ecosystem is not entirely given for inter-organizational business processes. Similar to private messaging technologies, e.g., WhatsApp[2], Telegram[3], Signal[4] etc., we expect the market for business-to-business communication to grow. A growing market for communication technologies makes compatibility with multiple messaging technologies an inevitable property to stay competitive. Therefore, it is essential to have an exchangeable messaging component that supports multiple messaging technologies.

Since collaboration largely depends on the message exchange component, we define further requirements explicitly for this component. As pointed out, supporting different messaging technologies is essential to be interoperable with as many business partners as possible. The minimum requirement, therefore, is to allow the definition of different messaging technologies for each foreign organization. There is no requirement on which technologies have to be available. The idea is to have an extendable system of messaging connectors. Depending on which messaging technologies the business partners support, corresponding messaging connectors must be configured to connect with them. COM-REQ 1 defines the requirements regarding messaging technology compatibility.

COM-REQ 1: Messaging Technology Compatibility

- The concept **must** support compatibility with multiple messaging technologies.
- It **must** at least be possible to define a messaging technology on organization level.

The previously defined vision prescribes executing business processes in a decentralized and asynchronous manner. Thus, exchanging messages must also use an asynchronous and decentralized technology. Relying on traditional centralized methods would entail

[2]https://www.whatsapp.com/
[3]https://telegram.org/
[4]https://signal.org/

additional risks, for example, attacks in the form of censorship by governmental organizations. Compared to that, decentralized approaches are more agile and harder to block. Decentralized protocols are typically open to the public. Thus everyone can host a node and contribute to the network. By operating a decentralized network with many nodes, the numerous redundant connections make the message exchange more fail-safe and scalable. Therefore, we define COM-REQ 2 to specify a requirement and a recommendation regarding the decentralized message exchange.

COM-REQ 2: Decentralized and Asynchronous Message Exchange

- The message exchange **must** be asynchronous and decentralized.
- The messaging technology **shall** be scalable and improve fail-safeness.

Many decentralized systems, especially BCT-based ones, are designed to ensure the pseudonymity property of all participants. Pseudonymity makes it hard to offer addressability in decentralized networks. When executing business processes, it is often inevitable to know all business partners. Thus, collaborating organizations have to reveal their identity to each other.

Furthermore, every participating organization must have an address bound to the organization's identity, i.e., to ensure the authenticity property of messages. The format of an address often depends on the messaging technology used. Typical formats are mail addresses, random alphanumeric identifiers, or domain names. Additionally, recipient addresses should not often change, at least between two business partners. Unstable recipient addresses could lead to confusion, raise uncertainties or hinder reachability.

The recipient's address must be known to exchange messages with another business partner. Specifically, the organization's workflows and workflow instances must be addressable to refer to them in events as specified by WFL-REQ 8. We conclude with the requirements defined in COM-REQ 3.

COM-REQ 3: Identifyable and Addressable Organizations

- Business partners **must** reveal their identity.
- Business partners **must** ensure a consistent possibility to address them as recipients and their workflows and workflow instances for exchanging events.

In order to provide a secure message exchange, previous research mainly relied on integrity, authenticity, non-repudiation, and confidentiality properties [83, 84]. Confidentiality is hard to achieve due to the global tracking of activities defined in our vision. Employing a confidential message exchange while allowing global tracking of activities would need further measures like zero-knowledge proofs. Therefore, ensuring confidentiality while enabling global activity tracking is considered for future work. As an interim solution, private peer-to-peer networks, e.g., IPFS Cluster[5], could be used alternatively to ensure a confidential message exchange, at least to some extent.

We expect that the following security properties must be ensured and conclude them in COM-REQ 4:

- **Integrity:** When exchanging messages with other organizations, we expect that the integrity of the messages is ensured, i.e., manipulating messages is impossible. A fundamental property for ensuring integrity is that the integrity hashes are deterministic. The integrity algorithm of both parties in a message exchange must be interoperable, which can be guaranteed by using the same message canonicalization algorithm. An example of such a canonicalization algorithm would be the JSON Canonicalization Scheme[6].

- **Authenticity:** The sender of every message must be identifiable. Furthermore, impersonation must be prohibited by using a secure authentication schema.

- **Non-Repudiation:** Each message exchange must secure the non-repudiation property, meaning that it must be impossible to deny a taken action. Without ensuring the non-repudiation property, no organization could trust the system.

- **Uniqueness:** To ensure that none of the messages can be reused or sent multiple times, each message must be unique. The uniqueness property further hinders an adversary from eavesdropping on the messages and re-sending them in the original sender's name. Therefore, even the authenticity property would be broken without ensuring the uniqueness property.

COM-REQ 4: Secure Message Exchange

- The message exchange technology **must** ensure the integrity of all message exchanges.
- The sender of a message (i.e., an event) **must** be identifiable.
- Having sent a message **must** be non-repudiable by the sender.
- Each message **must** be unique, i.e., the messaging technology **must** prevent replay attacks.

Sometimes, a verified timestamp is needed because the integrity and authenticity properties from the requirement COM-REQ 4 do not ensure a correct timestamp inside the message. This requirement does not ensure a high-precision timestamp. Instead, it ensures that the sender has committed to the message, which is essential for certain

[5]https://ipfscluster.io/
[6]https://www.rfc-editor.org/rfc/rfc8785

activities in case of a dispute. A commitment is not strictly necessary for every activity, but there should always be the possibility to enable commitments for pre-defined external events. We define this requirement in COM-REQ 5

COM-REQ 5: Commitments with Verifiable Timestamps

- It **must** be possible for the sender to commit to an external event by providing a verifiable timestamp.

Solely relying on a decentralized message exchange, as COM-REQ 2 specifies, does not work out in general. Many decentralized message exchange technologies rely on peer-to-peer connections that make it necessary for both parties to be online and reachable simultaneously. As this would be to the detriment of at least one business partner, the decentralized message exchange should be capable of employing a mailboxing mechanism. Mailboxing is essential when the receiving business partner is offline during a message exchange. In that case, it would not be possible to receive new messages, e.g., new orders or other business-related actions. Therefore, we define the requirement COM-REQ 6.

COM-REQ 6: Mailboxing

- The message exchange technology **must** employ a mailboxing mechanism, i.e., sending messages to the recipient must be possible even when the recipient is offline at the time of sending.

Even though a decentralized message exchange brings many benefits, it is typically asynchronous communication. That means, unlike in human discussions, there is no guarantee of an immediate response. Therefore, the sender can never be sure that the message was received without an explicit confirmation message. Additionally, a sending business party needs information on whether the event got accepted or rejected. Altogether, the following types of acknowledgments are required:

- **Confirmation of Receipt:** In decentralized messaging technologies, the event of receiving a message is hard to identify. We distinguish the following two cases:
  1. **Recipient is online:** The recipient is online and directly receives the message. In this case, the sender no longer needs to wait as the message was successfully transmitted to the desired target peer. A confirmation of receipt signed by the receiver needs to be sent to the sender.
  2. **Recipient is offline:** When the recipient is offline, a direct peer-to-peer communication is not possible. The requirement COM-REQ 6 specifies a mailboxing service specifically dedicated to this case. Thus, we can assume that a mailboxing peer stores the message until the recipient gets online again. The mailboxing service needs to respond with a signed confirmation of receipt. To verify the confirmation of receipt, the sender must know the public key of the mailboxing service beforehand. Without verifying a mailbox service's identity, a sender cannot assume that

the mailboxing service follows the service level agreements of the receiving party. Alternatively, the sender can also declare the message as received when a specific number of unknown peers store the message. This lower limit should be well chosen so that all unknown peers are not expected to go down until the recipient gets online again. The alternative method has no guarantees and should, therefore, only be used as a last resort, e.g., when the recipient does not employ a mailboxing service.

- **Acceptance or Rejection:** When the recipient processes a received message, multiple checks get executed. The event must be expected in the currently active activity of the business process, the payload wellformed, and all required payload properties supplied. Additionally, a custom rule engine can take an even closer look and employ several semantic and business-related checks that the employed workflow requires. The acknowledgment process must not take longer than a few seconds, as only the essential checks are intended during this process. Additional feedback and validation loops must be incorporated into the workflow design when needed. Finally, a corresponding acceptance or rejection message must be sent to the message's sender. The corresponding rejection reason should be given to the sender in case of a rejection.

Additional types of acknowledgment are not intended. If further acknowledgment types are necessary, they can be incorporated into the workflow design. In COM-REQ 7, we conclude with the requirements regarding acknowledgments for message exchanges.

COM-REQ 7: Acknowledgements on all Levels of the Message Exchange

- The messaging technology **must** exchange a confirmation of receipt upon the recipient receiving the message.
- The mailboxing mechanism **must** exchange a confirmation of receipt when the recipient is offline.
- The mailboxing service officially employed by the recipient **must** be identifiable and known to all senders.
- It **shall** be possible to declare a message as received upon being replicated on a pre-defined minimum limit of foreign (not officially commissioned) mailboxing services.
- Upon a recipient receiving a message, it **must** process the message and respond with an acceptance or rejection message.
- The rejection acknowledgment response **should** contain a reason that specifies why the event got rejected.

Accomplishing a privacy-preserving way of exchanging messages is essential when executing business processes. Specifically for decentralized networks with foreign nodes, preserving the privacy of business processes and organizations is hard to fulfill. In the past, various attacks have tried to trace connections and content in decentralized networks to extract valuable information, as shown in Section 3.7. Therefore, in critical scenarios, there is no way around operating private networks of decentralized peers, e.g., using technologies like IPFS cluster[7]. COM-REQ 8 summarizes the requirement for ensuring a privacy-preserving message exchange.

COM-REQ 8: Privacy-Preserving Message Exchange

- It **must** be possible to operate the workflow and messaging component in a privacy-preserving manner.

Exchanging files and documents is a common scenario in many business processes. Depending on the employed message exchange technology, exchanging files is supported out of the box. When file exchanges are unsupported, a separate decentralized file storage system, like IPFS or Swarm, is required. Similar to message exchanges, the previously defined integrity, confidentiality, and privacy properties must be preserved when using a separate file storage system. The exchanged files must be linked in the message such that the recipient can verify the files' integrity. The recipients are responsible for providing or paying for a pinning service so the file can not get lost. Therefore, we define the requirements in COM-REQ 9.

COM-REQ 9: File and Document Exchange

- It **must** be possible to exchange files via every compliant message exchange technology.
- The file's integrity, confidentiality, and privacy properties **must** be preserved.
- The file **must** be linked to the message when it is not directly included in the message.
- The recipient **must** employ a pinning service to ensure that the file is securely stored even when both business partners are never online simultaneously.

We are designing a decentralized system that inherently follows a distributed architecture. A well-known issue with distributed and specifically decentralized systems is to reach a consistent state. Especially when network partition issues or other downtimes occur, a distributed system is likely to reach an inconsistent state. Consistency specifically targets collaboration with other business partners. A business partner cannot trust any incoming events when they are not consistently committed (i.e., they could vanish or get invalid due to conflicting events). Thus, a condition must decide when an incoming event is considered consistent. Defining a probability limit for the consistent state condition might even be acceptable depending on the business process. For example, in the Bitcoin blockchain, a transaction is said to be finalized after waiting for six new blocks [85]. The

[7]https://ipfscluster.io/

probability of having a transaction excluded after waiting for six new blocks is negligible. Furthermore, there must be a single consistent view on the chain of events, i.e., the events must be in total order. A consistent total order is essential for validating a business process, e.g., it makes a difference whether *event A* occurs before *event B* or the other way around. The described consistency model is called *Serializability* and is further defined in COM-REQ 10. [86]

COM-REQ 10: Serializability

- The overall system (i.e., the workflow and message component) **must** comply with the serializable consistency model.
- A probability limit **should** be defined that specifies when a state is considered finalized.

Our vision also defines that tracking activities on a global scale should be possible. Particularly for SCM business processes, it is often required to be able to trace the products on a global scale. For example, to label a product with a specific certificate, it must be traceable back to its roots. Traceability in a loosely coupled workflow architecture brings some challenges to solve. First, due to the missing transparency of the internal workflows, tracing a product's route cannot be solved on a technical level. The buyer needs to trust each business partner along the products' way. An intermediary and maliciously acting business partner might switch the more expensive certified products with cheap and uncertified ones for its advantage. Detecting these circumstances on a BPM level goes beyond the aim of BPM.

Furthermore, the global inter-organizational workflow is often unknown and highly dynamic as it may span many different business partners or intermediaries. It is, therefore, hard to model a specific workflow beforehand. For example, the route of a product may depend on the current market situation, e.g., prizes of raw materials. Simply ignoring the precise definition of the global business process may not work, i.e., defining a global workflow that only models the existence of unknown intermediaries. Some malicious intermediary could skip appending his activity to the chain of inter-organizational activities not to attract further attention. Again, solving these kinds of issues technically on a BPM level goes beyond the aim of BPM.

Modeling a global workflow can improve the transparency of the overall process and consequently improve the trust properties. One such approach was proposed by Fdhila et al. [87]. They investigated in their research work the validation of Global Compliance Rules (GCRs) in process choreographies. Similar to the loosely coupled workflow architecture, they assumed that organizations have internal processes. Their research has shown that relying solely on public message exchanges between two organizations does not cover all relevant use cases for GCRs. In their example, a manufacturer needed to prove that each article must pass a final test after production. Since the activities *Production* and *Final Test* are private tasks of the manufacturer, publicly verifying this validation rule is not possible. Therefore, they proposed to keep a second workflow model that chains all public activities together. Thus, an organization specifies public activities so all organizations involved in the business process can check GCRs. The public workflow model should be separated from the message exchanges for privacy reasons. Still, all public activities must follow the same rules as any message exchange, i.e., to ensure all security properties. Nevertheless, there are still scenarios where exposing activities is not possible. Although Fdhila et al. further investigate this issue, it is out-of-scope for this research work.

Therefore, we expect a decentralized BPM system to provide enough flexibility to implement global tracking of activities. Solving the issues mentioned above should be incorporated into the workflow design based on the specific use case. Furthermore, validation of custom rules on the workflow consisting of public activities happens identically to the local workflow, i.e., by using the custom rule checking mechanism as defined in WFL-REQ 9. We conclude the requirements on global tracking of activities in COM-REQ 11.

COM-REQ 11: Global Tracking of Activities

- A decentralized BPM system **must** provide capabilities for globally tracking activities.
- It **must** be possible to incorporate global and local workflow transparency issues into the inter-organizational workflow design.
- Exposing activities or parts of activities into a public workflow spanning all organizations **must** be possible.
- Message exchanges between two organizations and the public workflow **shall** not be mixed.
- It **must** be possible to commit to public activities using BCT as defined in COM-REQ 5 for external events.
- Validation of GCRs **must** happen via the custom rule checking mechanism defined in WFL-REQ 9.

4.2.3 Research Question 2

In this section, we answer **RQ2**:

> *Which requirements does a BPM system need to provide for the decentralized execution of inter-organizational processes?*

In Section 4.1, we have discussed the current state and the vision we want to achieve. Furthermore, we defined two aspects we want to focus on during the requirements analysis process. The first aspect targets the design and execution of workflows, and the second focuses on inter-organizational communication requirements. Finally, we refer to the requirements defined in Section 4.2, which answers the second research question.

CHAPTER 5

Concept and Prototype

In this chapter, we propose a concept that complies with the requirements defined in Chapter 4. In Section 5.2, we describe the implementation of the concept in the form of a prototype. Finally, we answer **RQ3:** *Which technologies and architecture can be leveraged to implement the defined requirements?* in Section 5.2.2.

5.1 Concept

The concept is based on the LCA, which combines local workflow execution engines of different organizations with an inter-organizational communication component. Our approach includes a workflow definition format that handles different aspects of the local workflow execution, like tackling communication with external business partners, autonomous execution using data-based routing, or validation of event payloads. The second aspect — Inter-Organizational Communication — handles the collaboration between business partners. In our proposed concept, we ensure collaboration by modeling an interoperable message exchange mechanism. The concept achieves the properties of integrity, non-repudiation, and consistency through commitments using BCT. Combined with our proposed mechanisms for modeling inter-organizational workflows, we additionally show that global activity tracking is possible. The requirements analysis process in Section 4.2 split the requirements into two aspects targetting *Workflow Design and Execution* and *Inter-Organizational Communication.* Therefore, we also split the concept into these two aspects. Section 5.1.1 introduces the concept regarding the workflow design and execution requirements, while Section 5.1.2 describes the inter-organizational communication concept.

5.1.1 Concept for Workflow Design and Execution

The aspect *Workflow Design and Execution* covers the definition of a format that models workflows and satisfies the requirements from Section 4.2.1. Therefore, the format incorporates a way to validate payloads, supports the dynamic generation of external events, and allows a workflow to use data-based routing. Additionally, the workflow execution uses custom rule checks based on an existing concept. We introduce this format and its features in the following sections.

5.1.1.1 Workflow Definition Format

A workflow models a business process that is typically repetitive, i.e., a process gets executed over and over. Therefore, we model a workflow once and instantiate it every time we want to execute it. These instantiations are called workflow instances. A typical business process consists of activities executed by a human worker or a software service. In the following, we refer to human workers and software services interchangeably as workers. Each activity waits for input from the corresponding worker until the process moves on to the next activity. The actual path that is taken in a business process depends on actions or events that the responsible worker submits. For example, a business process that ships packages could take a completely different path when a worker submits the information that the package contains dangerous goods. When the workflow transitions from one activity to another, the input from the worker may be stored in the workflow instances' context. The context is a storage that is bound to the workflow instance. Binding the context to the workflow does not make sense, because the workflow is only used as an abstract definition (i.e., a blueprint) for the derived workflow instances. Therefore, a workflow context would be shared among all workflow instances, which we do not want. With a workflow instance context, every activity can access data that a previous transition stored in the context. For example, a worker prepares and submits an order in an activity. Thus, the activity moves on to the next and stores the submitted order in the context. An upcoming activity wants to send this order to the supplier and uses the order stored in the context. Communicating with other participants works through activities declared as external. These external activities have the following information modeled in the workflow definition:

1. Which business partners to contact?
2. How to contact them?
3. What information the message needs to contain?

A business partner might already respond with information that needs to be stored in the context. Therefore, an external activity defines what to expect as acknowledgment from the contacted business partner. Figure 5.1 visualizes the basic concept with a minimal buyer-supplier example.

Listing 4 shows the proposed format for defining workflows in the language TypeScript[1]. TypeScript is a JavaScript variant that incorporates a type system. Thus every property of the defined interfaces also defines the expected type. The proposed format is thought

[1] https://www.typescriptlang.org/

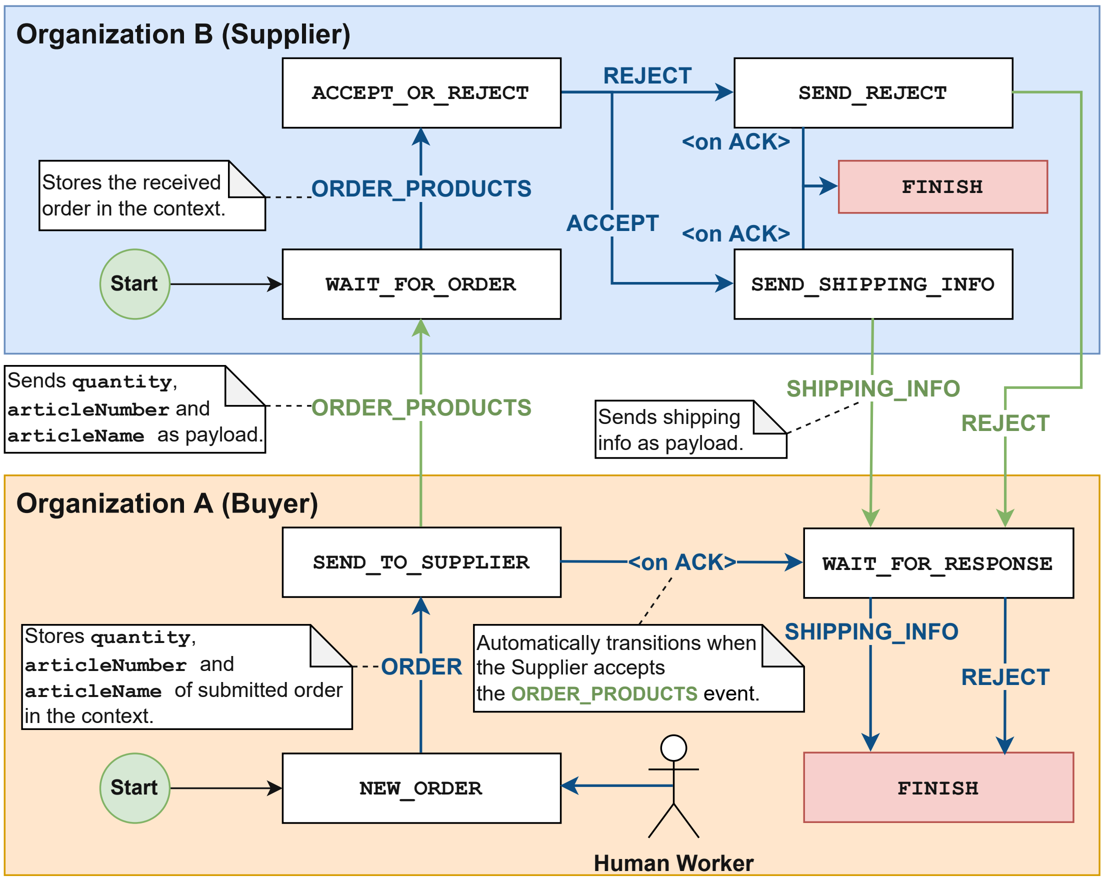

Organization B (Supplier)
ACCEPT_OR_REJECT
REJECT
SEND_REJECT
<on ACK>
FINISH
Stores the received order in the context.
ORDER_PRODUCTS
ACCEPT
<on ACK>
Start
WAIT_FOR_ORDER
SEND_SHIPPING_INFO
Sends quantity, articleNumber and articleName as payload.
ORDER_PRODUCTS
SHIPPING_INFO
Sends shipping info as payload.
REJECT
Organization A (Buyer)
SEND_TO_SUPPLIER
<on ACK>
WAIT_FOR_RESPONSE
Stores quantity, articleNumber and articleName of submitted order in the context.
ORDER
Automatically transitions when the Supplier accepts the ORDER_PRODUCTS event.
SHIPPING_INFO
REJECT
Start
NEW_ORDER
FINISH
Human Worker

Figure 5.1: A workflow example demonstrating the mechanisms of our proposed concept.

to be used either as JSON or Yet Another Markup Language (YAML) as definition languages. Although the extension of an existing standard format for business processes (e.g., BPMN) would make more sense, we propose a custom format to focus on the concepts behind it. As our concept operates on a lower level, it is still possible to map any feature-compatible BPMN workflow to our custom workflow. We do not aim to provide a fully BPMN-compatible feature set but rather limit our concept to the identified requirements.

The following paragraphs describe Listing 4 in more detail:

`WorkflowDefinition` Interface The interface `WorkflowDefinition` is the head of the format that includes an identifier (Line 2), the definition of all contained activities (Line 3) and specifies the initial activity (Line 4). In most use cases, the initial activity will be an idle activity that waits for an incoming event, i.e., a local event or an external event from a business partner. The list of activities is a mapping from an activity identifier to the definition of an activity.

```typescript
interface WorkflowDefinition {
  id: string;                                    // Identifier of Workflow
  activities: Record<string, ActivityObject>;            // Activities
  initial: string;                                  // Initial Activity
}

interface ActivityObject {
  on?: Record<string, EventObject | string>;             // Transitions
  conditionalNavigation?: Record<string, string>;          // Junctions
  external?: boolean;                           // Activity sends Events
  externalParticipants: ExternalParticipant[];           // Recipients
  externalCondition?: ListCondition;        // Acknowledgment Condition
  final?: boolean;                                // Is Final Activity?
  globalTracking?: ExternalParticipant;   // Global Tracking Integration
}

interface EventObject {
  target: string;                                    // Target Activity
  external?: boolean;                 // Externally triggered Transition?
  schema?: object;                      // JSON Schema Payload Validation
  assign?: ObjectDefinition;             // Context Assignment Definition
}

interface ExternalParticipant {
  id: string;                          // For Response Identification
  connectorType: string;                              // Connector Type
  recipientInfo: Record<string, any>;           // Recipient Information
  event: string;                                          // Event Name
  payload?: ObjectDefinition;             // Dynamic Payload Definition
  acceptanceSchema?: object;          // Payload Schema Validation on ACK
  assignOnAcceptance?: ObjectDefinition;    // Context Assignment on ACK
  rejectionSchema?: object;          // Payload Schema Validation on NACK
  assignOnRejection?: ObjectDefinition;    // Context Assignment on NACK
}

type ObjectType = "string" | "object" | "number" | "boolean" | "array";

interface ObjectDefinition {
  objectType: ObjectType;                     // Type of current Object
  value?: any;                              // constant value to assign
  jsonPath?: string;              // JSON Path Expression for Assignment
  properties?: Record<string, ObjectDefinition>;    // Nested Properties
}

interface ListCondition {
  allOf?: string[];                   // all participants need to respond
  anyOf?: string[];                   // any participant needs to respond
  min?: number;                              // lower limit of responses
  max?: number;                              // upper limit of responses
}
```

Listing 4: Custom Workflow Definition Format defined in TypeScript.

`ActivityObject` Interface The `ActivityObject` interface defines transitions to other activities (Line 8) and contains two flags to mark an activity as external (Line 10) or as final (Line 13) activity in a workflow. Additionally, when the external flag is set, external participants (Line 11) need to be defined. External activities send events via a messaging component to the external participants (external participants are the business partners to collaborate on the inter-organizational business process). These external events are sent automatically once the activity gets active in the workflow. The definition of an external participant is described in the interface `ExternalParticipant`. The `ActivityObject` also enables modeling of junctions for data-based routing via the field `conditionalNavigation` (Line 9) mapping. The `conditionalNavigation` property maps an expression-language key to a target activity, similar to normal transitions. The only difference is the expression-language-based key that is evaluated at runtime given the workflow instances' context. The key's expression must evaluate to a boolean value that decides whether the route to the defined target activity is used. More information on data-based routing is given in Section 5.1.1.2.

Additionally, each activity can be exposed to an overall inter-organizational workflow that enables global activity tracking. Global tracking of activities works by defining an event that gets exposed to the inter-organizational workflow. The global workflow is again hosted by a certain participant. Thus, we can use the `ExternalParticipant` interface on the field `globalTracking` (Line 14) to define an event that is sent to the inter-organizational workflow, equally to how external events are handled. More on global tracking of activities is described in Section 5.1.2.5.

`ListCondition` Interface Optionally, an external condition (Line 12) can be given that an external activity has to satisfy. When the external condition is satisfied, the current activity is seen as completed. The external condition specifies how many or which external participants have to respond with a successful acknowledgment. The interface `ListCondition` provides four different conditions which should suffice for most use cases:

- `allOf` (Line 46) allows to set a pre-defined list of external participants. The participants are referenced via the external participant identifier (Line 25). This condition expects all referenced participants to respond with a successful acknowledgment response.
- `anyOf` (Line 47) works similarly to the `allOf` condition. However, instead of requiring all reference participants to respond, the `anyOf` condition expects only one of the referenced participants to respond successfully.
- `min` (Line 48) requires a minimum number of participants to respond successfully.
- `max` (Line 49) allows setting a maximum number of participants that must respond successfully. This condition is useful when implementing a scenario where the first few responses are accepted, and all other responses are ignored, e.g., the fastest

response wins in a competition. The order of the participant acknowledgments is essential, which makes the finalization of the order using BCT inevitable. The finalization process is described in Section 5.1.2.2.

Without specifying an external condition, all defined external participants need to respond successfully for an external activity to be completed. Once an activity is completed, it accepts events to move on to another activity via a defined transition.

`EventObject` Interface The definition of transitions to other activities (Line 8) in the `ActivityObject` interface maps event identifiers to event specifications. Event specifications can either reference the target activity directly or contain an `EventObject`. An `EventObject` allows specifying sophisticated mechanisms for transitions other than a simple navigation to the target activity. Beyond specifying a target activity (Line 18), a transition might expect an attached payload that can be schematically and semantically validated via the schema property (Line 20). The schema property is optional (i.e., the validation of event payloads is not mandatory) and allows to define the schema of the event payload in the JSON Schema[2] format. The JSON Schema format also provides simple validation rules. Section 2.6 describes JSON Schema in more detail.

Additionally, the optional `assign` property (Line 21) provides a mechanism to store event payloads in the workflow instances' context as defined in WFL-REQ 6. As required by WFL-REQ 5, transitions must be explicitly defined as external transitions to accept events from outside the current organization for security reasons. Therefore, an `EventObject` contains an optional external flag (Line 19).

`ExternalParticipant` Interface The `ExternalParticipant` interface is used on an external activity and specifies whom to send an event to, which event, and what data the payload contains. The fields `connectorType` (Line 26) and `recipientInfo` (Line 27) define the messaging connector to use and additional information needed to contact the business partner. The `recipientInfo` is a key-value pair mapping that the messaging connector uses to contact the right recipient, for example, an address. Alternatively, JSON Path[3] expressions can be used to dynamically retrieve values from the context. JSON Path is an expression language that supports navigating and referencing properties in a JSON object, similar to what XPath[4] provides with XML data. Connectors provide compatibility with multiple messaging technologies as defined in COM-REQ 1. Additionally, connectors need to take care of the mailboxing requirements defined in COM-REQ 6 and COM-REQ 7. The name of the event at the other business partner is defined in the field `event` (Line 28).

When sending an event to another business partner, we expect an *acceptance* or *rejection* acknowledgment response as defined in COM-REQ 7. The *acceptance*

[2]`https://json-schema.org/`
[3]`https://goessner.net/articles/JsonPath/`
[4]`https://www.w3.org/TR/xpath-3/`

and *rejection* acknowledgments can also contain payloads. Therefore, the definition of an `ExternalParticipant` contains a similar mechanism as transitions (i.e., `EventObject`). The fields `acceptanceSchema` (Line 30) and `rejectionSchema` (Line 32) allow the validation of the received payload using JSON Schema. Additionally, the fields `assignOnAcceptance` (Line 31) and `assignOnRejection` (Line 33) provide a way to assign the received payload to the context for later usage. For example, the event payload for an acceptance acknowledgment might contain the identifier to directly reference and communicate with a workflow instance at the other business partner. Receiving a workflow instance identifier is required when we trigger a new instance at the business partner and do not know the instance identifier yet, as we have defined in WFL-REQ 2. On the other hand, the event payload for a rejection response should contain a rejection reason which might be needed in the context, as COM-REQ 7 specifies.

The `EventObject` and `ExternalParticipant` interfaces provide a mechanism to assign an event payload to the context of a workflow instance. The mechanism is used in the `assign` (Line 21), `assignOnAcceptance` (Line 31) and `assignOnRejection` (Line 33) fields. Additionally, the `payload` (Line 29) field provides a way to dynamically construct event payloads, which uses the same mechanism as assigning properties to the context. These assignment fields use the `ObjectDefinition` interface to dynamically construct an object with a pre-defined schema that is used to build payloads or assign properties to the context.

`ObjectDefinition` Interface An `ObjectDefinition` is a recursive data structure that allows building nested objects. Each `ObjectDefinition` might either specify a constant value (Line 40) or a JSON Path expression (Line 41) that references some value in the context of the `ObjectDefinition`. When defining an `ObjectDefinition` for the `assign*` fields, the event payload is used as context. In contrast, when building event payloads, the workflow instance's context is used as context. For example, when building an event payload, we can reference a value that was stored in the context in a previously submitted event payload. The `properties` (Line 42) field allows to recursively define an object, i.e., to build nested structured objects.

Listing 5 defines an example that shows the usage of the `ObjectDefinition` interface. When evaluating the given `ObjectDefinition` using a corresponding context object filled with data, the resulting object from Listing 6 gets produced. The object definition in Listing 5 defines two properties on the root level: `articleNumber` and `deliveryAddress`. While the first field gets evaluated using a JSON Path expression, the latter field (`deliveryAddress`) defines a nested object. This nested object further defines the two properties `addressLine` and `department`. The `addressLine` field uses a constant string, and the `department` field uses a JSON Path expression that reads the `name` property of the `departmentInfo` object from the context.

```
{
  "objectType": "object",
  "properties": {
    "articleNumber":   { "jsonPath": "$.context.articleNumber" },
    "deliveryAddress": {
      "objectType": "object",
      "properties": {
        "addressLine": { "value": "Mainstreet 1" },
        "department":   { "jsonPath": "$.context.departmentInfo.name" }
}}}}
```

Listing 5: An example showing the usage of the `ObjectDefinition` interface.

```
{
  "articleNumber": "500081002",
  "deliveryAddress": {
    "addressLine": "Mainstreet 1",
    "department":   "Sales"
}}
```

Listing 6: The resulting object generated by the `ObjectDefinition` in Listing 5 using a corresponding context.

5.1.1.2 Data-based Routing

During the requirements analysis phase, we have identified data-based routing as an essential requirement, which we defined in WFL-REQ 7 and WFL-REQ 8. Data-based routing enables autonomous routing based on data stored in the workflow instances' context. Our proposed workflow definition format supports data-based routing through the following mechanisms:

- **Junctions:** An activity in the workflow, defined by the `ActivityObject` interface, either navigates via an incoming event that triggers a transition or exclusively via a conditional navigation mapping. Junction activities cannot respond to any other event, because once a junction is active, it atomically transitions to the next activity. The conditional navigation mapping maps a key to a target activity. The key contains an expression evaluated using the workflow instances' context. The expression uses the JEXL[5] expression language that is easy to understand because it supports most of the expression constructs used in JavaScript or shell scripts [88]. When the expression defined in the key evaluates to true, the corresponding route to the target activity is taken. We limit our concept to allow only one condition to take on the value true, otherwise, the workflow would result in multiple activities being active (i.e., having parallel activities). Nevertheless, this limitation complies

[5] https://commons.apache.org/proper/commons-jexl/

with the corresponding requirement WFL-REQ 8, because it is only necessary to allow dynamic routing, i.e., choosing the path through the workflow based on dynamically retrieved values from the context.

- **JSON Path Expressions:** On the other hand, the proposed workflow definition format allows data-based routing by evaluating the recipient information of external participants at runtime using JSON Path expressions. The recipient address (e.g., a domain or recipient identifier) can be given via an event payload. The event payload gets assigned to the context, which is used to evaluate the JSON Path expressions.

5.1.1.3 Custom Rule Checks

The need for checking workflow-specific rules rises with the growing complexity of workflows such that simple semantical validations do not suffice. For this reason, WFL-REQ 9 requires the implementation of custom rule checks.

We use an existing concept for rule checking defined by Kleebinder [16]. The existing approach utilizes externally managed rule engines subscribed to all events. Rule engines are software services that subscribe to incoming events and check pre-defined rules. Modeling the rule engines as separate software services decouples the rule checking from the workflow execution. This architectural abstraction makes the complexity of workflow-specific rule checks maintainable. In addition to transition events, workflows and workflow instances are checked as soon as they are created. The workflow and rule engines communicate via a pre-defined interface. All defined rule engines must check the event and respond with a validation result. A reason must be included in the response when a validation failure occurs.

COM-REQ 7 requires that the validation result gets sent back to the workflow via an acknowledgment event. Using events even for the responses enables fully transparent auditing of the validation process. The workflow engine must also pause all interactions with the workflow while waiting for validation results to stay in a consistent state. Otherwise, the workflow must be rolled back later in case of a validation failure.

5.1.2 Concept for Inter-Organizational Communication

In our concept for defining and executing workflows, we described that external activities have one or multiple external participants defined. These external participant definitions also contain information on the messages exchanged with a business partner. In this section, we describe how the collaboration with these external participants works.

5.1.2.1 Immutable Event Logs

Before starting the communication between two organizations, a contract needs to be negotiated that defines the connection between the organizations. The contract includes the synchronization points of both workflows, i.e., which events get exchanged or which payload is necessary. The concept assumes that such a contract already exists. Communication with business partners works by exchanging messages in the form of events as defined by the contract. These events trigger transitions in the local workflow instance of the business partner. A new workflow instance gets created automatically if no workflow instance exists but the corresponding workflow does. Depending on the used message exchange technology, an identifier or an address of the newly created workflow instance is sent back to the originating organization of the event. The originating organization can use the address of the workflow instance in subsequent message exchanges, similar to a ticket number in support systems. The other organization can directly allocate the event to the corresponding workflow instance when a workflow instance address is given. Alternatively, both organizations could agree on using the same workflow instance identifier throughout a business process.

In addition to the workflow semantics on a higher level, we also need to define the underlying transport technology. To preserve compatibility, we do not define a specific messaging technology. Instead, we propose to use the messaging technology as an independent transport layer. On top of the transport layer, we maintain a replicated data type in the form of an event log. The event log serves as an event storage between two organizations. An event log perfectly fits from an architectural point of view because we need all events exchanged between two organizations in a totally ordered sequence. Both business partners must have the same view on the order of the events. Otherwise, there is the possibility of inconsistent business process states. In our concept, events reference the previous event to form a chain, i.e., each event has a reference to the previous event, like blocks in a blockchain. Having a chain of events ensures the uniqueness of each event. Since event logs are not finalized out of the box, a separate finalization step is needed for all communications with external partners. The finalization mechanism for event logs is described in Section 5.1.2.2.

On top of the transport layer, our system design uses multiple event log streams that store events. By defining aggregations on events, we can always calculate the current state of a business process. This approach — called event sourcing — involves storing data in streams and using aggregations to calculate a specific data view. We describe the specific event logs in more detail in the following paragraphs. Figure 5.2 visually describes our proposed event log architecture. The *Workflows* stream holds a list of all defined workflows and is bound to an organization.

Each of the defined workflows needs two event streams (depicted in blue):

1. **ExternalEvents:** The *ExternalEvents* stream is publicly writable and provides a postbox-like service that other business partners can use to trigger new workflow instances of a specific workflow. In other words, if a business partner wants to trigger a new workflow instance, the business partner needs to append the respective event to this event log.

2. **WorkflowInstances:** The *WorkflowInstances* event log models the list of all workflow instances that belong to an organization. This stream contains only events

that create new workflow instances. It is only writeable by the owning organization, i.e., no foreign business partner can force-create a workflow instance for security reasons.

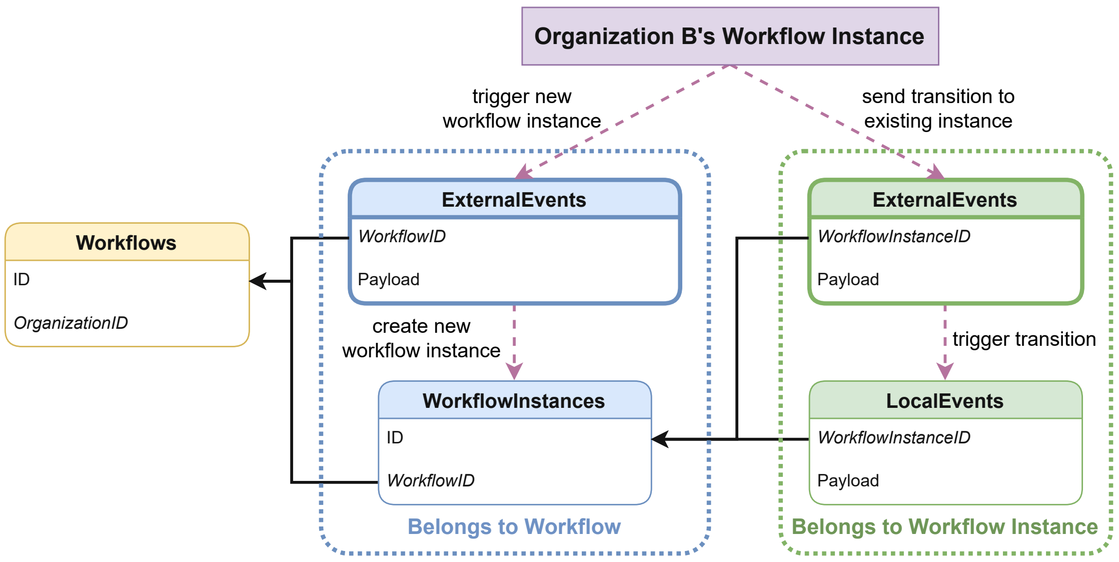


Figure 5.2: Diagram visualizing the event log streams in our concept.

Moreover, each of the workflow instances defines two additional streams (depicted in green):

1. **ExternalEvents:** Similar to the *ExternalEvents* stream that belongs to a workflow, this event stream also contains events from other business partners. It is also publicly writeable. Instead of triggering new workflow instances, this *ExternalEvents* stream is a postbox for workflow instance transitions. Therefore, other business partners use this stream to collaborate by sending events to this *ExternalEvents* stream. After processing the incoming events, a corresponding event gets created in the *LocalEvents* stream. This ensures that we have a single source of truth, which contains all events in one stream.

2. **LocalEvents:** The *LocalEvents* stream contains all business process-related events, i.e., transitions and acceptance or rejection events. We can calculate the state of a process by aggregating all events with a dedicated aggregation function.

Figure 5.2 additionally visualizes the collaboration with a business partner called Organization B. Both *ExternalEvents* streams model the entry point into a foreign organization's workflow system. Organization B can either trigger a new workflow instance or collaborate on existing workflow instances. The organization can then decide to accept or reject the external event from the *ExternalEvents* stream and forward it to the corresponding internal event stream, i.e., either the *WorkflowInstances* or the *LocalEvents* stream.

Managing the local process with event logs is not strictly mandatory for our concept, but event logs bring similar properties to the local process execution as they do for collaborations. Furthermore, the local process execution is not bound to a single machine, but could potentially be used in a distributed way, e.g., each human worker belonging to an organization could access all business processes and execute them locally. Therefore, we decided to model the whole business process (i.e., the local process and all collaborations between business partners) with event logs.

To identify the origin of events and preserve their integrity, all events must be digitally signed by their originating organization. Additionally, all events in the event log are chained like a block in a blockchain. Chaining all events in the event log makes each event unique. There might be similar events, but they never have an identical predecessor in the event log. As all events are chained through their cryptographic hash, referencing an event implicitly guarantees that the event log's history stays consistent.

The workflow system is the application that implements our concept and handles all event streams accordingly, i.e., it subscribes to all streams and acts on incoming events. One key argument that we considered was that event streams should be as short-lived as possible for efficiency and performance reasons. Having short-lived streams also means that an event stream should not grow over time to an unmanageable stack of events. Our concept proposed event sourcing as architecture, which implies that we need to do a full-stream scan to calculate an aggregation. Thus, streams with fewer events are more efficient in calculating aggregations. Therefore, we separated all event streams (i.e., *LocalEvents* and *ExternalEvents*) to belong to a single workflow instance. Workflow instances typically do not have a long lifetime, so we expect them to end soon (e.g., after some days or weeks) and not to grow over time.

Additionally, our event log architecture must ensure bi-directional communication between two organizations. In the requirement COM-REQ 7, we have defined that the receiving organization (or its mailboxing service) must send a receipt confirmation for incoming events. Furthermore, the organization must send an acknowledgment event (i.e., accepting or rejecting the event) to the originating organization after processing an external event. Both response events must be appended to the *ExternalEvents* stream bound to the workflow instance of the originating organization. The workflow system of the originating organization subscribes to incoming events. It can subsequently process these responses and update its local workflow instance accordingly, e.g., automatically issue a transition to the next activity. The processing of incoming events ensures that all important data is stored in the corresponding *LocalEvents* stream. Thus, the *LocalEvents* stream has a transparent history that allows calculating the current state of a workflow by performing an aggregation on it.

Another critical aspect of the event streams concerns permission handling. By replicating the event logs over a decentralized transport mechanism, permissions are not enforceable centrally. Checking permissions (i.e., who can write to which stream) in a decentralized and trustful way is hard to implement. Therefore, many event log concepts do not directly enforce permissions on write. Instead, write permissions are checked when reading the events. Unauthorized events that do not respect the defined permissions are simply ignored.

This optimistic permission-handling mechanism also has the downside of being prone to the split-brain problem. Two business partners could have an inconsistent set of permission checks implemented, leading to both seeing a different view of the data. Specifically for the *ExternalEvents* stream, as it is the interface between two organizations, the split-brain problem could lead to substantial issues. Missing events could lead to uncertainties on both sides. In practice, this should not be a problem for nodes that follow a consistent implementation. By publishing the configured permissions for every event stream, collaborating organizations could check that they use the same configuration. Another downside of the optimistic permission-handling mechanism is that there must be an attack-prevention mechanism enabled. In Section 3.7, we discussed reports stating that peer-to-peer networks are used heavily for attacks. Our concept does not implement any additional attack-prevention mechanisms, but it relies on the measurements taken by the underlying peer-to-peer network.

5.1.2.2 Handling Conflicts in the Replicated Event Log

Our current architecture replicates an event log data structure over a transport layer. When an organization receives new data over the transport layer, it must merge the incoming data with its local replica. Similar to version control systems like Git, conflicts might occur, and the data cannot be merged without further measures. Therefore, we propose using an event log data structure based on a Conflict-Free Replicated Data Type (CRDT), which embed a conflict resolution algorithm directly into the data structure. Conflicts can still occur, but the CRDT solves conflicts with application-specific conflict resolution methods. Data objects can be merged in any order while always leading to a consistent result. The exact consistency properties are defined by the CRDT, so choosing the right CRDT is essential. Figure 5.3 shows an example of a simple CRDT, which uses very simple consistency properties, namely *uniqueness* and *numerically sorted*. Merging never results in a conflict, irrespective of the merge order, because merging only involves conflict-free operations:

1. **Append:** Merge both data structures by appending the second to the first list.
2. **Remove Duplicates:** Remove all duplicate items.
3. **Sort:** Sort the items numerically.

Numerically sorting the items could even be seen as optional, given that the CRDT defines its consistency properties such that two objects are identical regardless of the order.

As the example in Figure 5.3 shows, agreeing on a consistent state works irrespective of the merge order and, most importantly, without interaction other than exchanging replicas. Getting non-interactive consistency suits using a peer-to-peer network as a transport layer.

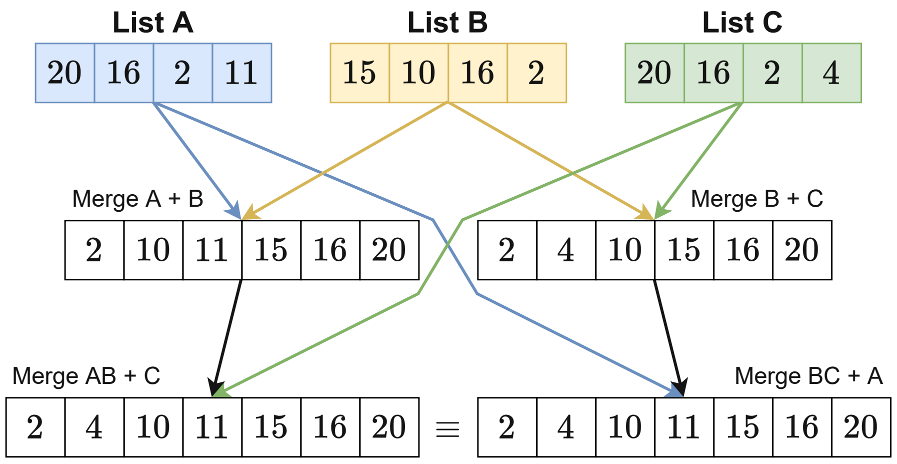


Figure 5.3: Simple CRDT consistently forming a sorted list of unique items.

In our requirements analysis, we have identified more specific requirements that we need to find a fitting CRDT for. We propose to use Merkle-CRDTs, although other CRDTs with equal properties would work as well. Merkle-CRDTs are a mix of Merkle-DAGs and CRDTs proposed by Sanjuan et al. [89]. Merkle-DAGs, as introduced in Section 2.3, are directed acyclic graphs where each node links its children via their hash. A Merkle-CRDT uses Merkle-DAGs as its underlying data structure. When an event gets appended to the Merkle-CRDT-based event log, it is added as a new root of the DAG. The updated Merkle DAG then gets replicated to all other participating nodes that merge their local DAG with the updated one. To be precise, it suffices to replicate the root node, as the replicating client can subsequently request missing intermediate nodes (i.e., lazily loading children nodes).

For solving the conflicts when merging two Merkle DAGs, some notion of time is needed to put two conflicting nodes in order. Wall clock times do not suffice as CRDTs should work without a central entity, and even using a central time server does not guarantee that none of the nodes is spoofing the time. The solution in distributed computing is the usage of logical clocks. Logical clocks do not necessarily expose an absolute time value but can put two events in relative order. A logical clock defines whether an event α happened before an event β. Merkle-CRDTs use the Merkle DAG as a logical clock, which the researchers called a Merkle Clock. Merkle Clocks can be used to merge two Merkle DAGs. When merging two Merkle DAGs $\mathcal{M}_\alpha$ (with root node α) with $\mathcal{M}_\beta$ (with root node β), the following conditions are checked:

1. **Case $\boldsymbol{\alpha = \beta}$:** Both DAGs are equal, so no merging is necessary.

2. **Case $\boldsymbol{\alpha \in \mathcal{M}_\beta}$:** $\mathcal{M}_\alpha$ is already part of $\mathcal{M}_\beta$, so $\mathcal{M}_\beta$ is more up to date and is used as new Merkle DAG.

3. **Case $\beta \in \mathcal{M}_\alpha$:** $\mathcal{M}_\beta$ is already part of $\mathcal{M}_\alpha$, so $\mathcal{M}_\alpha$ is more up to date and is used as new Merkle DAG.

4. **Otherwise:** Both DAGs are joined by having two root nodes. Two events happened concurrently in terms of the logical clock.

Specifically, the conditions 2 and 3 profit from the immutability property of the Merkle DAG. By checking if a root α is part of another Merkle DAG, all children of α are automatically checked as they are linked via their hash in the root node.

Due to the generic condition 4, Merkle-CRDTs do not provide a totally ordered sequence of events out of the box. In addition to this limitation, CRDTs only provide eventual consistency. At any time, another event log replica could pop up (e.g., containing old and unseen events) that is not merged with all other replicas of the CRDT. Consequently, CRDTs do not provide a finalized event log without further measurements. Therefore, to get a totally ordered and finalized sequence of events, we need a component that finalizes the event log. Figure 5.4 shows how a blockchain can be utilized to reach a consensus on totally ordering concurrent events. When a consistent view on all replicas suffices, the event hashes can be used for a consistent order.

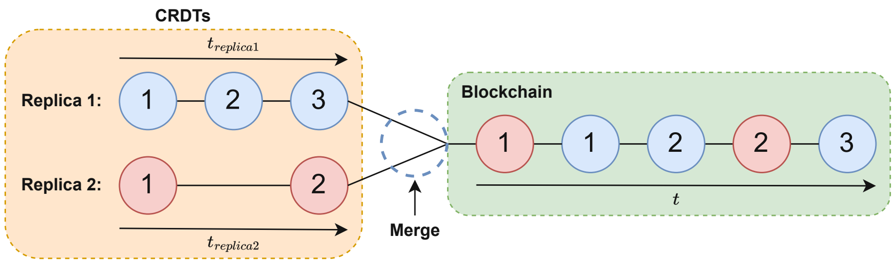


Figure 5.4: Illustrates how a blockchain reaches a consensus on totally ordering concurrent events from different replicas.

5.1.2.3 Commitments

If we need a better guarantee on ordering concurrent events, an external system can assist in finding a consensus, e.g., a blockchain. That is where commitments using a BCT come in. Commitments not only help to find a consensus on concurrent events, they additionally finalize event occurrences, ensuring non-repudiation of occurred events.

Before sending an event to another business partner (i.e., via any of the *ExternalEvents* streams), the sending organization has to commit to the event. The process of committing an event works by consistently hashing the event. Ensuring the consistency of hashes involves agreeing on a consistent event serialization. Simply using a cryptographic hash function on a JSON object could lead to inconsistent hashes as the ordering on the keys is not fixed. Therefore, our events in the JSON format need to go through a canonicalization algorithm. The JSON Canonicalization Scheme (JCS) is such an algorithm that, once applied to a JSON object, always results in the same serialized output. After applying the JCS, a secure cryptographic hash of the canonicalized event must be calculated. The resulting hash string must then be appended in a verifiable form to a blockchain. All collaborating organizations must agree on the used BCT such that all participants can verify the commitments at any time. This approach was proposed in the concept from Kleebinder for ensuring consistency in workflow systems [16]. Müller et al. [2] classified this architecture under the pattern "Blockchain as a Transparent Process Event Log".

Once committed to the agreed-upon BCT, the resulting hash string gets sent embedded into the event to the other organization. We do not put any more requirements on the cryptographic hash other than using a state-of-the-art hashing algorithm. Since the sending organization can choose the algorithm, the multihash[6] format should be used. Multihashes wrap a cryptographic hash with additional metadata like the used hash algorithm and its digest length. These additional metadata fields ensure that all other organizations know how to verify the event.

As mentioned before, commitments also assist in the handling of concurrent events. A blockchain provides an order on commitments that we can use to order events in case the business process needs a time-critical conflict resolution. After looking up the commitment on the blockchain, we can compare the block numbers and even the transaction position when two commitments have been finalized in the same block.

5.1.2.4 Exchanging Files and Documents

Exchanging files and documents is an essential feature of business processes. Common types of exchanged documents are invoices, orders, or acknowledgment documents of orders. We have defined in COM-REQ 9 that exchanging documents must be possible when executing business processes. Even though many transport layers would support documents embedded in the exchanged events, we do not want to rely on that assumption in our concept. Sending files would blow up the size of the CRDT-based event log, leading to inefficiencies in the long term. Therefore, we propose to send files via an external peer-to-peer and content-addressed file storage. As introduced in Chapter 3, content-addressed storages provide a hash-based address that points to the file. This address, also called the content identifier, is used to reference the file and at the same time provides self-verification of the file's content, i.e., checking the file's integrity is possible by using its hash-based address. The content identifier must be referenced in an event such that the recipient can access the file. Additionally, the sending organization needs to take care of the availability of the file. In peer-to-peer storages, pinning services exist that replicate a file, so it is available when the organization's node is not.

Confidentiality of business processes and their documents is an important topic. Many decentralized and peer-to-peer networks are entirely public and do not provide confidentiality out of the box. Thus, we propose to use private peer-to-peer networks when confidentiality is needed. Private peer-to-peer networks provide the same mechanisms as

[6]https://multiformats.io/multihash/

public networks. Instead of being accessible publicly, private networks allow only specific organizations to join. A popular private network that is based on IPFS is IPFS Cluster[7].

5.1.2.5 Global Tracking of Activities

Global tracking of activities allows the implementation of process choreographies for inter-organizational business processes. Our concept is based on loosely coupled workflows that collaborate via a pre-defined interface with other organizations' workflows. Based on the defined requirements for global activity tracking in COM-REQ 11, we propose the concept of directly modeling the inter-organizational business process as a separate and independent workflow. This inter-organizational workflow needs to be known and defined upfront and spans all organizations that are part of the workflow, but there is exactly one organization that is hosting the workflow, i.e., the organization whose local workflow system subscribes to all the event log streams and handles incoming events. The hosting organization might look like a single point of failure in the concept. However, storing the workflow model in a public event log allows all organizations to operate independently. A specific property of the inter-organizational workflow is that it solely contains external transitions, i.e., all transitions are triggered by events from the *ExternalEvents* stream. Overall, the inter-organizational workflow represents the progress from all connected local processes which belong to the collaborating organizations.

In our workflow format defined in Section 5.1.1, we have defined that each activity can be configured to expose an event to the inter-organizational workflow. Publishing events to the inter-organizational workflow is not restricted to external events. Thus, even internal activities can be exposed via an event. Since the workflow definition allows to build an event dynamically, internal details can still be omitted in the public event sent to the inter-organizational workflow. All external events sent to the inter-organizational workflow behave similarly to all other external events in that a commitment to the blockchain can be posted such that the event gets non-repudiable. Since every organization can operate the inter-organizational workflow independently, they can also define custom rules using the rule engine mechanism defined in 5.1.1.3.

Figure 5.5 shows an example workflow that showcases a SCM business process. Organic cotton T-shirts can be ordered and tracked in this SCM business process. The global tracking of activities implementation assists in validating the certification, manufacturing, and shipment process along the supply chain. The dotted violet arrows illustrate the public events sent to the inter-organizational workflow on the top. The green activities visualize external activities which send external events (via the green arrows) to other organizations.

[7] `https://ipfscluster.io/`

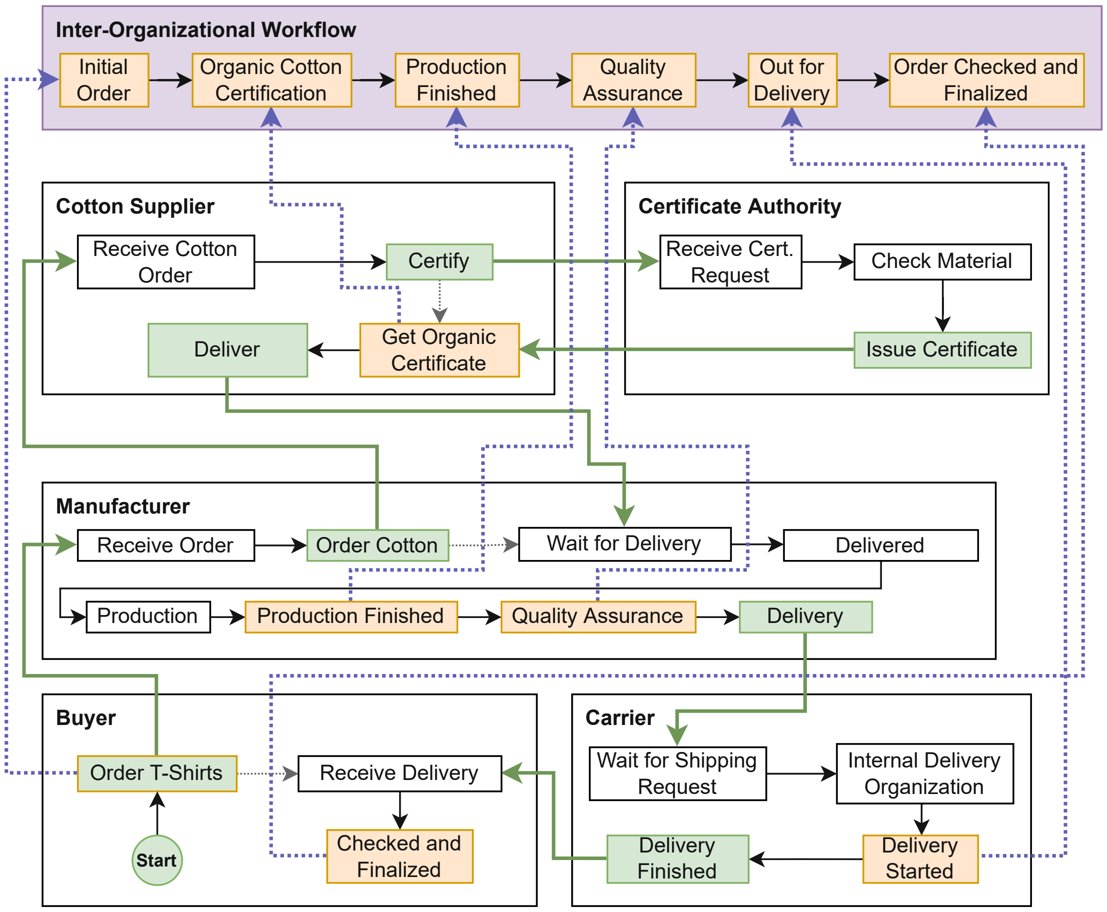


Figure 5.5: Illustration of a Global Activity Tracking example showcasing a SCM process with certified organic cotton T-shirts.

5.2 Prototype

Following our description of the proposed concept from Section 5.1, we implemented a proof-of-concept prototype. Our prototypical implementation is based on the Time-Traveling State Machine (TTSM) prototype from Kleebinder [16]. We are reusing the TTSM prototype as basis published on the code hosting service GitHub[8]. We have published our prototype on GitHub[9] and on Zenodo[10]. In the following, we briefly describe the TTSM prototype from Kleebinder and present our contribution to the TTSM prototype.

The TTSM prototype implemented an approach for verifiable BPM using a time-traveling mechanism. Time-traveling was implemented using an event sourcing mechanism with

[8]`https://github.com/danielkleebinder/ttsm-prototype`
[9]`https://github.com/alexnavratil/ttsm-prototype`
[10]10.5281/zenodo.7779531

custom aggregations that allow specifying a timestamp. By aggregating all events until the specified point in time, the event sourcing mechanism ensures that we get the current state at the given time. Additionally to the time-traveling approach, the TTSM prototype implements custom rules checks and commitments on a blockchain. The prototype consists of the following five modules:

- **Workflow:** The workflow module manages the local workflow design and execution aspect by exposing REST endpoints for creating a workflow, instantiating a workflow, and triggering transitions in a workflow instance.
- **Persistence:** The persistence module connects to an external event storage service and persists all workflow-related events in the corresponding event streams. The module also serves as an event bus, which all other modules can use to react to specific events.
- **Consistency:** The consistency module is responsible for the communication with external organizations. Additionally, a consistency strategy can be configured that commits all external events to a smart contract on a blockchain before sending the events.
- **Rules:** The rules module allows registering custom rule-checking engines. Therefore, the module listens to events on the event bus provided by the persistence module. When a new event is received, it gets forwarded via HTTP calls to all registered rule-checking engines. Based on the rule validation results, a corresponding acknowledgment event gets dispatched back to the persistence module.
- **Integration:** The integration module allows the integration of third-party process engines.

5.2.1 Contribution to the TTSM Prototype

The given TTSM prototype was restricted to a choreography-style workflow execution architecture. In a choreography-style workflow execution architecture, one organization proposes a workflow to all other participating organizations. All organizations are working on the same workflow throughout a business process. In theory, every organization could still employ some internal process steps that are not exposed to the outside. However, all transitions were replicated in the prototype implementation. To implement loosely-coupled workflows, we have refactored the given prototype first. As our research work does not focus on integration with third-party workflow engines, we have disabled the integration module temporarily. The following sections describe our contributions to the TTSM prototype in detail.

5.2.1.1 Workflow Definition Format Converter

The TTSM prototype tightly integrates a finite state machine library called XState[11]. Up until now, workflows have been designed in XState's state machine description format which is very flexible when using JavaScript or TypeScript. The XState format also contains more advanced mechanisms where functions must be supplied, e.g., Actions or Guarded Transitions. Unfortunately, as soon as we use these more advanced mechanisms, we can no longer serialize the format to JSON, because JSON does not support the serialization of JavaScript functions. As our proposed workflow definition format needs these more advanced mechanisms, we implemented a converter that generates XState state machines based on a given workflow definition. In other words, our workflow definition format allows the definition of workflows in a low-code format that can be expanded into XState state machines.

In the following paragraphs, we briefly describe the implementation of the converter for activities and transitions:

Activity Conversion Every activity that is defined in the workflow definition format mainly provides input to how transitions on that activity should be handled. Therefore, the converter takes the activity definition and defines all needed transitions. The converter needs a more complex conversion in case of external activities. The workflow definition format supports the definition of multiple external participants and a condition that decides when an external activity is finished, e.g., how many or which external participants must accept the external event to continue with the local workflow execution. Additionally, the workflow system must handle the responses to external events, i.e., the validation of the event payload using JSON Schema and subsequently assigning properties to the context. Therefore, we model external activities as compound states in XState. Compound states are states that have another state machine embedded, i.e., they have child states and transitions.

In more detail, Figure 5.6 describes the implementation of external activities using compound states. The "Place Order" activity marked in blue contains child states for both external participants "Organization B" and "Organization C". A compound state is marked as finished when all child states have reached a final state. Alternatively, we can supply a custom condition for transitioning to another state that ignores whether the originating state has finished. Therefore, both child states have an internal final state named `$EXTERNAL_RECEIVED_ACK` that gets active when the workflow system receives an acceptance acknowledgment from the corresponding external participant. Each acknowledgment gets converted to an internal event (i.e., events prefixed with a dollar sign) with the name `$RECEIVE_ACK_<externalParticipantId>` with `<externalParticipantId>` being the identifier of the external participant as specified in the workflow definition. Additionally, each transition originating from an external activity gets a custom condition applied that implements the external condition specified

[11]`https://xstate.js.org/`

in the workflow definition, i.e., the condition that specifies which or how many participants must accept the external event.

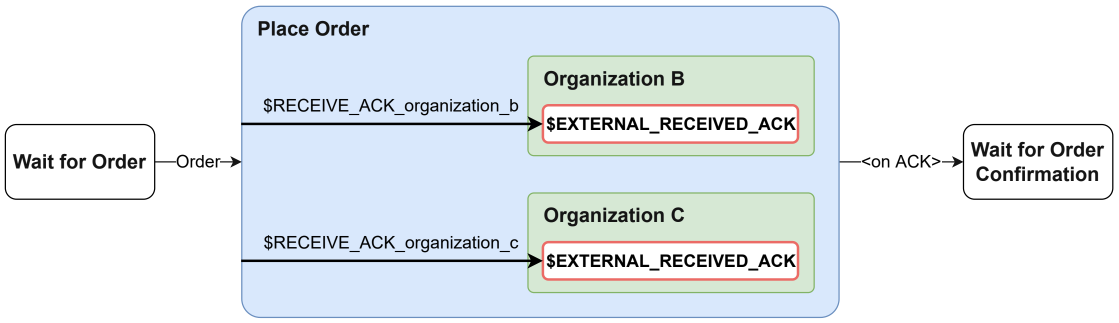


Figure 5.6: Internal state machine representation of external activities using compound states in XState.

Transition Conversion In our proposed workflow definition format, a transition can define a JSON Schema for the incoming event payload. Additionally, the incoming event payload can be assigned dynamically to the workflow instances' context. Therefore, our workflow definition converter must incorporate both functionalities into the state machine definition. XState supports the definition of actions that execute a pre-defined function when a transition is made. We define an action on every transition that checks the incoming event payload against the defined JSON Schema from the workflow definition. Validating a JSON Schema against a JSON object is implemented using the Ajv[12] library. If the validation fails, a corresponding exception is thrown such that the transition gets canceled by XState.

When the validation is successful, our defined action assigns the pre-defined object definition to the context. Therefore, the object definition given in the `assign` field of the workflow definition format is evaluated using the event payload as the evaluation context. Evaluating an object definition requires a recursive algorithm that iterates over all properties and assigns the defined constant value. In case, a JSONPath expression is defined for the property, the expression gets evaluated and assigned to the property. JSONPath expressions are evaluated using the JSONPath Plus[13] library. The evaluation of the object definition outputs an object that we need to merge with the current workflow instance context. Merging also needs a recursive algorithm that does not overwrite too many fields but rather extends the context with the defined properties. We use the library deepmerge[14], which implements a recursive algorithm that fits our requirements.

[12]https://ajv.js.org/
[13]https://github.com/JSONPath-Plus/JSONPath
[14]https://github.com/TehShrike/deepmerge

5.2.1.2 Immutable OrbitDB Event Logs

We extended the TTSM prototype with our proposed communication concept by using a peer-to-peer network as the transport layer. We use OrbitDB[15], an IPFS-based decentralized database. OrbitDB implements a Merkle-CRDT event log as its core persistence layer [89]. On top of the event log, OrbitDB provides different high-level storage types like a key-value store or a document storage. The OrbitDB event log uses the IPFS DAG together with a Publish/Subscribe (PubSub) implementation from libp2p[16]. PubSub is an architecture that allows publishers to send data into a topic. A topic is a group that contains all data published on that topic. Subscribers can then subscribe to a topic and receive notifications for new data in near real-time. Libp2p is the underlying network stack of IPFS. The libp2p PubSub implementation uses the IPFS peer-to-peer network to distribute all the topics and their data. OrbitDB uses the PubSub implementation to replicate events to other peers that subscribed to an event log stream. An event log stream is a PubSub topic that other OrbitDB clients can subscribe to get notified when a new event is published.

The given TTSM prototype previously used an event store database in the persistence module. We refactored the persistence module to implement the strategy design pattern. The refactored TTSM prototype features a strategy that uses the event store database and another strategy that uses OrbitDB event logs as a database. One of the persistence modules' main feature is the event bus, which is provided to all modules. We implemented a wrapper on top of OrbitDB for managing the connections to the event log streams.

Additionally, we faced the issue that OrbitDB does not provide a way to get notified for new events published by other OrbitDB clients. OrbitDB fires the following replication-related events:

- **`replicate`:** The `replicate` event gets fired when another peer starts replicating an event.
- **`replicate.progress`:** The `replicate.progress` event gets fired during the processing of incoming events.
- **`replicated`:** The `replicated` event gets emitted when a replication process has finished, i.e., new events got added to the local replica.

Subscribing to the `replicated` event does not suffice as we cannot identify which events are new. Therefore, we implemented another wrapper on top of OrbitDB's events that implements an algorithm to identify incoming events. Listing 7 shows the relevant code snippet for identifying incoming events. Upon getting a `replicate` event, a new replicate process is started. We need to check that this is the only replication process, as our algorithm is not safe against concurrent replications. The only event that

[15]`https://github.com/orbitdb/orbit-db`
[16]`https://libp2p.io/`

emits the incoming events is the `replicate.progress` event. Thus, we subscribe to the `replicate.progress` event and store the incoming events in a separate array. When the replication process has finished processing all incoming events, i.e., when the `replicated` event fired, we need to sort all stored incoming events by their logical clock value. After that, we can emit the identified incoming events to the subscribers.

```
const eventSubject = new Subject<Event>();
let cachedEntries = [];

db.events.on("replicate", () => {
  if (cachedEntries.length > 0) {
    throw new Error("Concurrent replication detected!")
  }
});

db.events.on("replicate.progress", (address, hash, entry) => {
  cachedEntries.push(entry);
});

db.events.on("replicated", () => {
  cachedEntries.sort((a, b) => a.clock.time - b.clock.time);
  cachedEntries.forEach(entry => eventSubject.next(entry));

  cachedEntries = [];
});
```

Listing 7: Event notification wrapper that identifies incoming events.

Using both implemented OrbitDB wrappers, we implemented a persistence strategy. On initialization, the persistence strategy connects to all event log streams and attaches the corresponding event listeners. The persistence strategy only needs to take care of the local event execution as all external collaboration features are outsourced to the consistency module. When a new workflow or workflow instance gets created, the persistence strategy creates a corresponding event in the *Workflow* or *WorkflowInstances* stream, respectively. Additionally, the OrbitDB persistence strategy defines aggregations for listing all workflows and their instances. The persistence strategy connects to the *LocalEvents* stream and persists all transitions and acknowledgment events. To retrieve the current state of a workflow instance, an aggregation is defined that sums up all events to get to the latest state.

In addition to the persistence strategy, we have implemented a consistency strategy that uses OrbitDB event log streams to communicate with other organizations. This consistency strategy incorporates all external communication as specified in our concept. It mainly needs to connect and subscribe to the *ExternalEvents* streams of all local workflows and workflow instances. However, when sending an event to another organization, the consistency strategy needs to connect to the *ExternalEvents* stream of the receiving organization. OrbitDB does not implement confirmations of receipt, making it hard to

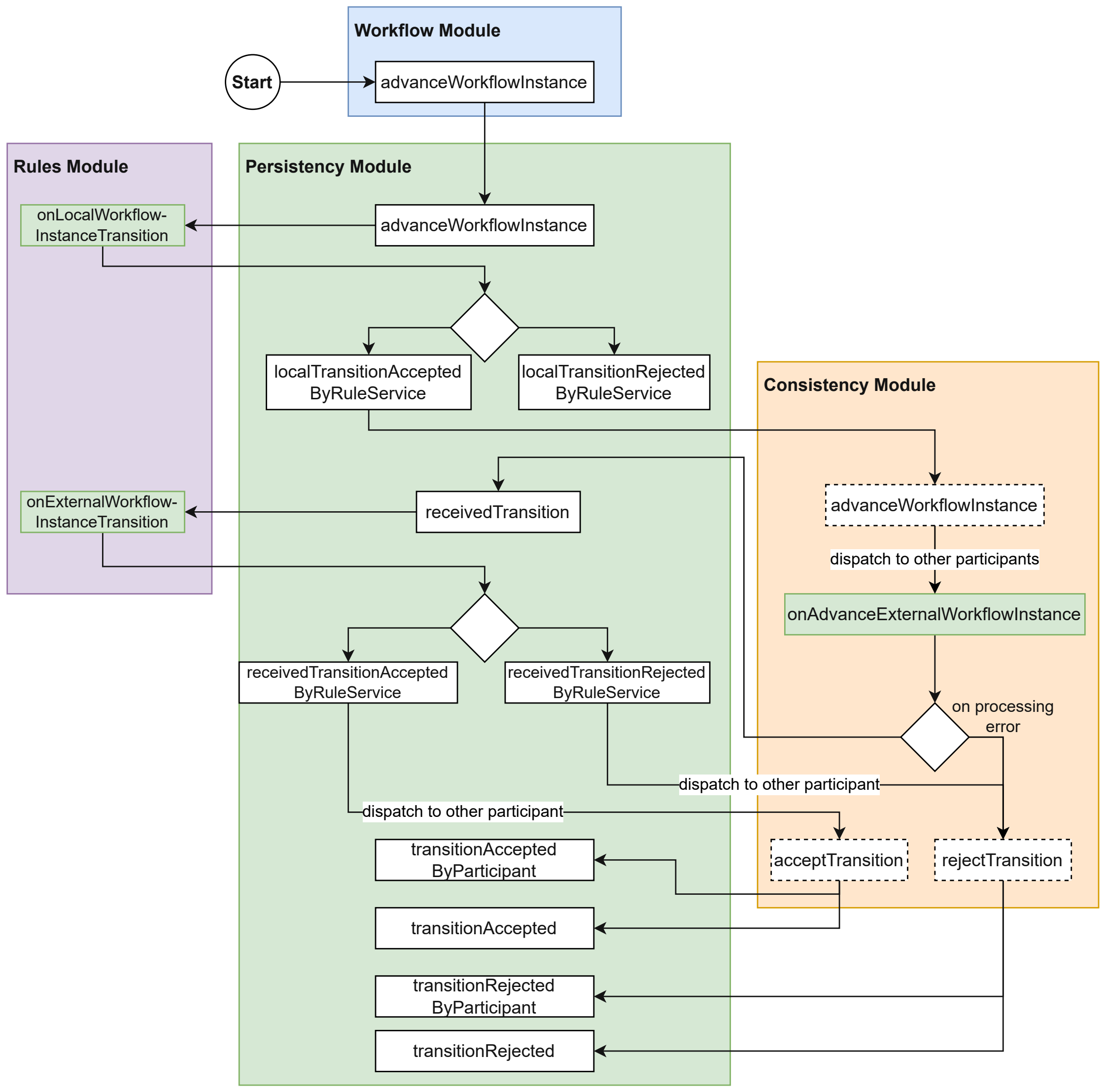

Workflow Module
Start
advanceWorkflowInstance
Rules Module
Persistency Module
onLocalWorkflow-InstanceTransition
advanceWorkflowInstance
localTransitionAccepted ByRuleService
localTransitionRejected ByRuleService
Consistency Module
receivedTransition
onExternalWorkflow-InstanceTransition
advanceWorkflowInstance
dispatch to other participants
onAdvanceExternalWorkflowInstance
receivedTransitionAccepted ByRuleService
receivedTransitionRejected ByRuleService
on processing error
dispatch to other participant
dispatch to other participant
transitionAccepted ByParticipant
acceptTransition
rejectTransition
transitionAccepted
transitionRejected ByParticipant
transitionRejected

Figure 5.7: Event flow in our prototype when an external transition gets executed.

close a connection efficiently. Therefore, we added a workaround that keeps connections open for a pre-configured timespan, in hope that the event gets replicated by the receiving organization during this timespan. In case the event does not get replicated during this grace period, the event is still stored in the local replica of the event log. The external participant does not notice that it missed an event. When connecting again to the remote event stream replica, OrbitDB will try to replicate the missed event. However, there is currently no feature implemented in the prototype that retries connecting to remote event logs with non-replicated (i.e., missed) events.

When the consistency strategy receives an event that targets a local workflow or workflow instance, the event gets processed and checked by the registered rule engines. After the rule engines have finished all checks, a corresponding acknowledgment event is sent back to the originating organization. Additionally, the defined actions on the XState state machine are executed, i.e., the validation of the event payload and assignments of properties to the context. Figure 5.7 illustrates the event flow in the prototype when executing an external transition in the local workflow. The event flow diagram shows the interaction between all modules. The white boxes depict persistence events, while the white boxes with a dashed border depict consistency events that are exchanged between two organizations. The green boxes show essential functions that process events in the prototype.

5.2.2 Research Question 3

In this section, we answer **RQ3**:

> *Which technologies and architecture can be leveraged to implement the defined requirements?*

In Section 5.2, we described the implementation of a proof-of-concept prototype. Our prototype is based on the TTSM prototype by Kleebinder [16]. We have split our contribution to the prototype into the two aspects defined in the requirements analysis phase. The first contribution was to refactor the prototype to support the execution of loosely-coupled workflows, as our concept proposes. We tackled the refactoring by implementing our workflow design and execution concept. The local workflow execution uses the XState finite state machine. The XState state machine provides many customization options, which we need, to convert our workflow definition format into a state machine. The conversion process is described in detail in Section 5.2.1.1.

The second contribution was to implement the inter-organizational communication aspect as defined in our concept. We decided to use OrbitDB as a decentralized database. OrbitDB uses IPFS and libp2p as the network layer to replicate a Merkle-CRDT-based immutable event log to all participants. The TTSM prototype uses the strategy design pattern to exchange specific parts of the prototype. We implemented two strategies in the prototype:

1. **Persistence Strategy:** The persistence strategy is responsible for persisting all local workflow execution events. Additionally, the persistence strategy provides aggregations to list all workflows, workflow instances, and the current state of a workflow instance.

2. **Consistency Strategy:** The consistency strategy is responsible for the bi-directional communication with external organizations. It additionally handles the commitments to the blockchain.

The description of the prototype implementation answers the third research question.

CHAPTER 6 Evaluation

In this chapter, we evaluate the concept and the prototype in two parts:

1. In Section 6.1, we ensure the correct implementation of the defined requirements from Chapter 4 by reasoning on each taken decision and mechanism in our concept from Section 5.1.

2. Section 6.2 implements a real-world business process. The implementation empirically evaluates that the identified requirements and the implemented concept solve the problem described in Section 1.1.

As pointed out initially in Section 1.3, the evaluation is part of the research methodology of this thesis, i.e., Design Science by Hevner et al. [11]. Finally, to improve the proposed requirements and the concept in future work, we reflect on limitations and discuss possible future work in Section 6.3.

6.1 Fulfilment of Requirements in Concept

In this section, we evaluate our proposed concept from Section 5.1 in terms of fulfilling the requirements. We conduct a qualitative evaluation in a descriptive form using informed arguments as proposed in the Design Science research methodology by Hevner et al. [11]. Therefore, we additionally conduct and discuss approaches from related works. The following paragraphs reference the related requirements from Chapter 4.

WFL-REQ 1: Preservation of Organization Internals The concept uses loosely coupled workflows that split the overall process into multiple decoupled local processes. These local processes are maintained and executed independently on the infrastructure of the respective organization. Thus, the concept separates the local processes and preserves the privacy of each participating organization. Therefore, Van der Aalst et al. [80] proposed the loosely coupled architecture concept to solve the problem defined in this requirement.

WFL-REQ 2: Flexibility in the Complexity of the Global Business Process The proposed workflow definition format provides simple mechanisms allowing to build arbitrary complex workflows. Each external activity can define multiple external organizations that are contacted via events. These events can be dynamically constructed using data submitted to the workflow.

We defined two *ExternalEvents* event log streams, which are bound to the workflow and the workflow instance. The *ExternalEvents* stream bound to the workflow allows triggering a new workflow. In contrast, the *ExternalEvents* stream that is bound to the instance provides a way to collaborate with an existing workflow instance. Similarly, the BPMN specification introduces a concept called correlation [90]. Correlations must be defined for asynchronous messages such that they can be assigned to the corresponding process instance. Corradini et al. [91] generalized the correlation mechanism to existing and possibly new instances.

WFL-REQ 3: Extensible and Flexible Workflow Definition The concept proposes some easy-to-implement mechanisms that we even re-used throughout the concept. For example, one can define an inter-organizational choreography-based workflow consisting of external transitions. We re-used the structure to define external participants and its mechanism to expose certain activities to the inter-organizational workflow. The inter-organizational workflow allows all organizations to control the business process choreography independently. Thus, extending our concept with global activity tracking capabilities was possible without inventing new mechanisms.

WFL-REQ 4 – WFL-REQ 7: Activities, Events, Transitions and Event Payloads Our proposed workflow definition format allows defining activities connected via transitions. Events trigger these transitions. Transitions can be marked as external, which exposes them to be triggered by a foreign organization. Activities can also be marked as external to collaborate with other organizations. External activities include a mechanism to dynamically build event payloads that target a specific workflow of another organization. We can validate the schematic correctness of payloads along with simple validation rules by including a payload validation mechanism based on JSON Schema. Received event payloads can be dynamically assigned to a storage component called the context, which is bound to the workflow instance.

Kirchoff et al. [92] proposed a similar mechanism for defining service interfaces in process-driven decision support systems. The fact that their proposed system is designed for collaboration with external third-party services strengthens the need for validations. Errors in the data could potentially make the system's data flow prone to errors without employing appropriate validations. The researchers state that errors delay making decisions. Furthermore, and even worse, unperceived errors could lead to wrong or suboptimal decisions. The same risk applies to business process management systems in general.

WFL-REQ 8: Data-based Routing The concept supports data-based routing using different mechanisms based on the use case:

- **Junctions:** Activities can be used to add junctions to the workflow. The workflow can take different routes by providing conditions based on the JEXL expression language. The JEXL expression language can use properties from the workflow instances' context, which makes data-based routing possible.
- **External Participants:** External activities can define multiple external participants. The recipient information can re-use properties from the workflow instances' context. Therefore, we use JSONPath expressions to reference properties in the context. The context gets filled with submitted properties contained in event payloads.

Data-based routing is an important concept when autonomously executing business processes. In addition to improving the execution of business processes, data-based routing helps to define formal semantics on the data flow. Corradini et al. [91] proposed a formalization on BPMN choreographies which heavily relies on data-based routing. Data-based routing intrinsically assumes the existence of schema validations on event payloads as defined in WFL-REQ 7. Their proposed concept incorporated a JavaScript-based expression language to support data-based routing. We explicitly consider this expression language too powerful as it allows for non-deterministic decisions, i.e., using randomness or side effects. Therefore, the usage of the JEXL expression language fits better with business processes as it works using a closed-world assumption with data that is already available in the business process. Thus, our concept uses the JEXL expression language to ensure a transparent process execution.

WFL-REQ 9: Custom Rule Validation for every Action We used a concept proposed in the research work from Kleebinder [16]. An external rule-checking engine can be registered that validates pre-defined rules. Our concept triggers the rule-checking engine on each transition between activities. Soleymanzadeh et al. [93] proposed the usage of JsonLogic[1] to implement custom validation rules for business processes. In general, having a deterministic set of validation rules makes the whole business process more transparent such that every business partner can inspect rejection decisions.

COM-REQ 1: Messaging Technology Compatibility The workflow definition format provides a way to define so-called connectors. These connectors provide compatibility

[1] `https://jsonlogic.com/`

with multiple messaging technologies. Hence, making the transmission technology independent of the concept and shifting the responsibility to the collaborating organizations. In the past, several research works investigated service-oriented architectures together with a web services-based approach for BPM [91, 94]. Every organization implements web services for its business processes. Communication between the web services is achieved through HTTP requests. Adapting the existing web service-based approach to peer-to-peer messaging is an interesting opportunity for future work.

COM-REQ 2: Decentralized and Asynchronous Message Exchange The proposed communication concept uses CRDT-based event logs replicated over a peer-to-peer network. This type of communication is asynchronous and decentralized. We do not put any further constraint on the used peer-to-peer network other than what we defined in the requirements. Utilizing peer-to-peer networks perfectly fits the chosen LCA. Molina-Jimenez et al. [95] proposed a similar concept while focusing on RosettaNet Partner Interface Processs (PIPs). Instead of utilizing BCT to achieve a consistent holistic view of a business process, they implemented a 3-way handshake protocol that gets executed at the end of a business process. Their proposed concept ensures that both parties have a consistent view of the business process outcome. Ensuring a consistent view of the outcome only at the end of the process is not an efficient approach in terms of having early feedback. As business processes can last over a longer time, ensuring that every message exchange is leading to a consistent view, as defined by our proposed concept, is an important aspect.

COM-REQ 3: Identifyable and Addressable Organizations We have incorporated this requirement into our connector mechanism. Therefore, identifying and addressing organizations, their workflows, and instances solely relies on the connector used to communicate with an organization. As mentioned before in requirement WFL-REQ 2, the BPMN specification proposes a similar concept called correlations [90]. In our prototype, the workflow instance identifier is exchanged in addition to the payload such that the receiving party can route the message to the referenced process instance.

COM-REQ 4: Secure Message Exchange Ensuring a secure message exchange builds on two pillars. Firstly, the event log ensures that all events are unique by digitally signing and chaining them, i.e., each event references its predecessor in the event log through its hash. Secondly, an organization can commit to an event by publishing a commitment using BCT. Commitments allow all other organizations to verify an event's occurrence and make the event non-repudiable.

Müller et al. [2] classified different patterns that utilize BCT. In our concept, we have mixed three different patterns and combined them with peer-to-peer communication while focusing on using only subtle aspects of BCT. We combined the identified patterns "*Blockchain as a Tamper-Proof Hashed Data Storage*" together with "*Blockchains as a Transparent Process Event Log*" and "*Blockchain-Based Business Process Engine*". Instead of encoding the business processes completely on a blockchain, we incorporated peer-to-peer messaging technologies together with CRDTs to accomplish similar consistency guarantees. This combination of different technologies allows our concept to get rid of disadvantages (related to e.g., privacy, efficiency and costs) introduced by BPM engines which are entirely based on BCT.

COM-REQ 5: Commitments with Verifiable Timestamps We have used a concept proposed by Kleebinder [16] that makes commitments to a blockchain. BCT provides a way to find a consensus on a state which we can utilize to get a relative ordering between any two events. Commitments with BCT allow us to prove that an event has occurred and even provide consensus on when an event has occurred. Thus, commitments can assist the CRDT-based event log to find a total ordering of events in case of conflicting concurrent events.

Blockchains also provide verifiable timestamps with a certain precision. The achievable precision largely depends on the used BCT and has to be decided for the specific use case. An example of timing-related constraints is defined in the RosettaNet PIPs [95]. There, external organizations are expected to respond in a certain amount of time (e.g., 2 or 3 hours).

Our proposed concept adheres to the BCT pattern "*Blockchain as a Tamper-Proof Hashed Data Storage*" described by Müller et al. [2].

COM-REQ 6: Mailboxing We are using a peer-to-peer network as our transport layer. Requiring a mailbox service must be satisfied by the used peer-to-peer network as defined during our requirements analysis. Molina-Jimenez et al. [95] even loosened this requirement such that both parties do not need to be online at the same time in LCA-based process engines. The fact that all parties and their processes are loosely coupled does not strictly require them to collaborate in a synchronous messaging style. Nevertheless, missing a mailbox service makes executing inter-organizational business processes not practicable and possibly slows down reaching a common goal.

COM-REQ 7: Acknowledgements on all Levels of the Message Exchange Every organization must send a receipt confirmation upon receiving an event. Additionally, an acknowledgment event must be sent to the originating organization after an event is processed. A receipt confirmation must even be sent by the mailbox service, at least when the organization is offline. Implementing a mailbox service is the responsibility of the used messaging connector.

RosettaNet PIPs similarly require acknowledgments by defining two types of messages: [95]

1. **Business Action Messages** are used to trigger a specific event in the business process of another party, like ordering articles.

2. **Business Signal Messages** are responses to action messages which are also called acknowledgments.

COM-REQ 8: Privacy-Preserving Message Exchange Our concept is based on loosely coupled workflows that combine internal processes executed locally at their owning organization. The LCA allows us to model workflows in a privacy-preserving way. Additionally, we are using a peer-to-peer network as the transport layer. As the concept does not define a specific peer-to-peer network, we require it to preserve the privacy of all exchanged events. Typically, preserving the privacy of events is hard to achieve with public peer-to-peer networks, but their private variants should allow that, e.g., IPFS cluster for IPFS.

COM-REQ 9: File and Document Exchange Exchanging files is possible by using an external peer-to-peer storage component like IPFS. The used external peer-to-peer storage component must be compliant with our defined requirements. Using a content-addressable storage allows files to be linked directly inside the events. The file's reference is mainly based on a cryptographic hash that implicitly ensures the integrity of the file's content.

COM-REQ 10: Serializability Ensuring consistency works in two phases. Firstly, we use a CRDT-based event log that ensures eventual consistency and provides a relative ordering of the event while still allowing concurrent events. Therefore, the used CRDT manages a logical clock that provides a total ordering of the events as long as there are no concurrent events. In the case of concurrent events, the CRDT ensures that all replicas have a consistent view of the data by ordering concurrent events based on the cryptographic hashes of the events. Additionally, our concept uses commitments to a blockchain, which provides verifiable timestamps and a consensus on the total ordering of events. By combining both approaches, we achieve a total ordering of all events. Therefore, we informally argue that our concept provides a guarantee for serializable consistency. Conducting a formal proof is out-of-scope for this work.

COM-REQ 11: Global Tracking of Activities In Section 5.1.2.5, we introduced our approach to extending the concept with global activity tracking capabilities. Therefore, an inter-organizational workflow that spans all collaborating organizations must be modeled (and agreed on) upfront. This inter-organizational workflow can then be collaboratively executed by all local processes.

Global tracking of activities was described to be an essential part of inter-organizational business processes in the research work by Sturm et al. [10]. Modeling the global workflow is a separate challenge that is out-of-scope for this work. Nevertheless, employing and maintaining a global workflow opens entirely new doors for inter-organizational BPM. Apart from giving every participating organization a holistic view of the processes current state, it allows them to apply custom validations or retrospectively analyze the efficiency of the process, e.g., finding bottlenecks in the overall process.

6.2 Concept Evaluation

In this section, we evaluate the concept based on the implemented prototype from Section 5.2 by conducting an empirical test against a real-world problem. We tested the concept against the defined requirements in the previous section. A common business process is a simple ordering process between a buyer and a seller. Such an ordering process is standardized in the Peppol Business Interoperability Specification[2]. This specification is maintained by OpenPeppol[3], a non-profit association whose aim is to standardize and improve procurement processes with European public sector buyers.

The ordering process consists of two parties, the buyer and the seller. Both parties exchange messages in the form of UBL[4] (Universal Business Language) documents. The party only sends an order request message that instantiates a new ordering process at the seller. The seller then must decide on how to handle the order request, as it is possible to either accept it fully, accept it only partially or fully reject the order. Depending on the chosen action, a corresponding order response message is generated by an external system and attached to the response message sent to the buyer. The buyer then updates its local state and the process completes for both parties. Figure 6.1 visually depicts the ordering process in the BPMN format.

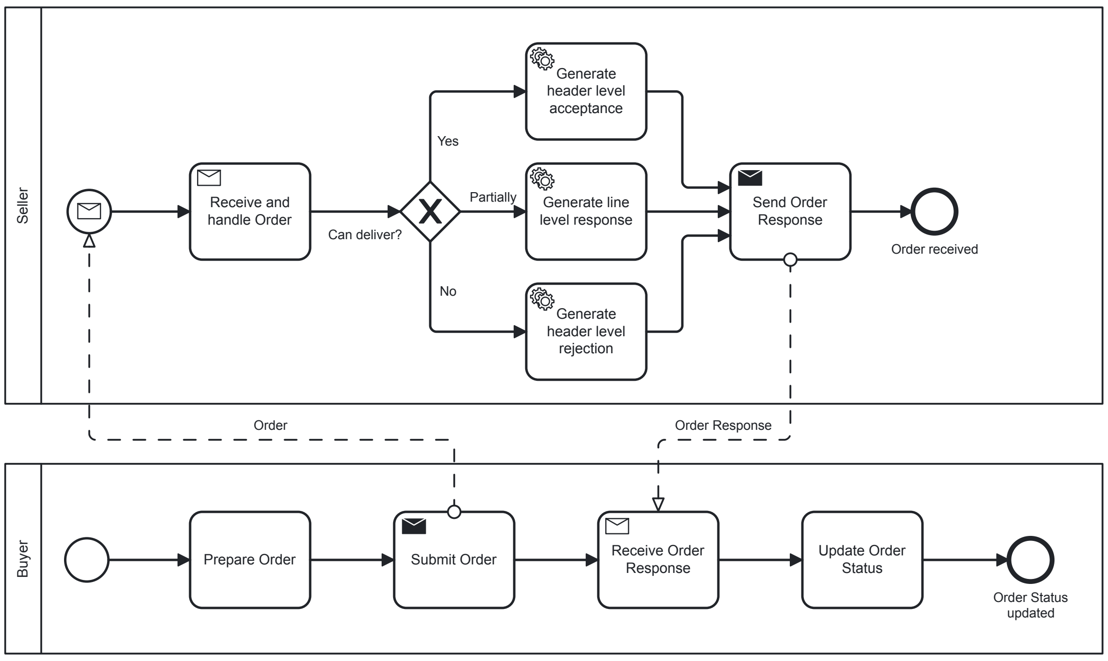


Figure 6.1: BPMN diagram defining the Peppol Ordering process from the Peppol Business Interoperability Specification.

[2]https://docs.peppol.eu/poacc/upgrade-3/profiles/28-ordering/
[3]https://peppol.eu/about-openpeppol/
[4]http://docs.oasis-open.org/ubl/UBL-2.1.html

6.2.1 Business Process Modeling Evaluation

To implement the defined BPMN-based ordering process from Figure 6.1, we need to convert the business process manually into our defined workflow format introduced in Section 5.1.1. The process consists of two parties, the seller and the buyer. Both parties operate their local process that collaborates with the local process of the opposite party.

Our prototype does not fully implement the concept. Therefore, we had to find compromises in the modeling of the workflows. The seller's local process utilizes service tasks to generate the corresponding UBL documents based on the outcome of the decision gateway. As service tasks are not supported by our concept, we expect some external document generation service to listen to every workflow instance of the ordering process. Upon reaching one of the service tasks, the document generation service should send an event with the corresponding document attached as IPFS CID. Additionally, our prototype does not support gateways, therefore, we replaced the gateway with a user-based decision, i.e., the user can decide by submitting a corresponding `YES`, `NO` or `PARTIALLY` action on the "Receive and handle Order" activity. The resulting workflow definitions for the buyer's and seller's local processes can be found in the published repository[5].

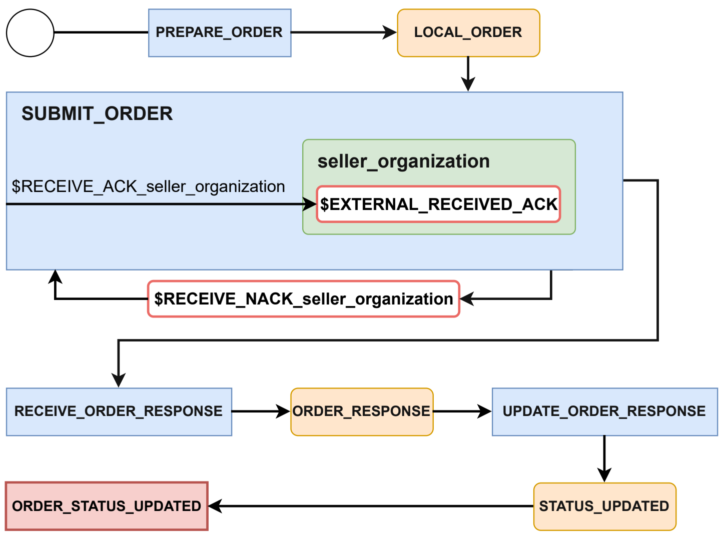
Finally, we need to convert the given workflow model into a state chart definition that incorporates all execution semantics, i.e., validation, assignments and data-based routing. Figures 6.2a and 6.2b show the state charts for the buyer's and the seller's local process, respectively.

6.2.2 Business Process Execution Evaluation

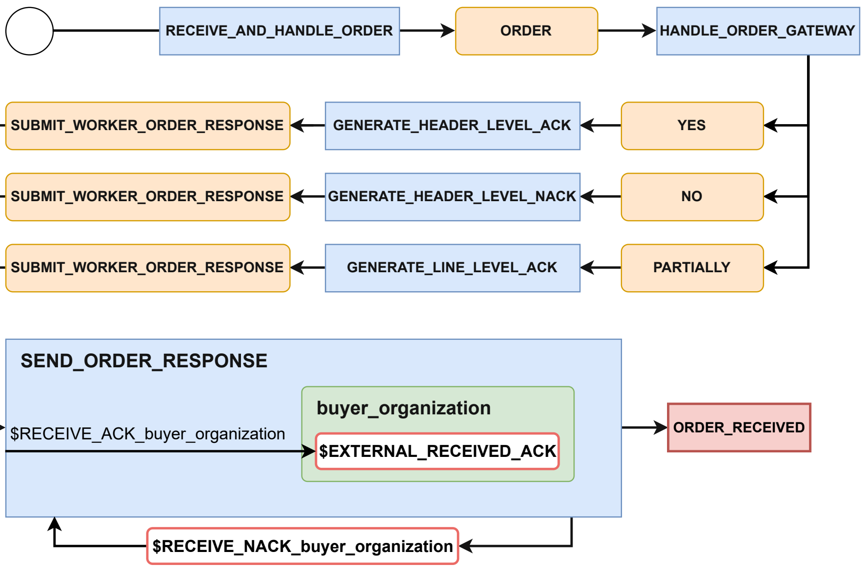
To show the utility of our business process execution concept, we evaluated the ordering process by constructing a set of events that navigate through a successful order. First, the buyer submits a `LOCAL_ORDER` event to its local process. The `LOCAL_ORDER` event example is given in Listing 8 which orders 6 Pen articles. In our workflow definition, we added a corresponding JSON Schema validation and a context assignment definition that assigns the order to the local context. In the subsequent activity, shown in the state chart from Figure 6.2a, the external activity called `SUBMIT_ORDER` gets active which contacts the seller's organization.

As our business process defines that a new workflow instance gets instantiated at the seller, we limit the recipient information to an `organizationId` and a `workflowId`. When a new workflow instance gets instantiated at the seller, the seller process handles the incoming `ORDER` event. As the seller's state chart from Figure 6.2b shows, the human worker needs to decide whether to accept, partially accept or decline the order by sending the respective `YES`, `PARTIALLY` or `NO` event. Depending on the corresponding decision outcome, the seller's document generation service submits an IPFS CID to the process which links to the generated order response UBL document. Listing 9 gives an example of the document submission event.

[5]10.5281/zenodo.7779531



(a) State Chart of the buyer's local process


RECEIVE_AND_HANDLE_ORDER
ORDER
HANDLE_ORDER_GATEWAY
SUBMIT_WORKER_ORDER_RESPONSE
GENERATE_HEADER_LEVEL_ACK
YES
SUBMIT_WORKER_ORDER_RESPONSE
GENERATE_HEADER_LEVEL_NACK
NO
SUBMIT_WORKER_ORDER_RESPONSE
GENERATE_LINE_LEVEL_ACK
PARTIALLY
SEND_ORDER_RESPONSE
buyer_organization
$RECEIVE_ACK_buyer_organization
$EXTERNAL_RECEIVED_ACK
ORDER_RECEIVED
$RECEIVE_NACK_buyer_organization

(b) State Chart of the seller's local process

Figure 6.2: State Charts resulting from the modeled workflow definitions.

Finally, the seller's process sends an ORDER_RESPONSE event back to the buyer's process. The buyer processes the incoming order response and updates its local status.

```json
{
  "event": "LOCAL_ORDER",
  "payload": {
    "organizationId":
↪   "0341ee6923e6e7fdc09e055e74b6a207e4b8e04b022c1432f084a2959782a54c75",
    "workflowId": "aee843ae-4f83-47c0-8247-403d1dc0b54a",
    "quantity": 6,
    "articleNumber": "24574567",
    "articleName": "Pen"
}}
```

Listing 8: Example of an ORDER event that the buyer submits to the local process defined in the state chart from Figure 6.2a.

```json
{
    "event": "SUBMIT_WORKER_ORDER_RESPONSE",
    "payload": {
        "orderResponseCid": "IPFS-CID-PLACEHOLDER"
}}
```

Listing 9: Example of a SUBMIT_WORKER_ORDER_RESPONSE event that the seller's document generation service submits to the seller's local process defined in the state chart from Figure 6.2b.

The given example shows the utility of our requirements, altogether with the concept and the implemented prototype. Nevertheless, we have also identified some limitations and future work which we discuss in Section 6.3.

6.3 Limitations and Future Work

This section concludes the qualitative evaluation. Although the evaluation of the requirements and concepts show the real-world utility of our research, we also identified some limitations that we discuss in this section.

BPMN Integration In Section 5.1.1, we introduced our custom workflow definition format that incorporates the previously identified requirements. We have chosen to not use any pre-defined standards like BPMN to focus on our requirements, and not to fixate on the constraints of an existing format. Using this approach helped us to think more creatively about new ways to define and execute inter-organizational business processes. Nevertheless, to establish our concept for real-world production use cases, it is important to integrate our concepts into an established standard, e.g., BPMN. The integration needs more extensive research to identify how our proposed concepts fit best into the existing concepts in BPMN.

Reliability of Transport Technology Our approach currently relies on the reliability of the chosen transport technology. In our prototype, we have chosen OrbitDB which uses a peer-to-peer PubSub mechanism. However, the PubSub implementation is still an experimental feature and does not guarantee the reliable delivery of messages. In general, reliable transport technology is important in business processes. Our approach incorporates commitments using BCT which would help to ensure the delivery of messages, but we did not use this guarantee in our concept actively. Therefore, we propose integrating a mechanism for handling lost messages into our concept.

Consistency Guarantees of our Concept In Section 5.1.2, we discussed our concept on inter-organizational execution that incorporates immutable event logs using CRDT and commitments using BCT. These concepts implement the "Serializability" requirement defined in COM-REQ 10. However, proving that our concept satisfies the serializability requirements requires a formal proof of the consistency properties. Therefore, our concept is currently limited to our informal argumentations given in Sections 5.1.2 and 6.1.

Extending the Research Area to Additional Real World Scenarios Our concept mainly focused on the implementation of a SCM business process. SCM processes cover a large range of requirements on business processes in general. Therefore, applying our concept to other common business processes should be possible. Nevertheless, we propose to incorporate additional real-world processes to extend the research perspective. Additionally, we focused on certain aspects of business processes that would need to be extended in future work.

Throughout our evaluation, we identified possible future requirements. Molina-Jimenez et al. [95] mentioned the need for timing constraints in business process execution, i.e., certain responses are expected to occur in a specific timeframe. Therefore, extending our validation mechanism to allow for timing-constraint checks would be an interesting addition to our concept.

BPMN defines so-called data stores that allow sharing of data between multiple processes and process instances. For example, a data store might be a database that stores all invoices or orders. These data objects are needed by multiple processes, e.g., in processes for ordering and handling the return of items. Therefore, it is worth evaluating whether shared data stores should be directly integrated into the concept. Alternatively, implementing shared data stores is possible with external service workers that interact with the business process and submit required data, like oracles do in BCT. The main risk of implementing a shared data store into our concept would be that the business process execution is getting too complex.

Throughout our evaluation, we identified that many business processes still rely on the exchange of XML-based documents. An example would be UBL documents for exchanging orders or invoices, as described in our evaluation. Our concept does not directly specify the format of event payloads and we focused only on JSON documents in our prototype. Therefore, the concept would need additional requirements on event payloads regarding document formats. Directly supporting multiple document formats brings additional advantages as all other mechanisms (i.e., validations, data-based routing) would apply to all supported document formats.

Confidentiality in Global Tracking of Activities In Section 4.2.2, we mentioned that ensuring confidentiality and global tracking of activities at the same time need additional research. Therefore, our concept is limited in that regard and implements global tracking of activities without further confidentiality constraints. Possible solutions would need to incorporate additional mechanisms like Zero Knowledge Proofs which have been out of scope for this work. As global tracking of activities is an important feature of inter-organizational business processes, it is important to put more research efforts into this direction.

Investigation of Further Related Projects Our prototype described in Section 5.2 focused on using OrbitDB with PubSub on IPFS as underlying transport technology. The area of peer-to-peer technologies is shaping rapidly. Thus, extending our research to other peer-to-peer technologies would be important for future work. By using IPFS, we have chosen a fundamental technology that influences many other projects. For example, Fireproof[6] or the Web Native File System[7] present a promising approach. Both projects build upon IPFS as storage backend and add additional functionality and mechanisms, like Encryption or Self-Sovereign Identity, on top of IPFS.

[6]`https://fireproof.storage/`
[7]`https://github.com/wnfs-wg`

CHAPTER 7

Conclusion

BPM is essential in all organizations, specifically when collaborating with external business partners. Organizations rely mainly on solutions that incorporate a trusted intermediate party to execute external processes. As such an intermediate party incurs a trust relationship, it represents a single point of failure that organizations want to eliminate. In this research work, we have identified alternative solutions that do not rely on a trusted intermediate party. Instead, many previous research works identified BCT as a promising component that solves many problems in BPM.

In this research work, we used the Design Science research methodology by Hevner et al. [11]. Therefore, we conducted two narrative literature reviews to gain the necessary background knowledge and to research related works. Next, we employed a requirements analysis phase based on a defined vision. The vision helped to structurally find essential requirements and iteratively work towards the target vision.

We have split the requirements into two aspects: *Workflow Design and Execution* and *Inter-Organizational Communication*. The *Workflow Design and Execution* requirements focused to a large extent on how to structure and define an inter-organizational workflow. We developed a custom workflow definition format to focus on the requirements, which also helped to find novel approaches. The workflow definition format allows modeling a workflow and the data flow while being able to define simple validation rules and implement data-based routing. Data-based routing allows transparently taking decisions based on previously submitted and processed data. Furthermore, we proposed a mechanism for dynamically generating event payloads using submitted data.

We also defined that inter-organizational workflows must be split into loosely coupled local processes that collaborate via a pre-defined communication channel. Loosely coupled business processes help to preserve the privacy of all collaborating organizations, which we identified as a common problem in BPM systems that solely rely on BCT.

In addition to our concept targeting the *Workflow Design and Execution* aspect, we also proposed a concept for the *Inter-Organizational Communication* aspect. Communication between two organizations works by exchanging messages in the form of events. Therefore, we proposed to employ immutable event logs as a data store for exchanging events between two business partners. It is important that the events are in a totally ordered sequence. Using a CRDT, it is possible to replicate the event logs over a peer-to-peer network and to ensure that both business partners have a consistent view of the data. Additionally, every business partner must commit to every sent event by posting a hash-based commitment onto a blockchain. The receiving business partner can consequently use this commitment to verify the validity of received events. Overall, our concept allows for modeling and executing inter-organizational business processes using peer-to-peer networks and BCT.

In order to prove the practical utility of our concept, we implemented a prototype and executed a real-world SCM business process. During our research, we also identified known limitations and potential future work. Apart from integrating our concepts into existing standards, like BPMN, a formal proof of the consistency guarantees of our CRDT-based event log approach would be an interesting topic for future work. In addition to that, we have identified BPM to be a large and complex topic that demands focusing on specific aspects. Therefore, extending our research to other aspects and real-world scenarios could help improve the identified requirements and our proposed concept. Furthermore, extending our research into the area of global activity tracking could help to identify further requirements, specifically toward ensuring the confidentiality of inter-organizational business processes. Peer-to-peer technology is evolving quickly, resulting in new mechanisms, solutions, or libraries popping up regularly. Therefore, researching additional peer-to-peer approaches would be an interesting contribution.

APPENDIX A

BPM Literature Review

The search query in Listing 10 targets all papers about inter-organizational business processes that additionally contain the words *blockchain* and *data* or *storage*. The latter search terms already restrict the results to contain only the most relevant papers for the topic of this thesis. Furthermore, the search query limits the paper's publication year to 2018 and later.

```
TITLE-ABS-KEY (
    inter-organizational AND business AND
    process AND blockchain AND ( data OR storage )
) AND (
    PUBYEAR = 2018 OR PUBYEAR > 2018
) AND (
    EXCLUDE ( DOCTYPE , "cr" )
) AND (
    LIMIT-TO ( LANGUAGE , "english" )
)
```

Listing 10: Search query targeting the integration of BCT with BPM focusing on storage approaches

The search query from Listing 10 resulted in 24 search results which are listed in table A.1. The table also lists whether some paper was excluded due to not being relevant to this thesis.

#	Title	Included
1	Towards Blockchain Support for Business Processes [18]	Yes
2	Blockchain as a platform for secure inter-organizational business processes [6]	Yes
3	Towards inter-organizational business process governance through blockchain [96]	No
4	Business Process Redesign Heuristics for Blockchain Solutions [97]	No
5	An architecture for multi-chain business process choreographies [98]	No
6	Blockchain for data sharing in the rational use of coastlines and seaport demands in inter-organizational networks: Development of a new intelligent decision support system [99]	No
7	An ontological analysis of artifact-centric business processes managed by smart contracts [100]	No
8	Upgradeability Concept for Collaborative Blockchain-Based Business Process Execution Framework [101]	No
9	Blockchain-Based Traceability of Inter-organisational Business Processes [22]	Yes
10	A simulation-based and data-driven framework for enabling the analysis and design of business processes based on blockchain and smart contracts solutions [102]	No
11	Blockchain-oriented Inter-organizational Collaboration between Healthcare Providers to Handle the COVID-19 Process [103]	No
12	Decentralized Control: A Novel Form of Interorganizational Workflow Interoperability [9]	Yes
13	Decentralized business process modeling and instance tracking secured by a blockchain [104]	No
14	Towards modeling privity and enforceability requirements for BPM based smart contracts [105]	Yes
15	Blockchain-based controlled information sharing in inter-organizational workflows [24]	Yes
16	Interpreted execution of business process models on blockchain [106]	Yes
17	Modeling and Enforcing Blockchain-Based Choreographies [20]	Yes
18	Blockchain support for execution, monitoring and discovery of interorganizational business processes [107]	No
19	Balancing Privity and Enforceability of BPM-Based Smart Contracts on Blockchains [23]	Yes
20	MAS-Aided Approval for Bypassing Decentralized Processes: An Architecture [108]	No
21	The role of the CFO of an industrial company: An analysis of the impact of blockchain technology [109]	No
22	Designing a Collaborative Construction-Project Platform on Blockchain Technology for Transparency, Traceability, and Information Symmetry [110]	No
23	Interorganizational process execution beyond ethereum: Road to a special purpose ecosystem [10]	Yes
24	Institutional and evolutionary aspects of modern interactions in transport and logistics systems [111]	No

Table A.1: Search results from 9.7.2022 of query from Listing 10

APPENDIX B

Distributed Off-Chain Storage Incentivization

Incentivization mechanisms are an essential component of distributed off-chain storage networks. They ensure that storage nodes are incentivized for their effort and give storage customers some guarantee that their data is stored safely. We gave an overview of incentivization mechanisms in Section 3.5. In this chapter, we describe the incentivization mechanisms of Filecoin and Swarm in more detail. Additionally, we give an overview of Storj's incentivization mechanism.

B.1 Incentivization in Filecoin

Filecoin is a peer-to-peer data network with a storage market that uses the IPFS network in the background. To prove the physical storage of data, Filecoin uses an algorithm called Proof-of-Replication. It is a derived form of a Proof-of-Space, which is also often called *useful proof of space*. The more advanced and currently used implementation can be looked up in the technical report by Fisch et al. [112]. Although the in-depth description of the current implementation in Filecoin is out of scope, we explain a basic version of Filecoin's incentive mechanism next to get an idea of how Proof-of-Storage systems work.

The basic version of Proof-of-Replication stores a special encoding of a file instead of the actual file. This special encoding is slow to compute but fast to decode. The goal of this incentive is to give an adversary no other option than storing the slow-to-compute encoding and blocking any attempts to retrieve the data from another node. The Proof-of-Replication algorithm in Filecoin uses a so-called Verifiable Delay Encoding (VDE), a special form of Verifiable Delay Functions (VDFs). VDFs are typically used in Proof-of-Space algorithms to prove that a specific time has passed.

The proving phase works in a challenge-response manner. In the challenge phase, the verifier challenges the node by querying a random set of block encodings. The proofer then has only a little time to respond with the encoding. This timespan must be as small as possible, so the proofer cannot reencode the file or retrieve the data from another node. Upon receiving the proof in encoded blocks, the verifier can decode the blocks and check the correctness of the encodings.

To ensure the storage of files over a longer period, Filecoin employs a Proof-of-Spacetime. The Proof-of-Spacetime requires a storage node to sequentially create Proof-of-Replications, which leads to a sequential chain of proofs. The proof chain can be verified later to compensate the storage node accordingly. The proof algorithm needs a very tight design to be secure against common attacks like Sybil, Outsourcing, or Generation attacks. In addition to incentivizing the persistent storage of data, Filecoin also charges clients for retrieving files and compensates the storage nodes for delivering them. For this purpose, Filecoin implements a Retrieval Market where clients can place an order for a file. The client and the identified retrieval node settle on a price upon finding a node that can deliver the file. The client issues payments via an off-chain payment channel to the retrieval node during the retrieval. [43, 113] The retrieval node must not directly be a storage node. It could also be a regular node that has the requested file cached or a node that forwards the request to some storage node.

B.2 Incentivization in Swarm

Swarm is another next-generation peer-to-peer data network that the Ethereum Foundation originally developed. The incentive mechanism of Swarm follows an entirely different approach. Instead of incentivizing nodes to store data for longer periods, Swarm's primary incentive compensates nodes for serving data. As pointed out above, such a mechanism works well for popular files. In that case, it pays off for a node to host a popular file because they get paid for serving it to clients. Swarm also offers storage incentives for compensating nodes to store unpopular files for longer periods. The whole incentive scheme of Swarm is built upon three mechanisms:

1. **SWAP (Swarm Accounting Protocol)**
   The Swarm Accounting Protocol is solely responsible for handling the bandwidth incentives. When thinking of a peer A that wants to retrieve some file from peer B, peer A first initiates a connection by executing the protocol handshake. During the handshake phase, the prices for retrieving a chunk of a file are negotiated between the peers. After the handshake phase, it keeps track of any chunk of data exchanged between those two peers in the Swarm network. The idea behind monitoring every exchange is that Swarm's primary goal is to allow compensation in a service-for-service and, alternatively, service-for-money form. For this reason, SWAP manages a balance between any two peers, with an initial balance of zero. As soon as peer A consumes some service of another peer B, the debt on the balance

increases towards the consuming peer A. Swap also defines a *Payment threshold* and a *Disconnect threshold*. Upon reaching the former threshold, the consuming peer A should pay for the consumed service. If peer A reaches the *Disconnect threshold* by continuing to consume the service without paying, peer B will disconnect and block any service to peer A until it settles its debt. The SWAP enforces this logic via a so-called chequebook smart contract that the consuming peer holds. Since managing the whole logic on-chain in the smart contract rate limits the chunk exchanges, the logic is executed off-chain by sending cheques for each chunk. The consuming peer A sends the cheques to the serving peer B. Due to the cumulative design of these cheques, peer B does not need to cash each received cheque. It suffices to wait for a few cheques and only submit the last one to the chequebook contract. Consequently, the chequebook contract validates the cheque and sends the respective amount of money to peer B's account. When both peers trust each other, peer B could save some transaction costs by not submitting every cheque but batching them and waiting for a certain amount of time. The SWAP will even allow a peer to wait for the debt to be settled over time. Automatic settling of debts should allow free usage for any node that only expects to upload and download small amounts of content. Paying nodes will still get a fast service without waiting for debts to amortize over time. Figure B.1 visualizes the essential steps of SWAP. The red arrow pointing on the scale shows the current state of the balance. As soon as it reaches one of the defined payment thresholds, the corresponding peer has to take action. Otherwise, the opposite peer will disconnect when reaching the disconnect threshold.

When requesting a chunk, every node will forward the chunk request to some next node until a node is found that owns the requested chunk. As every in-between node has to pay for forwarding the chunk request, nodes are incentivized to cache the chunk upon receiving it. Once the chunk is requested again soon, the in-between node no longer needs to request it again and therefore gains more profit. Therefore, this implicit auto-scaling incentive improves the redundancy of chunks and implicitly reduces the latency for retrieving the chunk, as fewer hops will be needed to find the chunk.

2. **SWEAR (SWarm Enforcement And Registration)**
SWEAR is a smart contract that handles the incentives for storing data for longer periods. When a client wants to safely store chunks on a node that guarantees storage with pre-defined conditions, it can purchase these storage promises from this node. Each node that provides storage has to register first via the SWEAR contract. During the registration, the storage node has to deposit a certain amount of collateral. This collateral gets seized when the node violates any of the storage promises. This mechanism incentivizes nodes to keep their promises and behave honestly. Losing reputation is not an option, as nodes could easily create new identities on the fly. In case of a lost chunk, litigation is needed, which is handled by the SWINDLE smart contract.

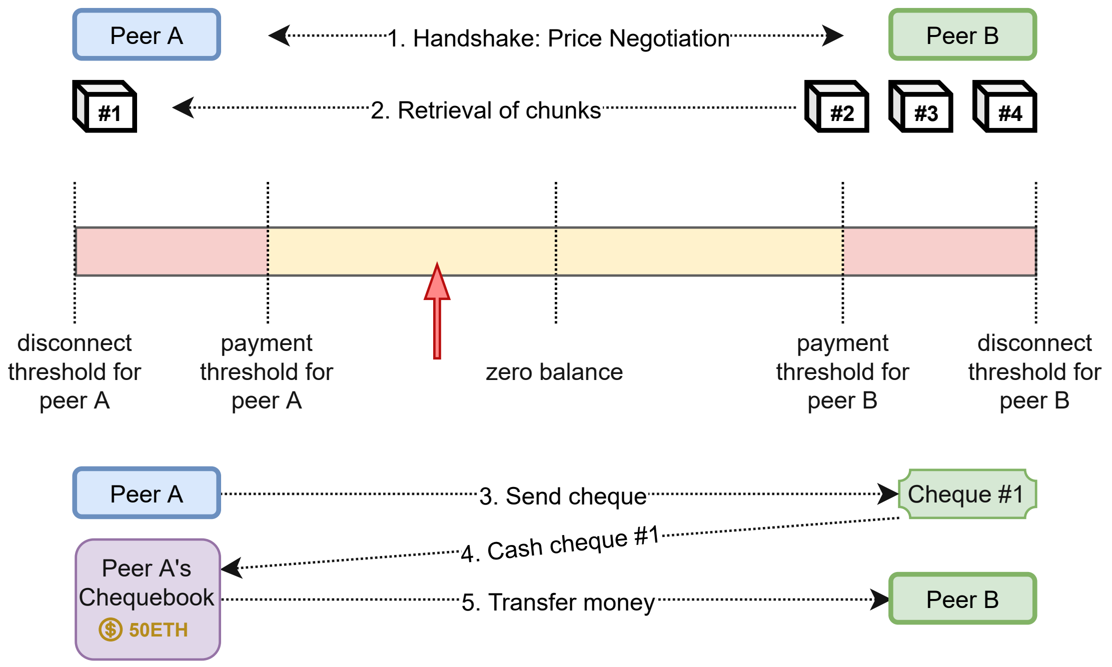

Peer A
1. Handshake: Price Negotiation
Peer B
#1
2. Retrieval of chunks
#2
#3
#4
disconnect threshold for peer A
payment threshold for peer A
zero balance
payment threshold for peer B
disconnect threshold for peer B
Peer A
3. Send cheque
Cheque #1
4. Cash cheque #1
Peer A's Chequebook
50ETH
5. Transfer money
Peer B

Figure B.1: Visualization of the Swarm Accounting Protocol (SWAP) inspired by Trón et al. [114]

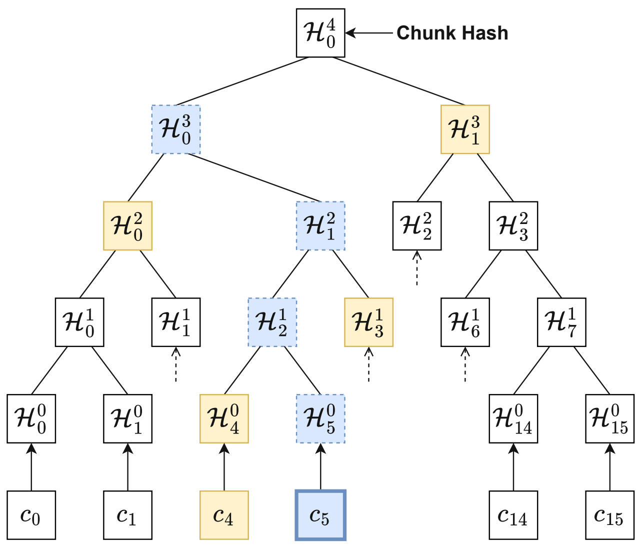
3. **SWINDLE (Secured With INsurance Deposit Litigation and Escrow)** Clients receive receipts for stored chunks, which can be used to enforce litigation when a storage node loses the chunk. Any node with a valid receipt for a lost chunk can initiate the litigation process by posting a challenge on the SWINDLE contract. To compensate an honest node for taking on the challenge and uploading the chunk, the challenger must deposit adequate funds. This mechanism should also hinder any foolish attacks against the blockchain, as the whole litigation process gets executed on-chain. If a storage node still stores the challenged chunk, it can refute the challenge by uploading the requested chunk to the chain. The SWINDLE contract then verifies the uploaded chunk by checking its hash against the challenge receipt. If the refutation of the challenge is valid, the storage node gets compensated for its efforts. The remaining balance of the deposit gets refunded to the challenger. If a refutation of the challenge is not possible or the deadline passes, the registered storage node in the SWEAR contract will lose its deposit.

   An alternative refutation to a challenge is presenting a Proof-of-Storage in the form of Swarm's Proof-of-Custody. A Proof-of-Custody is based on the Proof-of-Inclusion on Merkle trees, which can quickly prove and verify that some segment is included in the tree. Merkle trees were introduced in Section 2.2. Figure B.2a describes how the basic Proof-of-Inclusion is constructed from Merkle trees. The challenged chunk is split into equal-sized segments $c = c_0||c_1|| \dots ||c_{n-1}$, and the challenge requests a random segment to be checked in the proof. In more detail, the challenger needs to provide a random seed from which the segment's index derives. In the example from Figure B.2a the segment c_5 was selected. The hash function $\mathcal{H}_i^\lambda$ is defined as follows:

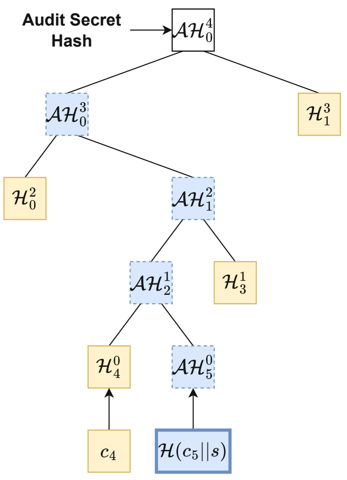
$$\mathcal{M}_{2,\mathcal{H}}(c,\lambda,i) \overset{\text{def}}{=} \begin{cases} \mathcal{H}(c_i), & \text{if } \lambda = 0 \\ \mathcal{H}(\mathcal{M}_{2,\mathcal{H}}(c,\lambda-1,2*i)||\mathcal{M}_{2,\mathcal{H}}(c,\lambda-1,2*i+1)), & \text{otherwise} \end{cases}$$

$$\mathcal{H}_i^\lambda \overset{\text{def}}{=} \mathcal{M}_{2,\mathcal{H}}(c,\lambda,i)$$

The storage node then needs to provide the audit secret hash to prove the custody of the chunk. Figure B.2b shows how to calculate the secret. The requested segment is hashed together with the random challenge appended. The audit secret hash is calculated by following the path from the modified segment to the root and calculating the Merkle tree root. To refute the challenge, the storage node must provide the original Proof-of-Inclusion together with the audit secret hash. The challenger can check the Proof-of-Inclusion since the root of the Merkle tree is equal to the hash of the chunk, which is a publicly known value. Additionally, the challenger can check the audit secret hash by modifying the requested segment with the provided seed and rebuilding the Audit Secret Merkle tree. [115]



(a) Proof-of-Inclusion for chunk c_5 in a Merkle tree inspired by Trón et al. [115].



(b) Proof-of-Custody: Calculation of the Audit Secret Hash inspired by Trón et al. [115]

Figure B.2: Proof-of-Custody in Swarm

Overall, Swarm's incentivization concept takes on a completely different approach by primarily incentivizing bandwidth instead of storage as Filecoin is doing it, but the mechanisms behind follow similar technical approaches [46, 43, 116].

B.3 Incentivization in Storj

Besides Filecoin and Swarm, Storj is another popular data network that generally puts more trust into its nodes. There are no complex checks for every stored byte, and no deposit mechanism exists for punishing dishonest nodes. Instead, a reputation system punishes nodes for dishonest behavior, which should generally be a rare occasion. Storj splits its network into parts which are managed by so-called satellite nodes. These satellite nodes conduct periodic audit checks on each storage node. The audit checks consist of retrieving chunks and subsequently checking their correctness. Correctness checks use erasure codes, a widespread technique to virtually create redundancy in data. Only a certain amount of chunks need to be downloaded to retrieve a file. The remaining chunks can be recomputed with the erasure code algorithm. [43, 117]

List of Figures

List of Tables

List of Listings

Acronyms

ABE Attribute-Based Encryption. 30

BCT Blockchain Technology. 1–3, 5–10, 23, 39–41, 50, 56, 59, 64, 73, 74, 88, 89, 95, 97–99, 111

BPM Business Process Management. 1–5, 8, 23, 39, 40, 42, 46, 49, 55, 56, 76, 88–90, 97–99, 111

BPMN Business Process Model and Notation. 5–8, 43, 45, 61, 86–88, 91, 92, 94, 95, 98, 107, 108

BPMS Business Process Management System. 7, 8

CID Content Identifier. 30–32, 92

CRDT Conflict-Free Replicated Data Type. 34, 71–74, 88–90, 95, 98, 107

CTA Case Transfer Architecture. 39, 40, 42

DAG Directed Acyclic Graph. 10–12, 29–32, 72, 73, 80, 107

DHT Distributed Hash Table. 12, 13, 18, 19, 28, 29, 31, 33, 36–38

DNS Domain Name System. 31

ENS Ethereum Name Service. 31

ICN Information-Centric Networking. 26, 28, 29, 31

IPFS InterPlanetary File System. 10, 12, 14, 26, 29–34, 36–38, 54, 75, 80, 83, 90, 92, 96, 101, 107

IPLD InterPlanetary Linked Data. 30–32

IPNS InterPlanetary Name System. 31–33

JSON JavaScript Object Notation. 20–22, 61, 64, 73, 74, 78, 79, 96, 111

LCA	Loosely Coupled Architecture. 39, 40, 59, 88–90
PIP	Partner Interface Process. 88, 89
PubSub	Publish/Subscribe. 80
RPC	Remote Procedure Call. 16, 17
RTT	Round-Trip Time. 19, 20
SCM	Supply Chain Management. 3, 5–7, 42, 43, 45, 48, 55, 75, 76, 95, 98, 107, 108
TTSM	Time-Traveling State Machine. 76–78, 80, 83
YAML	Yet Another Markup Language. 61

Bibliography

[1] B. Hitpass. *Business Process Management (BPM): Concepts, and How to Apply and Integrate it with IT*. Bernhard Hitpass, 2014. https://books.google.at/books?id=5CEGBAAAQBAJ.

[2] M. Müller, N. Ostern, and M. Rosemann. Silver bullet for all trust issues? blockchain-based trust patterns for collaborative business processes. In *Lecture Notes in Business Information Processing*, pages 3–18. Springer International Publishing, 2020. doi:10.1007/978-3-030-58779-6_1.

[3] R. B. Handfield and C. Bechtel. The role of trust and relationship structure in improving supply chain responsiveness. *Industrial Marketing Management*, 31(4):367–382, July 2002. doi:10.1016/s0019-8501(01)00169-9.

[4] O. L. Pintado. Challenges of blockchain-based collaborative business processes: An overview of the caterpillar system. In *Blockchain and Robotic Process Automation*, pages 31–42. Springer International Publishing, 2021. doi:10.1007/978-3-030-81409-0_3.

[5] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling. Untrusted business process monitoring and execution using blockchain. In *Lecture Notes in Computer Science*, pages 329–347. Springer International Publishing, 2016. doi:10.1007/978-3-319-45348-4_19.

[6] B. Carminati, E. Ferrari, and C. Rondanini. Blockchain as a platform for secure inter-organizational business processes. In *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*. IEEE, Oct. 2018. doi:10.1109/cic.2018.00027.

[7] C. Prybila, S. Schulte, C. Hochreiner, and I. Weber. Runtime verification for business processes utilizing the bitcoin blockchain. *Future Generation Computer Systems*, 107:816–831, June 2020. doi:10.1016/j.future.2017.08.024.

[8] O. López-Pintado, L. García-Bañuelos, M. Dumas, and I. Weber. Caterpillar: A blockchain-based business process management system. *BPM (Demos)*, 172, 2017.

[9] C. Sturm, J. Szalanczi, S. Jablonski, and S. Schönig. Decentralized control: A novel form of interorganizational workflow interoperability. In *Lecture Notes in Business Information Processing*, pages 261–276. Springer International Publishing, 2020. doi:10.1007/978-3-030-63479-7_18.

[10] C. Sturm and S. Jablonski. Interorganizational process execution beyond ethereum: Road to a special purpose ecosystem. In *Proceedings of the workshops co-organized with the 13th IFIP WG 8.1 working conference on the Practice of Enterprise Modelling (PoEM 2020)*. CEUR-WS, 2020. https://ceur-ws.org/Vol-2749/paper6.pdf.

[11] Hevner, March, Park, and Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75, 2004. doi:10.2307/25148625.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. https://books.google.at/books?id=6oHuKQe3TjQC.

[13] B. N. Green, C. D. Johnson, and A. Adams. Writing narrative literature reviews for peer-reviewed journals: secrets of the trade. *Journal of Chiropractic Medicine*, 5(3):101–117, Sept. 2006. doi:10.1016/s0899-3467(07)60142-6.

[14] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers. *Fundamentals of Business Process Management*. Springer Berlin Heidelberg, 2018. doi:10.1007/978-3-662-56509-4.

[15] M. Weske. Business process management architectures. In *Business Process Management*, pages 333–371. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-28616-2_7.

[16] D. Kleebinder. Time-travelling state machines for verifiable bpm. *TU Wien*, 2023. doi:10.34726/HSS.2023.99122.

[17] R. Breu, S. Dustdar, J. Eder, C. Huemer, G. Kappel, J. Kopke, P. Langer, J. Mangler, J. Mendling, G. Neumann, S. Rinderle-Ma, S. Schulte, S. Sobernig, and B. Weber. Towards living inter-organizational processes. In *2013 IEEE 15th Conference on Business Informatics*. IEEE, July 2013. doi:10.1109/cbi.2013.59.

[18] J. Mendling. Towards blockchain support for business processes. In *Lecture Notes in Business Information Processing*, pages 243–248. Springer International Publishing, 2018. doi:10.1007/978-3-319-94214-8_15.

[19] J. Mendling, I. Weber, W. V. D. Aalst, J. V. Brocke, C. Cabanillas, F. Daniel, S. Debois, C. D. Ciccio, M. Dumas, S. Dustdar, A. Gal, L. García-Bañuelos, G. Governatori, R. Hull, M. L. Rosa, H. Leopold, F. Leymann, J. Recker, M. Reichert, H. A. Reijers, S. Rinderle-Ma, A. Solti, M. Rosemann, S. Schulte, M. P. Singh, T. Slaats, M. Staples, B. Weber, M. Weidlich, M. Weske, X. Xu, and

L. Zhu. Blockchains for business process management - challenges and opportunities. *ACM Transactions on Management Information Systems*, 9(1):1–16, Feb. 2018. doi:10.1145/3183367.

[20] J. Ladleif, M. Weske, and I. Weber. Modeling and enforcing blockchain-based choreographies. In *Lecture Notes in Computer Science*, pages 69–85. Springer International Publishing, 2019. doi:10.1007/978-3-030-26619-6_7.

[21] O. Lopez-Pintado, M. Dumas, L. Garcia-Banuelos, and I. Weber. Interpreted execution of business process models on blockchain. In *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, Oct. 2019. doi:10.1109/edoc.2019.00033.

[22] C. D. Ciccio, A. Cecconi, J. Mendling, D. Felix, D. Haas, D. Lilek, F. Riel, A. Rumpl, and P. Uhlig. Blockchain-based traceability of inter-organisational business processes. In *Lecture Notes in Business Information Processing*, pages 56–68. Springer International Publishing, 2018. doi:10.1007/978-3-319-94214-8_4.

[23] J. Köpke, M. Franceschetti, and J. Eder. Balancing privity and enforceability of BPM-based smart contracts on blockchains. In *Business Process Management: Blockchain and Central and Eastern Europe Forum*, pages 87–102. Springer International Publishing, 2019. doi:10.1007/978-3-030-30429-4_7.

[24] C. Rondanini, B. Carminati, F. Daidone, and E. Ferrari. Blockchain-based controlled information sharing in inter-organizational workflows. In *2020 IEEE International Conference on Services Computing (SCC)*. IEEE, Nov. 2020. doi:10.1109/scc49832.2020.00056.

[25] A. Tomescu. What is a merkle tree? https://decentralizedthoughts.github.io/2020-12-22-what-is-a-merkle-tree/, Dec. 2020. accessed Dec. 2022.

[26] R. Vagg. Content-addressed distributed data structures [speakeasyjs]. https://www.youtube.com/watch?v=VtzpJU4Cns8, Sept. 2020. accessed Jun. 2022.

[27] Wikipedia contributors. Radix tree. https://en.wikipedia.org/w/index.php?title=Radix_tree&oldid=1105756012. accessed Nov. 2022.

[28] J. Cook and P. Wackerow. Patricia merkle trees. https://ethereum.org/en/developers/docs/data-structures-and-encoding/patricia-merkle-trie/. accessed Sep. 2022.

[29] D. Stearns. The trie data structure. https://drstearns.github.io/tutorials/trie/, Oct. 2021. accessed Dec. 2022.

[30] B. Heep. R/kademlia: Recursive and topology-aware overlay routing. In *2010 Australasian Telecommunication Networks and Applications Conference*. IEEE, Oct. 2010. doi:10.1109/atnac.2010.5680244.

[31] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Peer-to-Peer Systems*, pages 53–65. Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-45748-8_5.

[32] Protocol Labs. The distributed hash table. https://curriculum.pl-launchpad.io/curriculum/libp2p/dht/. accessed Dec. 2022.

[33] I. Baumgart and S. Mies. S/kademlia: A practicable approach towards secure key-based routing. In *2007 International Conference on Parallel and Distributed Systems*. IEEE, 2007. doi:10.1109/icpads.2007.4447808.

[34] M. J. Freedman and D. Maziéres. Sloppy hashing and self-organizing clusters. In *Peer-to-Peer Systems II*, pages 45–55. Springer Berlin Heidelberg, 2003. doi:10.1007/978-3-540-45172-3_4.

[35] H. Kanemitsu and H. Nakazato. KadRTT: Routing with network proximity and uniform ID arrangement in kademlia. In *2021 IFIP Networking Conference (IFIP Networking)*. IEEE, June 2021. doi:10.23919/ifipnetworking52078.2021.9472816.

[36] P. J. Taylor, T. Dargahi, A. Dehghantanha, R. M. Parizi, and K.-K. R. Choo. A systematic literature review of blockchain cyber security. *Digital Communications and Networks*, 6(2):147–156, May 2020. doi:10.1016/j.dcan.2019.01.005.

[37] S. Ali, G. Wang, B. White, and R. L. Cottrell. A blockchain-based decentralized data storage and access framework for PingER. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, Aug. 2018. doi:10.1109/trustcom/bigdatase.2018.00179.

[38] R. Kumar, N. Marchang, and R. Tripathi. Distributed off-chain storage of patient diagnostic reports in healthcare system using IPFS and blockchain. In *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, Jan. 2020. doi:10.1109/comsnets48256.2020.9027313.

[39] A. M. Athreya, A. A. Kumar, S. M. Nagarajath, H. L. Gururaj, V. R. Kumar, D. N. Sachin, and K. R. Rakesh. Peer-to-peer distributed storage using interplanetary file system. In *Advances in Intelligent Systems and Computing*, pages 711–721. Springer Singapore, Aug. 2020. doi:10.1007/978-981-15-3514-7_54.

[40] M. Shah, M. Shaikh, V. Mishra, and G. Tuscano. Decentralized cloud storage using blockchain. In *2020 4th International Conference on Trends in Electronics and*

Informatics (ICOEI)(48184). IEEE, June 2020. doi:10.1109/icoei48184.2020.9143004.

[41] L. Sari and M. Sipos. Filetribe: Blockchain-based secure file sharing on ipfs. In *European Wireless 2019; 25th European Wireless Conference*, pages 1–6, 2019.

[42] J. Zhang, S. Zhong, J. Wang, L. Wang, Y. Yang, B. Wei, and G. Zhou. A review on blockchain-based systems and applications. In *Internet of Vehicles. Technologies and Services Toward Smart Cities*, pages 237–249. Springer International Publishing, 2020. doi:10.1007/978-3-030-38651-1_20.

[43] E. Daniel and F. Tschorsch. Ipfs and friends: A qualitative comparison of next generation peer-to-peer data networks. *IEEE Communications Surveys & Tutorials*, 24(1):31–52, 2022. doi:10.1109/comst.2022.3143147.

[44] R. Kumar and R. Tripathi. A secure and distributed framework for sharing COVID-19 patient reports using consortium blockchain and IPFS. In *2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC)*. IEEE, Nov. 2020. doi:10.1109/pdgc50313.2020.9315755.

[45] T. Xiao and Y. Huang. Design and implementation of web system based on blockchain. In *Cloud Computing and Security*, pages 706–717. Springer International Publishing, 2018. doi:10.1007/978-3-030-00009-7_63.

[46] H. Huang, J. Lin, B. Zheng, Z. Zheng, and J. Bian. When blockchain meets distributed file systems: An overview, challenges, and open issues. *IEEE Access*, 8:50574–50586, 2020. doi:10.1109/access.2020.2979881.

[47] K. Poudel, A. B. Aryal, A. Pokhrel, and P. Upadhyaya. Photograph ownership and authorization using blockchain. In *2019 Artificial Intelligence for Transforming Business and Society (AITB)*. IEEE, Nov. 2019. doi:10.1109/aitb48515.2019.8947438.

[48] O. Dzobo, B. Malila, and L. Sithole. Proposed framework for blockchain technology in a decentralised energy network. *Protection and Control of Modern Power Systems*, 6(1), Sept. 2021. doi:10.1186/s41601-021-00209-8.

[49] J. Kan and K. S. Kim. MTFS: Merkle-tree-based file system. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, May 2019. doi:10.1109/bloc.2019.8751389.

[50] M. Verdonck and G. Poels. Decentralized data access with IPFS and smart contract permission management for electronic health records. In *Business Process Management Workshops*, pages 5–16. Springer International Publishing, 2020. doi:10.1007/978-3-030-66498-5_1.

[51] P. P. Devi, S. A. Bragadeesh, and A. Umamakeswari. Secure data management using IPFS and ethereum. In *Lecture Notes on Data Engineering and Communications Technologies*, pages 565–578. Springer Singapore, 2021. doi:10.1007/978-981-33-4968-1_44.

[52] C. Rahalkar and D. Gujar. Content addressed p2p file system for the web with blockchain-based meta-data integrity. In *2019 International Conference on Advances in Computing, Communication and Control (ICAC3)*. IEEE, Dec. 2019. doi:10.1109/icac347590.2019.9036792.

[53] L. Herencic, P. Ilak, I. Rajsl, Z. Zmijarevic, M. Cvitanovic, M. Delimar, and B. Pecanac. Overview of the main challenges and threats for implementation of the advanced concept for decentralized trading in microgrids. In *IEEE EUROCON 2019 -18th International Conference on Smart Technologies*. IEEE, July 2019. doi:10.1109/eurocon.2019.8861906.

[54] E. J. Lopes, S. Kataria, S. Keshav, S. T. Ikram, M. R. Ghalib, A. Shankar, and M. Krichen. Live video streaming service with pay-as-you-use model on ethereum blockchain and InterPlanetary file system. *Wireless Networks*, 28(7):3111–3125, June 2022. doi:10.1007/s11276-022-03009-6.

[55] N. Z. Benisi, M. Aminian, and B. Javadi. Blockchain-based decentralized storage networks: A survey. *Journal of Network and Computer Applications*, 162:102656, July 2020. doi:10.1016/j.jnca.2020.102656.

[56] M. Nouman, K. Ullah, and M. Azam. Secure digital transactions in the education sector using blockchain. *ICST Transactions on Scalable Information Systems*, page 171758, July 2018. doi:10.4108/eai.3-11-2021.171758.

[57] S. Rahmadika and K.-H. Rhee. Toward privacy-preserving shared storage in untrusted blockchain p2p networks. *Wireless Communications and Mobile Computing*, 2019:1–13, May 2019. doi:10.1155/2019/6219868.

[58] M. E. Tschuchnig, D. Radovanovic, E. Hirsch, A.-M. Oberluggauer, and G. Schafer. Immutable and democratic data in permissionless peer-to-peer systems. In *2019 Sixth International Conference on Software Defined Systems (SDS)*. IEEE, June 2019. doi:10.1109/sds.2019.8768645.

[59] D. Trautwein, M. Schubotz, and B. Gipp. Leveraging node heterogeneity to improve content discovery and content retrieval in peer-to-peer networks. In *Proceedings of the CoNEXT Student Workshop*. ACM, Dec. 2021. doi:10.1145/3488658.3493781.

[60] S. Vimal and S. K. Srivatsa. A new cluster p2p file sharing system based on IPFS and blockchain technology. *Journal of Ambient Intelligence and Humanized Computing*, Sept. 2019. doi:10.1007/s12652-019-01453-5.

[61] Z. Ning, L. Xiao, W. Liang, W. Shi, and K.-C. Li. On the exploitation of blockchain for distributed file storage. *Journal of Sensors*, 2020:1–11, Dec. 2020. doi:10.1155/2020/8861688.

[62] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371, Dec. 2004. doi:10.1145/1041680.1041681.

[63] M. Zumwalt. The power of content-addressing. https://flyingzumwalt.gitbooks.io/decentralized-web-primer/content/avenues-for-access/lessons/power-of-content-addressing.html. accessed Jun. 2022.

[64] R. Norvill, B. B. F. Pontiveros, R. State, and A. Cullen. IPFS for reduction of chain size in ethereum. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, July 2018. doi:10.1109/cybermatics_2018.2018.00204.

[65] V. Trón. The book of swarm (v1.0 pre-release 7). https://www.ethswarm.org/The-Book-of-Swarm.pdf, accessed Apr. 2022.

[66] M. Steichen, B. Fiz, R. Norvill, W. Shbair, and R. State. Blockchain-based, decentralized access control for IPFS. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, July 2018. doi:10.1109/cybermatics_2018.2018.00253.

[67] J. Benet. Ipfs - content addressed, versioned, p2p file system, 2014. arXiv:1407.3561. doi:10.48550/ARXIV.1407.3561.

[68] Protocol Labs. Interplanetary name system (ipns). https://docs.ipfs.tech/concepts/ipns/#mutability-in-ipfs. accessed Dec. 2022.

[69] ENS Labs Ltd. ENS architecture. https://docs.ens.domains/. accessed Dec. 2022.

[70] B. Putz, M. Dietz, P. Empl, and G. Pernul. EtherTwin: Blockchain-based secure digital twin information management. *Information Processing & Management*, 58(1):102425, Jan. 2021. doi:10.1016/j.ipm.2020.102425.

[71] A. Allouch, O. Cheikhrouhou, A. Koubâa, K. Toumi, M. Khalgui, and T. N. Gia. UTM-chain: Blockchain-based secure unmanned traffic management for internet of drones. *MDPI AG Sensors*, 21(9):3049, Apr. 2021. doi:10.3390/s21093049.

[72] O. Yoachimik. Cloudflare ddos threat report 2022 q3. https://blog.cloudflare.com/cloudflare-ddos-threat-report-2022-q3/, Oct. 2022. accessed Oct. 2022.

[73] E. Brumaghin. Threat spotlight: Cyber criminal adoption of ipfs for phishing, malware campaigns. https://blog.talosintelligence.com/ipfs-abuse/, Nov. 2022.

[74] Anomali Threat Research. The interplanetary storm: New malware in wild using interplanetary file system's (ipfs) p2p network. https://www.anomali.com/blog/the-interplanetary-storm-new-malware-in-wild-using-interplanetary-file-systems-ipfs-p2p-network, June 2019. accessed Nov. 2022.

[75] C. Karapapas, I. Pittaras, N. Fotiou, and G. C. Polyzos. Ransomware as a service using smart contracts and IPFS. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, May 2020. doi:10.1109/icbc48266.2020.9169451.

[76] E. Politou, E. Alepis, C. Patsakis, F. Casino, and M. Alazab. Delegated content erasure in IPFS. *Future Generation Computer Systems*, 112:956–964, Nov. 2020. doi:10.1016/j.future.2020.06.037.

[77] L. Balduf, S. Henningsen, M. Florian, S. Rust, and B. Scheuermann. Monitoring data requests in decentralized data storage systems: A case study of IPFS. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, July 2022. doi:10.1109/icdcs54860.2022.00069.

[78] B. Prünster, A. Marsalek, and T. Zefferer. Total eclipse of the heart – disrupting the interplanetary file system. In *31st USENIX Security Symposium*, pages 3735–3752, United States, 2022. USENIX Association.

[79] Statista. Topic: Mobile internet usage worldwide. https://www.statista.com/topics/779/mobile-internet/, Oct. 2022. accessed Dec. 2022.

[80] W. M. van der Aalst. Process-oriented architectures for electronic commerce and interorganizational workflow. *Information Systems*, 24(8):639–671, Dec. 1999. doi:10.1016/s0306-4379(00)00003-x.

[81] A. Norta. Designing a smart-contract application layer for transacting decentralized autonomous organizations. In *Communications in Computer and Information Science*, pages 595–604. Springer Singapore, 2017. doi:10.1007/978-981-10-5427-3_61.

[82] X. Xu, C. Pautasso, L. Zhu, Q. Lu, and I. Weber. A pattern collection for blockchain-based applications. In *Proceedings of the 23rd European Conference on Pattern Languages of Programs*. ACM, July 2018. doi:10.1145/3282308.3282312.

[83] A. B. Sideridis and C. Z. Patrikakis, editors. *Next Generation Society. Technological and Legal Issues*. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-11631-5.

[84] A. Stasis and L. Demiri. Secure document exchange in the greek public sector via eDelivery. In *Communications in Computer and Information Science*, pages 213–227. Springer International Publishing, 2017. doi:10.1007/978-3-319-71117-1_15.

[85] M. Rosenfeld. Analysis of hashrate-based double spending. arXiv:1402.2009, Feb. 2014. doi:10.48550/ARXIV.1402.2009.

[86] K. Kingsbury. Serializability. https://jepsen.io/consistency/models/serializable. accessed Feb. 2023.

[87] W. Fdhila, D. Knuplesch, S. Rinderle-Ma, and M. Reichert. Verifying compliance in process choreographies: Foundations, algorithms, and implementation. *Information Systems*, 108:101983, Sept. 2022. doi:10.1016/j.is.2022.101983.

[88] The Apache Software Foundation. Java expression language (jexl). https://commons.apache.org/proper/commons-jexl/. accessed Feb. 2023.

[89] H. Sanjuan, S. Poyhtari, P. Teixeira, and I. Psaras. Merkle-CRDTs: Merkle-dags meet crdts. arXiv:2004.00107, Apr. 2020. doi:10.48550/ARXIV.2004.00107.

[90] Object Management Group®(OMG®). Business process model and notation. https://www.omg.org/spec/BPMN/2.0.2/About-BPMN. accessed Mar. 2023.

[91] F. Corradini, C. Muzi, B. Re, L. Rossi, and F. Tiezzi. Animating multiple instances in BPMN collaborations: From formal semantics to tool support. In *Lecture Notes in Computer Science*, pages 83–101. Springer International Publishing, 2018. doi:10.1007/978-3-319-98648-7_6.

[92] J. Kirchhoff, S. Gottschalk, and G. Engels. Detecting data incompatibilities in process-driven decision support systems. In *Lecture Notes in Business Information Processing*, pages 89–103. Springer International Publishing, 2022. doi:10.1007/978-3-031-11510-3_6.

[93] K. Soleymanzadeh, Y. Bul, S. Bagci, and G. Kardas. A tool for modeling JsonLogic based business process rules. In *2019 1st International Informatics and Software Engineering Conference (UBMYK)*. IEEE, Nov. 2019. doi:10.1109/ubmyk48245.2019.8965462.

[94] J. Pasley. How BPEL and SOA are changing web services development. *IEEE Internet Computing*, 9(3):60–67, May 2005. doi:10.1109/mic.2005.56.

[95] C. Molina-Jimenez and S. Shrivastava. Maintaining consistency between loosely coupled services in the presence of timing constraints and validation errors. In *2006 European Conference on Web Services (ECOWS'06)*. IEEE, 2006. doi:10.1109/ecows.2006.25.

[96] M. Van Wingerde. Towards inter-organizational business process governance through blockchain. In *Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at BPM 2019 co-located with 17th International Conference on Business Process Management (BPM 2019)*. CEUR-WS, 2019. https://ceur-ws.org/Vol-2420/paperDC6.pdf.

[97] F. Milani, L. Garcia-Banuelos, H. A. Reijers, and L. Stepanyan. Business process redesign heuristics for blockchain solutions. In *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, Oct. 2020. doi:10.1109/edoc49727.2020.00033.

[98] J. Ladleif, C. Friedow, and M. Weske. An architecture for multi-chain business process choreographies. In *Business Information Systems*, pages 184–196. Springer International Publishing, 2020. doi:10.1007/978-3-030-53337-3_14.

[99] A. Halabi-Echeverry, H. Nino-Vergara, N. Obregon-Neira, and S. M. N. Islam. Blockchain for data sharing in the rational use of coastlines and seaport demands in inter-organizational networks: Development of a new intelligent decision support system. In *2020 5th International Conference on Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA)*. IEEE, Nov. 2020. doi:10.1109/citisia50690.2020.9371837.

[100] M. van Wingerde and H. Weigand. An ontological analysis of artifact-centric business processes managed by smart contracts. In *2020 IEEE 22nd Conference on Business Informatics (CBI)*. IEEE, June 2020. doi:10.1109/cbi49978.2020.00032.

[101] P. Klinger, L. Nguyen, and F. Bodendorf. Upgradeability concept for collaborative blockchain-based business process execution framework. In *Blockchain – ICBC 2020*, pages 127–141. Springer International Publishing, 2020. doi:10.1007/978-3-030-59638-5_9.

[102] L. Argento, S. Graziano, A. Garro, A. Guzzo, F. Pasqua, and D. Saccà. A simulation-based and data-driven framework for enabling the analysis and design of business processes based on blockchain and smart contracts solutions. In *DLT 2020: Distributed Ledger Technology, Distributed Ledger Technology Workshop*, Aachen, Germany, 2020. RWTH Aachen. https://www.tib.eu/de/suchen/id/TIBKAT%3A179652803X.

[103] I. E. Kassmi and Z. Jarir. Blockchain-oriented inter-organizational collaboration between healthcare providers to handle the COVID-19 process. *International*

Journal of Advanced Computer Science and Applications, 12(12), 2021. `doi:10.14569/ijacsa.2021.0121294`.

[104] F. Härer. Decentralized business process modeling and instance tracking secured by a blockchain. In *Proceedings of the 26th European Conference on Information Systems (ECIS 2018)*, 2018. `doi:10.5281/ZENODO.2585717`.

[105] J. Köpke. Towards modeling privity and enforceability requirements for bpm based smart contracts. In *Companion Proceedings of Modellierung 2020 Short, Workshop and Tools & Demo Papers co-located with Modellierung 2020*, 2020. `https://ceur-ws.org/Vol-2542/MOD-DLT5.pdf`.

[106] O. Lopez-Pintado, M. Dumas, L. Garcia-Banuelos, and I. Weber. Interpreted execution of business process models on blockchain. In *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, Oct. 2019. `doi:10.1109/edoc.2019.00033`.

[107] M. Morales-Sandoval, J. A. Molina, H. M. Marin-Castro, and J. L. Gonzalez-Compean. Blockchain support for execution, monitoring and discovery of inter-organizational business processes. *PeerJ Computer Science*, 7:e731, Sept. 2021. `doi:10.7717/peerj-cs.731`.

[108] T. Kampik, A. Najjar, and D. Calvaresi. MAS-aided approval for bypassing decentralized processes: an architecture. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. IEEE, Dec. 2018. `doi:10.1109/wi.2018.000-6`.

[109] P. Sandner, A. Lange, and P. Schulden. The role of the CFO of an industrial company: An analysis of the impact of blockchain technology. *Future Internet*, 12(8):128, July 2020. `doi:10.3390/fi12080128`.

[110] C. Udokwu, A. Norta, and C. Wenna. Designing a collaborative construction-project platform on blockchain technology for transparency, traceability, and information symmetry. In *2021 2nd Asia Service Sciences and Software Engineering Conference*. ACM, Feb. 2021. `doi:10.1145/3456126.3456134`.

[111] G. V. Savin and V. M. Katochkov. Institutional and evolutionary aspects of modern interactions in transport and logistics systems. In *AIP Conference Proceedings*, volume 2389, 2021. `doi:10.1063/5.0063494`.

[112] B. Fisch, J. Bonneau, N. Greco, and J. Benet. Scaling proof-of-replication for filecoin mining. `https://research.protocol.ai/publications/scaling-proof-of-replication-for-filecoin-mining/fisch2018.pdf`, 2018.

[113] J. Benet, D. Dalrymple, and N. Greco. Proof of replication. `https://filecoin.io/proof-of-replication.pdf`, 2017. Protocol Labs. accessed Sep. 2022.

[114] V. Trón and A. Fischer. Swap, swear, and swindle games. https://www.youtube.com/watch?v=9Cgyhsjsfbg. accessed Oct. 2022.

[115] V. Trón, A. Fischer, and N. Johnson. smash-proof: auditable storage for swarm secured by masked audit secret hash. https://ethersphere.github.io/swarm-home/ethersphere/orange-papers/2/smash.pdf, 2016. accessed Oct. 2022.

[116] V. Trón, A. Fischer, D. A. Nagy, Z. Felföldi, and N. Johnson. Swap, swear and swindle: incentive system for swarm. https://ethersphere.github.io/swarm-home/ethersphere/orange-papers/1/sw%5E3.pdf, 2016. accessed Oct. 2022.

[117] Storj Labs. Storj: A decentralized cloud storage network framework. https://www.storj.io/storj.pdf, 2018. accessed Nov. 2022.