

Accountability Management of Human-based Activities in Collaborative Computing

based on a Distributed Ledger Technology

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Alexander Patronas

Matrikelnummer 00425487

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. Schahram Dustdar

Mitwirkung: Dr. Ognjen Šćekić

Wien, 1. September 2020

Alexander Patronas

Schahram Dustdar



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.



Accountability Management of Human-based Activities in Collaborative Computing

based on a Distributed Ledger Technology

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Alexander Patronas

Registration Number 00425487

to the Faculty of Informatics

at the TU Wien

Advisor: Univ. Prof. Dr. Schahram Dustdar

Assistance: Dr. Ognjen Šćekić

Vienna, 1st September, 2020

Alexander Patronas

Schahram Dustdar



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Alexander Patronas

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, 1. September 2020

Alexander Patronas



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I would like to thank my supervisors Univ. Prof. Dr. Scharam Dustdar and Dr. Ognjen Šćekić from the institute of Distributed Systems Group at the Vienna University of Technology. In particular, I would like to thank Dr. Ognjen Šćekić for the repeated brainstorming and feedback sessions throughout the development and writing of the thesis. Thank you for the open door whenever I had questions about the research topic.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Die Zuordnung von Verantwortlichkeiten für spezifische Handlungen oder Aktionen in einem verteilten System ist ein sehr wichtiger Bestandteil für die Vertrauenswürdigkeit und Nachvollziehbarkeit eines solchen Systems. Es erfordert ein klares Regelwerk, welches Fehlverhalten einschränkt und die Richtigkeit von Handlungen gewährleistet. Hierfür ist es notwendig verantwortliche Nutzer zu definieren und zu identifizieren.

Da die individuelle Verantwortlichkeit beim kollaborativem Prozess eine zentrale Rolle spielt, ist ein vertrauenswürdiges System, das Fehlverhalten verhindert, von entscheidender Bedeutung. In verteilten Systemen wird Vertrauen üblicherweise durch die Abhängigkeit von “Trusted Third Parties” (vertrauensvollen Dritten Parteien) aufgebaut, was die Verteilung sensibler Daten erfordert und zum Verlust der Datenhoheit führt. Um Unabhängigkeit von diesen “Trusted Third Parties” zu gewährleisten, sind neue Konzepte erforderlich, die nicht auf zentral verwaltete Zertifizierungsstellen basieren und sicherstellen, dass Fehlverhalten in einer solchen Umgebung erkannt und bis zum verantwortlichen Benutzer transparent überprüft und zurückverfolgt werden kann.

Mit der Einführung der ersten Distributed Ledger Technology (DLT) namens Bitcoin im Jahr 2008 wurde es möglich, Vertrauen zwischen einzelnen Teilnehmern in einem peer-to-peer (P2P) Netzwerk aufzubauen, ohne der Abhängigkeit einer vertrauensvollen dritten Partei beziehungsweise zentralen Zertifizierungsstelle.

Im Rahmen dieser Diplomarbeit wurde ein accountability model implementiert, welches eine unabhängige Nachvollziehbarkeit gewährleistet. Anhand eines dezentralen Systems und aktiver Beteiligung der Teilnehmer wird es ermöglicht, dass die von Menschen ausgeführten Aktivitäten verifizierbar und vertrauenswürdig abgebildet werden. Die unabhängige Generierung zuverlässiger und überprüfbarer Herkunftsinformationen auf der Grundlage des PROV Standards ist ein integraler Bestandteil dieser Umsetzung. Hierfür wird eine kryptographisch abgesicherte Vereinbarung, smart contract, im Ethereum Netzwerk erstellt um die Transaktionen der Teilnehmer und damit ihre Aktionen zu verankern und nachvollziehbar zu machen.

Zu Demonstrations- und Evaluationszwecken wurde zusätzliche ein funktionsfähiger Prototyp implementiert, der das SmartSociety programming framework um ein weiteres Szenario erweitert.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Providing accountability in a collaborative computer system based on human-based activity is a key challenge. The identification and definition of responsible users to ensure proper and correct data transfer demand the design of a system with a set of rules that limit misconduct and verify the correctness of actions. As individual accountability is pivotal in collaborative computing, a trustworthy system that prevents misconduct is crucial. In many distributed systems, trust is established through dependence on third parties, which requires the distribution of sensitive data and results in loss of data sovereignty. New concepts are therefore required that are not relying on trusted third parties, but ensure that misconduct in such an environment is recognized and traced back to the responsible user. With the introduction of the very first Distributed Ledger Technology (DLT) in 2008 named Bitcoin, it became possible for the first time to establish trust between peers in a peer-to-peer (P2P) network without relying on a trusted third party.

Throughout this thesis project, an accountability model was implemented, ensuring that activities carried out by people are verifiable and trustworthy using a decentralized system along with the engagement of a participating collective. The generation of reliable and verifiable provenance information, based on the PROV standard, plays a key role in this implementation. Moreover, a cryptographically enforced agreement, such as a smart contract in Ethereum is utilized to achieve the ability to track the transaction chain of the involved users and thus their actions. It emerged to a reliable candidate solution for detecting misconduct and verification of the correctness of their actions.

For demonstration and evaluation purposes a functional prototype was implemented that extends the SmartSociety programming framework by a working scenario.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung	ix
Abstract	xi
Contents	xiii
1 Introduction	1
1.1 Motivation and Problem Relevance	1
1.2 Expected Results	2
1.3 Methodological Approach	3
1.4 Structure of the Work	3
2 State of the Art	5
3 Background	7
3.1 Collective Adaptive Systems	7
3.2 Accountability	7
3.3 Provenance	9
3.4 The PROV Standard	12
3.5 Distributed Ledger Technology	15
3.6 Ethereum and Smart Contracts	25
3.7 Decentralized Applications	26
4 DLT Review	29
4.1 Bitcoin	29
4.2 Ethereum	29
4.3 IOTA	30
4.4 Nano	31
4.5 Comparison	33
5 Design	35
5.1 SmartSociety Platform	35
5.2 Accountability Model	39
5.3 Scope and Requirements	39
	xiii

5.4	Architecture	49
5.5	PROV-DM Mapping	53
6	Implementation	57
6.1	Technology Stack	57
6.2	Package Structure	59
6.3	Sequence Diagram	60
6.4	SmartAcc - Accountability Component	60
7	Evaluation	65
7.1	Setup	65
7.2	SmartAccTask Application	67
7.3	Functional Evaluation	69
8	Conclusion	79
8.1	Summary	79
8.2	Limitations	80
8.3	Implications	81
8.4	Outlook	81
A	The PROV Data Model	83
A.1	Common Elements	83
A.2	Component 1: Entities and Activities	84
A.3	Component 2: Derivations	87
A.4	Component 3: Agents, Responsibility, and Influence	88
A.5	Component 4: Bundles	90
A.6	Component 5: Alternate Entities	91
A.7	Component 6: Collections	92
A.8	Prov-DM Types and Relations	93
A.9	UML Prov-DM Component Overviews	94
B	DLT	99
	List of Figures	101
	List of Listings	104
	List of Tables	105
	Glossary	107
	Acronyms	113
	Bibliography	117

Introduction

The following chapter provides insight into the motivation, research questions, and a general overview of the fundamental idea of this thesis. Further, it discusses the methodological approach, structure, and expected results.

1.1 Motivation and Problem Relevance

Defining and identifying the key elements of accountability is a fundamental challenge in distributed systems [1]. An accountable system allows the detection and identification of responsible users and their activities in a transparent manner. In this context, the concept of accountability is also closely related to other fundamental concepts such as liability, trustworthiness, responsibility, and verifiability [1], [2].

Ideally, it is assumed that every human being will act truthfully and right. However, reality proves that people are prone to misbehavior if the system allows it. Therefore, the challenge of any system is to design a set of rules to prevent misconduct and to validate the correctness of actions. In many systems, trust is established through dependence on third parties, which requires the distribution of sensitive data and thus the loss of data sovereignty. Accordingly, a challenge for human-based activities in a collaborative computing system is the identification and definition of responsible users in order to ensure proper and correct data submission and to hold those accountable that have misused the system. That is especially difficult in the absence of centralized control authority. Hence, new concepts are required which do not rely on third parties, yet ensure that misconduct in such an environment is recognized and tracked back to the responsible user.

Consequently, the following research questions arise.

- How to define accountability and attribute it to the individual contributor?
- In what manner can trustworthiness and correctness of activities be ensured?
- Which methods exist to track misconduct and prevent it from happening in distributed systems?
- What kind of technologies can be exploited to provide accountability in a human-based collaborative computing system, without dependence on a trusted third party?
- Based on which prerequisites will a result of activity be considered as valid in a collaborative computing system?

1.2 Expected Results

The envisaged outcome of this thesis should be a prototype that provides accountability in a collaborative computing system based on human-activities. The objective is to ensure that activities carried out by people are verifiable and trustworthy using a decentralized system along with the engagement of a participating collective.

For the realization of the prototype, a suitable and standardized provenance meta-model will be created to record the activities and progress of human tasks and the actions of the peers involved. The collaborative model is generic and interchangeable for the use in a heterogeneous system to allow various applications for different use cases and scenarios. Besides, a technical background on Distributed Ledger Technology (DLT) and the assessment of different types of DLTs, this project also requires a thorough evaluation of this technology.

In summary, the projected research contributions of this thesis are:

- The analysis and comparison of selected DLTs and their properties
- Assessment and application of smart contracts and different consensus algorithms of DLTs
- Evaluation of provenance techniques in order to obtain a transparent record of activities and responsibilities
- Definition of an accountability model to be used on top of the SmartSociety platform
- Description of use-cases and requirements for the proposed solution
- Implementation of a prototype, which utilizes the technologies as mentioned above
- Evaluation and discussion of further improvements for the proposed solution to substantiate accountability in a collaborative environment

1.3 Methodological Approach

The methodological approach to reach the expected result is based on the guidelines for design science in information system research defined by Hevner et al. in 2004 [3]. Thus, to fully comprehend the background of this rather new field of Distributed Ledger Technologys (DLTs), extensive research is a prerequisite, along with a comprehensive literature research in domains of accountability, provenance, and collaborative systems.

Comprehensive knowledge of these steps will be used to develop an artifact such as a prototype based on a defined model. More precisely, a provenance meta-model with its attributes and actions defined in a standardized way. The development of the prototype consists of the implementation of an accountability component based on a collaborative task platform. It involves requirements analysis, software design, testing, and evaluation by an agile and iterative process. The DLT in combination with provenance may serve as a possible candidate to achieve accountability management.

The existing SmartSociety [4] platform for social computing, a distributed collaborative system, serves as the basis for this research. It was co-developed by the Distributed Systems Group of the Vienna University of Technology. A complete scenario will be added to the SmartSociety programming framework as a design evaluation method to demonstrate how the accountability component can be used to resolve the stated problem of this work. Finally, the results are analyzed and discussed with respect to functionality, usability, and limitations.

1.4 Structure of the Work

Chapter 2 - State of the Art This chapter outlines a brief overview of related work regarding accountability management of human-based activities in particular by utilizing DLT and provenance methodologies. Furthermore, a framework for collaborative computing is presented.

Chapter 3 - Background The background chapter introduces the fundamental concepts of accountability and provenance in collaborative computing systems. Besides, fundamental concepts of the PROV standard are described, in order to express provenance data of human activities in a transparent, coherent, and machine-readable manner. An essential part of this chapter covers the history and definition of DLT, its development as well as the core concepts and architecture based on Bitcoin. The last section briefly describes the second generation DLT and the concept of smart contracts and Decentralized Applications (DApps).

Chapter 4 - DLT Review This chapter, presents several DLTs and compares them based on their technology, architecture, and development activity, using data from GitHub.

Chapter 5 - Design The design chapter refers to the SmartSociety platform with its SmartSociety programming framework and the defined accountability model extending the SmartSociety framework. Moreover, the scope, requirements, use cases and architecture of the prototype are defined and illustrated.

Chapter 6 - Implementation This chapter describes the implementation of the proposed solution. The implemented accountability component with a well-defined provenance meta-model and underlying DLT solution extend the existing SmartSociety programming framework for collaborative task management.

Chapter 7 - Evaluation The evaluation chapter analyzes and discusses the implemented solution with respect to functionalities, usability, and performance. The SmartSociety programming framework is extended by an implementation of a demonstrated scenario which is discussed along with an evaluation of the features of the designed and implemented accountability component.

Chapter 8 - Conclusion The conclusion contains a detailed summary, discussion of the implications and limitations, as well as an outlook on expanding the research project.

CHAPTER 2

State of the Art

Bitcoin was the very first type of Distributed Ledger Technology (DLT). It was deployed in 2009 and emerged as a cryptographically secured currency with the biggest economic value in circulation [5]–[7]. In the first few years of Bitcoin’s existence, the subject and technology received very little attention and importance in research. Since then, as evidenced by literature, there has been a growing interest in the underlying technology, the design, and properties of the system, as well as a vast interest in the underlying data structure. Furthermore, its use for other applications besides cryptocurrency became a major interest, as well as, the various consensus algorithms, the pros and cons and challenges of this technology [7].

Accountability is a multidimensional concept and subject to research across different domains of science disciplines. In computer science, it is further divided into specific disciplines, such as accountability in computer networks and distributed systems [1]. Thus, the emphasis in both fields is to achieve an accountable system based on its data and information about the entities, persons involved, and their activities [8].

The innovative DLT has a wide range of applications such as privacy and data access permissions centered solutions for health-care systems or as accountable systems based on provenance to keep a clear record of ownership over an item [9]–[11]. In recent studies, an approach to define a provenance model, store and retrieve data from a DLT has been published [10]. Another approach discusses the use of DLT for provenance tracking by utilizing the DLT as a distributed access control system for health care relevant information so that the user stays in control of his private and sensitive data in respect to the General Data Protection Regulation (GDPR) [9].

DLT is also considered to be a possible candidate for redesigning our interactions with smart governance and city infrastructure. Based on the idea of a cyber-human smart city, the blockchain technology can act as a foundational layer without the dependency of a

2. STATE OF THE ART

third party for citizens to engage in collaborative actions, provided the right incentives are offered [12].

Background

In the first chapter, the terms and the meaning of accountability and the Collective Adaptive System (CAS) are presented. This chapter expands the relevant theoretical background with a focus on accountability, provenance, and Distributed Ledger Technology (DLT).

3.1 Collective Adaptive Systems

CAS is composed of a multitude of individual heterogeneous components, each of which has autonomous behaviors and distinct properties and interacts in a collectively unpredictable and complex way [13]. Adaptability is a central feature of such systems and allows each component to join or leave the collective at any time. Furthermore, the collective is adapting to the constant change in their composition and task execution goal. Components themselves can be very heterogeneous, such as reconfigurable hardware, software, and distributed systems, as well as humans. Each component can operate at different temporal and spatial scales and have individual (potentially contradictory) goals [14]. Collective intelligence is formed to reach consensus or jointly perform tasks. These CASs show the same property by indicating more functionality if they are causally linked and act as a collective rather than independently [15].

Different classes of CAS are distinguished in nature and technology, for example, insect colonies, human crowds, bio-synthetic systems, cars on the street, and computers on the internet [15].

3.2 Accountability

Accountability allows an entity or an actor to be held responsible for their actions based on their assigned identity. An agreement or commitment must be agreed upon, thus the

contractual partners or entities conduct their actions according to their respective obligations. If specific requirements are not met, it must be accounted for and result in penalties for the malicious behavior of the party that violated the contract. The correctness of states and actions has to be verifiable at all times [2]. It is a complex concept and subject of research across different domains of scientific disciplines. Achieving accountability in computer science is a fundamental research problem in computer networks, distributed systems, or information sciences in general. A transparent, immutable record of the flow of information, as well as the assignment of identities responsible for actions, plays a crucial role in achieving accountability in such environments [16], [17]. Weitzner et al [8] describe the meaning of information accountability as follows: “... *the use of information should be transparent, so it is possible to determine whether a particular use is appropriate under a given set of rules and that the system enables individuals and institutions to be held accountable for misuse*”.

3.2.1 Accountability Definition

In order to define accountability in more detail, the essential characteristics of accountability must be assessed. Lin [18] defines disclosure, liability, and non-repudiation as the most critical attributes for accountability in any context. Further, collective responsibility is included in the context of computer science and IT services due to its compositional nature. Yumerefendi and Chase [17] define three characteristics that allow participants to detect and isolate misbehavior by validating the integrity of actions and assignment of responsibilities when the observed behavior does not meet the defined specification. The three defined properties, as described in [17] are listed below.

Undeniable Actions are binding and cannot be rejected

Tamper-evident Any attempt to manipulate or corrupt the state is detectable

Certifiable The correctness of states and actions can be verified

Hence, an operative accountable system can prove integrity, and identify malicious or dishonest components to inhibit misconduct.

Throughout this work, the conceptual definition of accountability will be used:

“Accountability consists of defining governance to comply in a responsible manner with internal and external criteria, ensuring implementation of appropriate actions, explaining and justifying those actions and remedying any failure to act properly.” [2]

3.3 Provenance

Provenance information describes a log record of data or a thing in general. This information also referred to as lineage or pedigree, is crucial to determine all steps of the process and entities involved, that led to a specific result of a data item or product. Thus, provenance allows to analyze the underlying process, to track attribution and responsibility, and to decide whether a resulting data product can be trusted or not. With provenance information, basic questions can be answered, such as when was a data item created and modified and by whom? What was the source that led to the data product? Which process created the product? [16], [19]

Distributed systems enable information sharing, collaboration, and discovery without a centralized authority. For data management and distributed systems, provenance information becomes a crucial component to identify and trust the source of information that led to the product. [20]

3.3.1 Definition

Provenance has a wide range of different definitions depending on where it is applied. The following definition is used as a guideline throughout this thesis.

“Provenance is defined as a record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing. In particular, the provenance of information is crucial in deciding whether information is to be trusted, how it should be integrated with other diverse information sources, and how to give credit to its originators when reusing it. In an open and inclusive environment such as the Web, where users find information that is often contradictory or questionable, provenance can help those users to make trust judgements.” [21]

3.3.2 Key Components of a Provenance Management Solution

There are three decisive components for a provenance management solution. A capture mechanism, a representation model and a storage infrastructure solution for storing, querying and retrieving provenance information [19], [22]. With regard to this thesis, the three key components are described in more detail. An overview of all characteristics is illustrated in Figure 3.1.

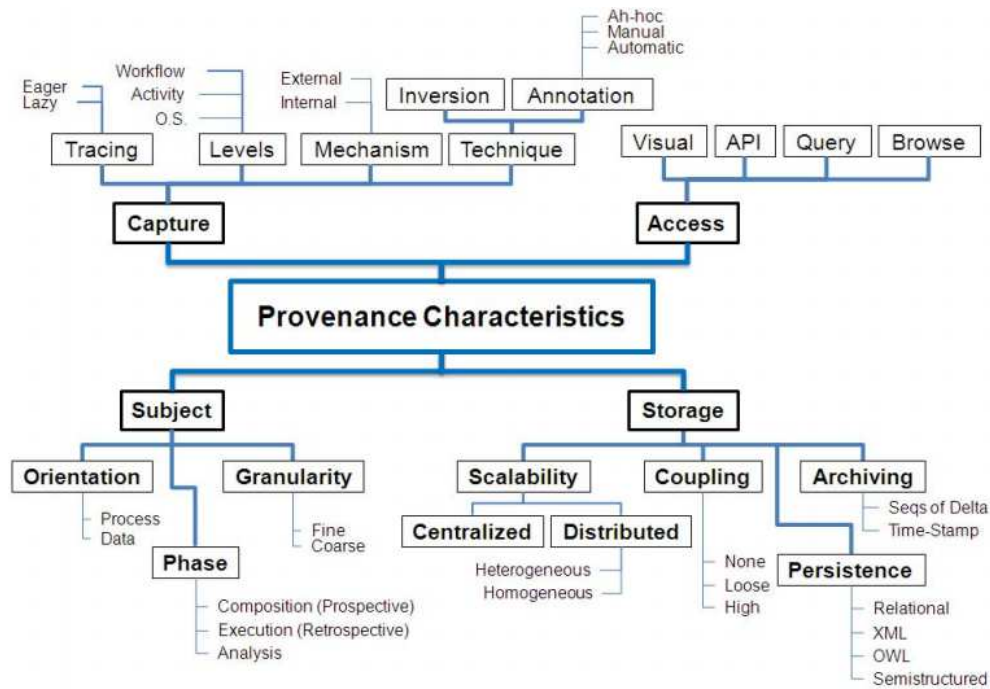


Figure 3.1: Overview of provenance characteristics [23]

Capture Mechanism In case of a computational task or any task in general, the capture mechanism requires access to the relevant details of the task, such as the progress and its specific steps, execution information and user-specific information.

Representational Model The representational model for provenance information has a direct impact on the cost of recording and its storage as well as their usage. The two main approaches to retrieve a representational model for provenance information are annotations and inversion [20], [23]. Annotations can be seen as metadata that include lineage information of a data product and descriptions of source data and processes. A metadata standard for derivation history was established in 2013 by Moreau and Missier, called PROV standard (section 3.4). The inversion method, on the other hand, identifies the source data by inverting its derivations.

Storage and Access Provenance information can grow rapidly with respect to the represented data, depending on how fine-grained the underlying data and the related provenance information is captured. Therefore, scalability needs be well considered in any solution. When maintaining data, it must be taken into account whether the data is immutable or if it can be updated or even versioned in order to be able to display the current state of its predecessors and the system. Concrete approaches include different semantic web languages, Resource Description Framework (RDF) and different dialects of Extensible Markup Language (XML) data, stored as files or as tuples in relational or graph databases.

3.3.3 Properties of Reliable Provenance

The lineage of data must have specific characteristics in order to be reliable [16], [24].

Confidentiality The confidential treatment of provenance records is essential because of the sensitivity of the information in the records. Unauthorized access should be prevented as well as the possibility to infer sensitive information by viewing a subset of the provenance data.

Integrity The integrity of data is an essential aspect of reliable provenance. It has to be ensured that the provenance data is and was not being tampered and can be validated.

Authenticity The authenticity of the data describes the ability to determine who has generated the provenance record. It includes the ownership and identity of the data.

Data Quality and Trust A reliable collection of provenance records has to ensure trustworthy and accurate collection mechanisms.

3.3.4 Provenance and Accountability

Provenance can serve as an essential element for accountable systems. The explicit representation of past processes makes it possible to trace the origin of data, actions and decisions. This makes the system transparent and allows conclusions to be drawn about compliance or violations within the system. The resulting trust in the system depends to a large extent on the tamper-proof and certifiable storage of the provenance information. [16]

3.4 The PROV Standard

The PROV standard is a provenance specification created by the World Wide Web Consortium (W3C) Provenance Working Group in 2013 [25], [26]. The specification defines the interoperable exchange of provenance information in heterogeneous distributed systems such as the web. It was developed based on the preceding Open Provenance Model (OPM) to generate a more flexible and interoperable ontology and data model to capture provenance information in a standardized format. The PROV core concepts are entities, activities, and agents (Figure 3.2). These core concepts can represent the derivation of an entity, which can be a physical or digital object. The responsible party, defined as an agent, and the required steps (activities) display their involvement and how the entity was used or created. With these basic concepts and a set of defined relations, it is possible to map the lineage and state changes of an object or data. [27]

One of the key documents of the PROV specification is the conceptual PROV Data Model (PROV-DM), that describes a generic model for provenance to capture domain and application-specific provenance information in a standardized vocabulary. It contains a mapping of the above core concepts as PROV-DM types and relations (Figure 3.2). The PROV-DM can be serialized in various formats, such as RDF, XML or as a native notation called PROV-N. The serialization implementations provide a convenient way to handle and exchange data between heterogeneous environments. [28]

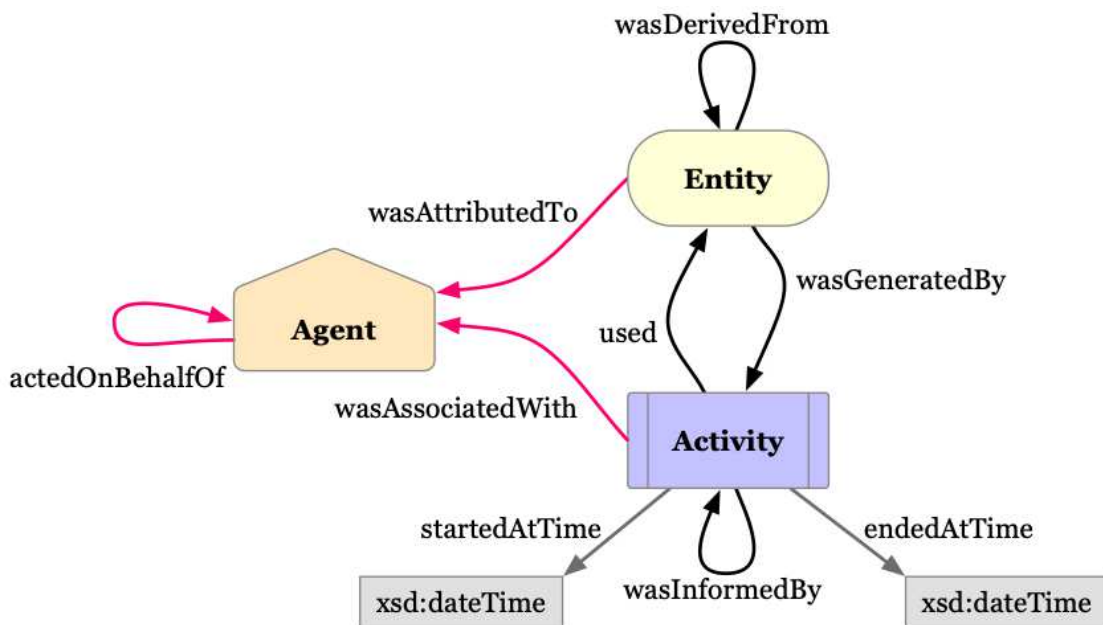


Figure 3.2: Provenance Core Concepts (PROV-DM Types/Relations) [29]

3.4.1 PROV Document Overview

The PROV standard consists of 12 documents ¹ that represent the specifications by W3C [26]. The most important documents are listed below.

PROV-DM This document describes a generic data model allowing to express provenance and to transform it into different representations. It is domain agnostic but can be easily extended with domain or application-specific descriptions [21].

PROV-CONSTRAINTS Defines a set of constraints that apply to the PROV-DM and serve as the primary purpose of validation. [30]

PROV-O This specification is an ontology using Web Ontology Language 2 (OWL2) to allow the mapping of PROV-DM to RDF graphs. [29]

PROV-N The PROV notation is a syntax for serializing the PROV-DM in a human-readable format. [31]

PROV-AQ Describes the mechanism for querying and accessing provenance with standard web protocols such as HTTP. [32]

PROV-PRIMER This document presents an introduction and guide to the PROV-DM. [33]

PROV-XML Defines an XML schema for the PROV-DM for serialization in XML. [34]

3.4.2 PROV Data Model

The document PROV-DM ² describes a generic data model standardized by the W3C. The data model defines three different types and their relations, mapped from the core concepts of PROV. An overview of the core concepts is illustrated in Figure 3.2. The nodes represent the three specific types of the data model, an entity, an activity and an agent. Directed edges represent relations between those PROV-DM type elements, such as generation, usage, association and derivation. In addition, each of the elements can be provided with attributes, both predefined, such as type and role, and custom-defined, such as status and version.

Another important aspect is the different perspectives that provenance can be recorded of. Luc Moreau describes three specific views in his introduction to PROV [26]. These are the data flow view, the process view, and the responsibility view. The data flow view is focused on the transformation of entities and the data flow within a system. Additionally, it is possible to capture the process view within a specific system that includes activities with their chronological information. Another aspect is the responsibility view, which focuses on assigning responsibility for specific activities and entities. Figure 3.3 illustrates

¹<https://www.w3.org/TR/prov-overview/>

²<https://www.w3.org/TR/prov-dm/>

3. BACKGROUND

an example of a provenance graph for editing an article. It includes the data view of the document entity, the process view for editing the document, and the responsibility view, since the responsible persons are also shown with their roles.

To be able to trust the recorded provenance information it might be useful to include information about who generated the provenance records. This can be described in a structure, called Bundle, as provenance of provenance.

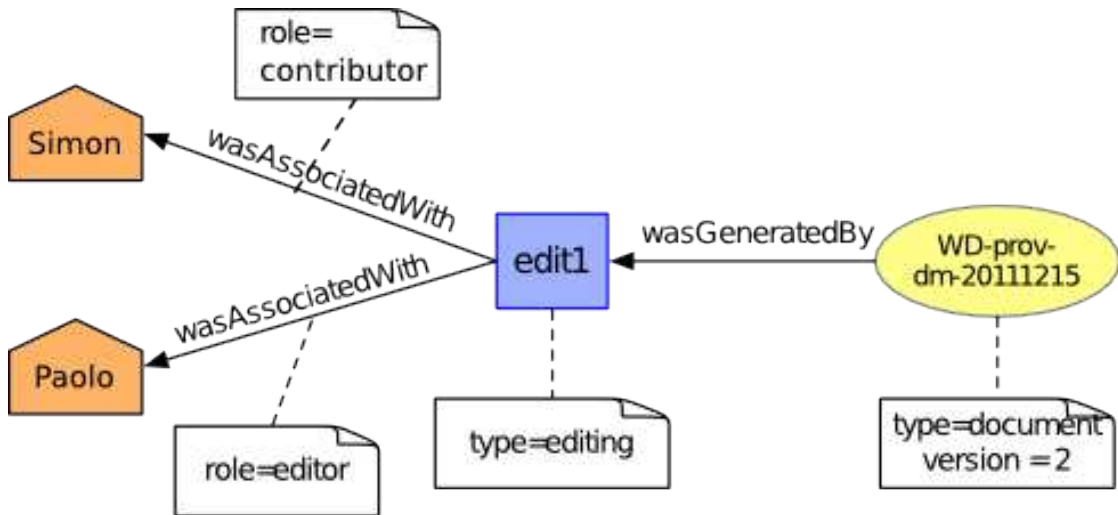


Figure 3.3: PROV graph for writing and editing a document (author's view)[21]

The types and relations in PROV-DM are categorized into six different components. An overview of the different components and a short description can be found in the following table 3.1. Detailed descriptions and explanations of the common essential elements and each component can be found in appendix A - The PROV Data Model.

Component	Core Structures	Description
1 Entities and Activities	yes	about entities and activities, and their interrelations
2 Derivation	yes	about derivation and its subtypes
3 Agent and Responsibility	yes	about agents and concepts ascribing responsibility to them
4 Bundles	no	about bundles, a mechanism to support provenance of provenance
5 Alternate	no	about relations linking entities referring to the same thing
6 Collections	no	about collections

Table 3.1: PROV-DM Component Overview [21]

3.5 Distributed Ledger Technology

“What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party.”

- Satoshi Nakamoto [5]

3.5.1 Background

Establishing trust between unknown nodes in a distributed system, such as the World Wide Web, is a fundamental research problem.

In a distributed system, cryptography and a trusted third party can be used to provide authentication, confidentiality, and integrity in order to establish security and trust in such an environment. Trusted third parties usually act as intermediaries for secure communication, they store, and utilize large amounts of sensitive and private information. The user is relying on these trusted third parties to act honestly, store their data securely, and preserve privacy. In 2008, shortly after the financial crisis, a person or group under the pseudonym Satoshi Nakamoto proposed a cryptographically secured electronic cash system - Bitcoin - which is based on a peer-to-peer (P2P) network without the need of a trusted third party [5].

Digital cash systems or cryptocurrencies were first introduced in the 1980s, where trusted third parties served as an initial solution in order to avoid the double spending problem [35], [36]. Nick Szabo, a computer scientist, focused on the domain of cryptography, was one of the first scientists to understand the imminent danger of trusted third parties and pointed out the necessity of a new trust protocol, which would make the involvement of a third party obsolete [37], [38]. Hence, when Bitcoin was introduced, its key innovation did not reside in digital currency or other foundational concepts of the system. The innovation was the very first implementation and proof of concept for establishing a platform of trust in a distributed system without the requirement of a third party to securely transfer assets. [39], [40]

The foundation of Bitcoin is based on preexisting well-known technological concepts, such as the public key as an ownership certification, linked timestamps with the entangled data structure Merkle tree, the proof-of-work (PoW) algorithm, cryptographically secured digital money, the double spending problem, and the Byzantine Fault Tolerance (BFT) [41]–[43]. A detailed overview of the chronological history of the fundamental key concepts, which led to Bitcoin, can be found in the appendix illustrated in Figure B.1.

Bitcoin, with its blockchain type of data structure, represents the very first kind of a DLT. The blockchain is an ordered reverted linked list of blocks of data. Each block is timestamped and chained by using cryptographic hashes referencing the previous block. A block aggregates several transactions that occur on the blockchain network. Therefore, the blockchain provides a distributed ledger and data storage, respectively, which can

append only new data to prevent revision and tampering. Adding new data or transaction requires a consensus by the participating nodes in the network to verify and confirm the validity of the new data. Additionally, the blockchain is replicated on multiple nodes across different countries worldwide. Anyone can join and participate in this network.

3.5.2 Terminology DLT/Blockchain

This rather new technology is under active development and, is therefore still evolving. The terms DLT and blockchain are used interchangeably in literature, scientific papers and other resources. With the uprising evolution of this technology and possible changes to the main data structure, it is in the best interest to use the term DLT as a general term. The type of DLT is defined by the way the data is distributed, structured and agreed upon. According to these conditions, a consensus is found. [41], [44]

In this thesis the definition from a report for the UK Government Office for Science in 2015 is used.

“Distributed ledgers are a type of database that is spread across multiple sites, countries or institutions, and is typically public. Records are stored one after the other in a continuous ledger, rather than sorted into blocks, but they can only be added when the participants reach a quorum.” [45, pp. 17-18]

3.5.3 Properties of DLT

Several properties are attributed to DLTs, as for instance, immutable, transparent, and reliable. The most important property, as currently found in research, is immutability [46]–[48]. Although DLT is described in literature as immutable or tamper-resistant, it must be taken into account that most systems base their consensus algorithm on computational effort. Therefore, a consensus algorithm within such a network is only resistant to attacks if a participating node does not overtake the complete consensus protocol by having more computing power than the majority of the network combined. A more appropriate term would therefore be “Mutable-By-Hashing-Power” [47]. Transparency is another property of DLT, which is based on the fact that all participating peers in the network have access to transactions for verification.

3.5.4 Types of DLT

Different types and evolutionary stages of DLT can be determined. The first available implementation of a DLT was Bitcoin with its Blockchain structure in 2008 [5]. The blockchain is one type of DLT, which consists of blocks of bundled transactions saved in a single chain structure with reference to its previous block - this data structure can be compared to a reverted linked list.

The following paragraphs describe three classification types of DLT. [49]–[51]

Public/Permissionless DLT

A public or permissionless DLT allows anyone to read, write, and participate in the verification process of transactions, also known as the consensus process. Any node can join or leave this network which resembles a P2P system. It can be seen as resistant to censorship as it is not possible to prevent a transaction from being added to the ledger if it is valid. Examples of such a public permissionless DLT are Bitcoin and Ethereum.

Private/Permissioned DLT

This kind of DLT is restricted in regard to who can join the network, has read/write access and is able to participate in the consensus process. Thus, only a smaller authenticated, trusted group of entities is participating in such networks, in most cases, centralized and run by one organization as an entirely private DLT. The restriction to only trusted nodes allows it to use traditional BFT to reach a consensus instead of the more complex and cumbersome approach of proof-of-work. This type of DLT brings back many features of a centralized system but still with some advantages that can be useful for an organization.

Consortium/Federated DLT

With a consortium DLT, the control of the consensus process relies on a preselected set of nodes, representing different entities. For instance, multiple financial institutes can federate to operate such a DLT. These DLTs can be regarded as “partially decentralized”. The DLT Corda, created by the consortium of the major financial institutes worldwide named R3³, is one of such a consortium DLT. Another instance is the Energy Web Foundation [52] that established a federated DLT for the energy sector.

³<https://www.r3.com/>

3.5.5 Evolution of DLT

Informally DLT can be categorized into three different generations evolved from the first generation of the initial Blockchain. [53], [54]

Generation 1.0

The initial application Bitcoin started the first generation of DLT with its very first type of DLT, the Blockchain.

Generation 2.0

The next advance in technology Ethereum evolved by adding improvements to its Blockchain for example, Smart Contracts and their Turing-complete script language Solidity.

Generation 3.0

The third generation of DLT can be seen as the wave of a Directed Acyclic Graph (DAG)-based DLTs as such Hashgraph, IOTA or NANO. Their common thread is that their platforms provide a fast, fee-less, and miner-less DLT targeting the Internet of Things (IoT) industry.

However, it has to be mentioned, that the first generation DLT is still used most frequently from a developers and market share perspective. Further, due to its nature of first-mover advantage, it is also the most tested and proven technology so far within the DLT ecosystem. The third generation based on DAG structure has yet to prove its technology, since there are legitimate questions and ongoing discussions about the security and reliability of the fast, fee-less, and miner-less concept.

3.5.6 Blockchain Architecture

This section describes the integral parts of a blockchain structure. It is mainly focused on the initial blockchain Bitcoin. There is a multitude of DLT architecture implementations, but they share the same key concepts.

Peer-to-Peer Network

The peer-to-peer (P2P) network is a distributed system of interconnected computers that does not rely on any central coordination by a party to facilitate the interaction. Each participating computer in this network is referred to as peer or node and is considered to be equal. Nodes act as a server and client, meaning they are both consumers and suppliers of resources, which is the main difference to the traditional client-server model. Therefore, the topology of this network is considered to be non-hierarchical or flat. The main advantage is that the data held on this network is not stored at one centralized point. Hence, there is no dependency on a single party that controls the entire network.

The P2P network architecture builds the foundation of the core characteristics of a DLT as it removes the dominant trusted third party and therefore enables decentralization of control, fault tolerance, and a resilient and open network. [6]

Cryptography

Asymmetric cryptography is based on one-way mathematical functions and is therefore considered as irreversible. It consists of a digital key pair, a private confidential key and derived from it a public key, which can be shared with anyone. This type of cryptography enables both authentication, by signing and verifying messages, and encryption of data.

With DLT, the private key is a 256-bit long random number and used to sign transactions digitally and to send funds. Ownership of the private key is essential since it gives the owner control over all funds connected with the corresponding address.

Wallet Address

A wallet is either a file or a database containing the digital key pairs of public and private keys. The owner's address is represented by a more compact and obfuscated public key. These digital fingerprints are usually calculated with Secure Hash Algorithm (SHA) and encoded with Base58Check. This enables human readability and error detection.

Merkle Tree

Merkle tree is a binary tree of hash pointers and allows a very efficient way of storing and verifying the integrity of included data elements. The leaf nodes represent cryptographic hashes of data input, whereas the non-leaf or internal nodes is the combined hash value of their child nodes. The tree is constructed from the bottom up until a remaining node with its hash is reached, the Merkle tree root. In the case of an odd number of data inputs or leaves, the single child node is copied and paired with itself for the hash value of its parent node.

DLT, with its P2P network architecture benefits of this structure as it allows lightweight clients to only process the Merkle tree root for a block of transactions.

Linked Timestamping

Linked timestamping represents a connected signed data structure with chronological order. The first concept of such a structure was built based on the idea of a notary service that evidences the last creation and modification date of digital documents [55]. Each document contains a timestamp that points backward in time to a previous document forming a long immutable and locked chain of documents. Modifications to one of these linked timestamped documents would render it invalid due to its chained dependency. The efficiency of storing linked timestamps can be increased by combining several documents in blocks and using the Merkle tree structure instead of a linear chain of hash pointers [41], [56].

Transaction

Transactions represent state changes of the DLT and can be seen as a transfer of data ownership. As shown below in Figure 3.4, ownership is transferred digitally by signing a hash of the previous transaction and the hashed public key of the next owner.

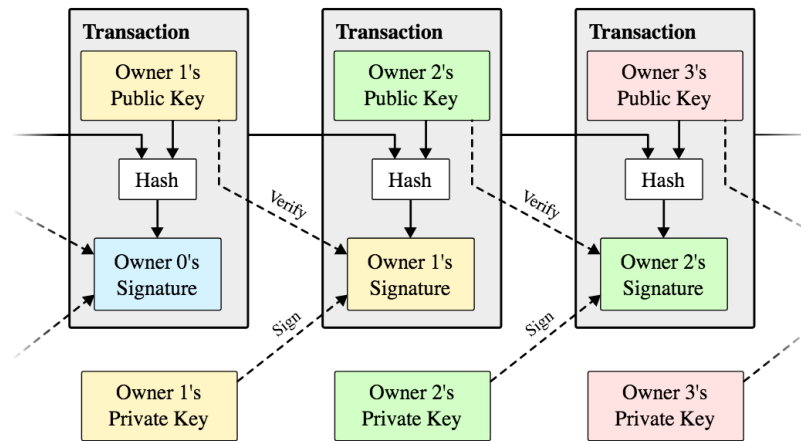


Figure 3.4: Bitcoin transactions - The chain of ownership [5]

Every node receiving a new transaction needs to be verified, based on specific protocol rules. If the verification fails, the transaction is being rejected. Otherwise the node includes an unconfirmed transaction into a local transaction pool, also known as Mempool and then propagates it further to its neighbor nodes. From there, mining nodes select transactions out of the transaction pool to include them into blocks that are added to the ledger.

Blocks

The above-described concepts of linked timestamping, Merkle tree, and transactions are essential compounds of a block. Transactions are aggregated into timestamped blocks, which in turn are added to an ordered reverse-linked chain, the blockchain. A specific block is considered to be secured, referenced as immutable, in the chain after six consecutive blocks have been added on top of the current block. The block height defines its current position in the list or chain of blocks (Figure 3.6), with the Genesis Block being the very first one starting with the index zero.

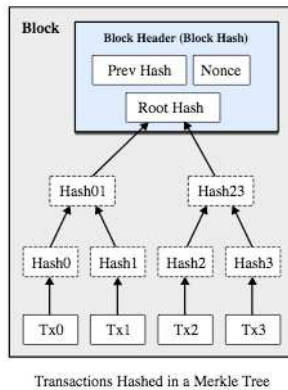


Figure 3.5: Block Structure [5]

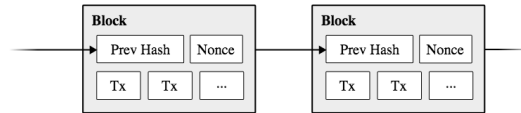


Figure 3.6: Linked list of blocks - Blockchain [5]

Consensus Protocol

A consensus protocol is used for the joint commitment and compliance with certain network rules between individual participating nodes of the P2P network. These network rules apply to all transaction-based state changes of the DLT as well as how they are included in the distributed ledger and under which circumstances a transaction is valid.

Scripting language

Scripting languages are often used to extend the DLT with functionality, such as specific types of transactions or contract code executed by the DLT. The Bitcoin protocol includes a scripting language *Script*, which is a Forth-like stack-based execution language deliberately designed to be stateless and not turing-complete. These properties result in limited complexity and predictable execution time by avoiding unintended side effects, such as infinite loops. There are different types of transactions scripts based on this language for Bitcoin.

3.5.7 Consensus Protocols

The crucial and main attribute of all DLTs is to reach consensus, or an agreement, for the global ledger in a network of untrustworthy connected nodes. This is a fundamental problem within distributed systems as described with the well-known Byzantine Generals Problem [57]. In this stated problem multiple generals, each commanding their division of the overall Byzantine army, encircle a city to be captured. They have to coordinate a common command, either to attack or retreat, through messengers since the generals and their divisions are physically separated. Generals and messengers may not be trusted within this scenario since they might fail to deliver the message or forge the message on purpose. This issue can only be solved with a majority agreement by loyal trustworthy participants. Distributed systems that prevail the Byzantine Generals Problem are characterized as Byzantine Fault Tolerance (BFT).

In case of blockchains or other types of Distributed Ledger Technology, the consensus algorithm validates and agrees on a common transaction history. Several different consensus algorithms that claim Byzantine Fault Tolerance have been implemented in different DLTs, and the main consensus mechanisms are summarized in the following subsections [42], [49], [58].

Proof of Work

Proof-of-work (PoW) is a consensus algorithm used by the Bitcoin or Ethereum network to achieve BFT. The main idea of this protocol is the requirement to solve a cryptographically complex problem in order to validate a block of transactions and agree upon a system-wide standard view. The entire network can easily verify the solution found to the given problem.

In Bitcoin, the computationally expensive problem is to find a valid block header hash value, based on a double SHA-256 hash function, to suggest a block to be added to the blockchain. Participating nodes are called miners, and the process is referred to as mining. A critical aspect of hashing functions is that they are random, and by that, each miner will have a probability of finding the next valid hash value proportional to their total invested computational power. Miners create a new candidate block and try to calculate a valid block header hash value by adjusting a nonce value for this block. The consensus algorithm requires a certain amount of work to find a valid hash value. It is controlled by the criteria that hash values are required to be less or equal to a specified target value. This is controlled by the criteria that hash values are required to be less or equal to a specified target value. The lower the target is the more difficult it is to find an appropriate value. Its difficulty is adjusted every 2016 blocks, depending on the overall computational power within the network, to maintain a verification rate of approximately ten minutes per block. Once a miner finds a valid solution, it is broadcasted to the entire network, which in turn can easily verify its correctness based on the values in the block. The used computation power to find a valid solution is rewarded with a block reward and transaction fees, which act as an incentive to participate in this consensus algorithm. Further, the required, artificial computational cost secures the network and detects double spending or Sybil Attack attacks. Only if the attacker has constant control over the majority, 51 percent or more, of the computational power within the entire network an attack such as double spending would be possible. The underlying theory is that it is much more difficult to control the majority of computational power compared to the number of identities or participating nodes. It is possible that multiple miners simultaneously find a valid solution for a block. Hence, the network will temporarily be partitioned, also known as forking, but eventually converge to the longest chain of blocks.

Proof of Stake

The proof-of-stake (PoS) mechanism resolves the wasteful computation power competition from PoW. Each participant's voting power is proportional to the invested amount of cryptocurrency they possess, referred to as the stake. The risk of losing investment due to

dishonest behavior is the incentive to maintain honest voting [59]. Since a participant with the highest amount would be the most dominant one in the network, other solutions to select the next block generator have been proposed by combining it with specific criteria. Such additional criteria can be a randomized approach like in NXT ⁴ or Blackcoin ⁵ where the next block generator is selected by providing the lowest hash value, a simple PoW approach, in combination with the size of the stake. In Peercoin ⁶ the age of coins combined with the size of the stake is taken into account for finding the next block generator.

The high energy consumption by blockchains relying on PoW algorithm sometimes leads to an adoption of their consensus algorithm to PoS. Ethereum is an instance which considers moving from a PoW algorithm to a PoS mechanism called Casper.

Delegated Proof of Stake

In the consensus algorithm delegated proof-of-stake (DPoS), the delegates are elected by stakeholders. The voting power that the stakeholder has is determined by the amount of the base token, or cryptocurrency, that the account is holding. Delegates maintain and secure the network by alternately creating and validating blocks of transactions. The representative number of trusted delegates is much smaller than in PoS, which makes the mechanism more efficient.

DPoS can be compared to a representative democratic system. It is considered to be more efficient concerning energy consumption and block creation and validation than PoW and PoS.

DLTs utilizing DPoS are, for example BitShares and SteemIt. BitShares ⁷ is a decentralized financial exchange platform and one of the first blockchains to introduce DPoS. SteemIt ⁸, a content-driven social media platform based on Steem ⁹, combines Proof-of-Brain [60] for distributing rewards to contributors and DPoS for the general block generation and establishing a reward pool [61].

Practical Byzantine Fault Tolerance

One of the first solutions to the Byzantine Generals Problem was an algorithm for the state machine replication that tolerates Byzantine faults, referred to Practical Byzantine Fault Tolerance (PBFT). PBFT's cycle of block generation is referred to as rounds. A round is based on a three-phase protocol with the following phases: pre-prepare, prepare and commit. In every round, a new primary node is elected as the leader to determine a new block with a defined order of included transactions. The candidate order for the

⁴<http://www.nxtcrypto.org/>

⁵<https://blackcoin.org/>

⁶<https://www.peercoin.net/>

⁷<https://bitshares.org/>

⁸<https://steemit.com/>

⁹<https://steem.com/>

3. BACKGROUND

transactions is then propagated to all the other validation nodes, called replicas. At the end of a successful round, the block is committed and added to the blockchain. Each phase needs a majority vote of two-thirds of all validating nodes to proceed to the next phase. The algorithm of PBFT can handle up to one-third of faulty byzantine replicas. Since, all participating nodes must be known to the network, the consensus algorithm is best utilized in private or permission-based blockchains, such as Hyperledger Fabric ¹⁰. The Stellar Consensus Protocol (SCP) adopts PBFT by allowing its participants to choose a set of nodes to be trusted. This federated model enables a Byzantine agreement with a specified set of nodes, without involving all nodes.

¹⁰<https://www.hyperledger.org/use/fabric>

3.6 Ethereum and Smart Contracts

The concept of smart contracts was first introduced by Nick Szabo [37].

“A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives of smart contract design are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitration and enforcement costs, and other transaction costs[1].” [37]

With the introduction of DLTs and especially Ethereum [62], the concept of smart contracts was implemented and made available on decentralized and cryptographically secured networks. Scripting languages like Solidity by Ethereum provide the tools to implement such a computer program, a smart contract, with automated enforcement when specified terms or requirements are fulfilled. These smart contracts are deployed on the blockchain itself, making them a cryptographically enforced agreement, which is available and transparent for its users. One use case for a smart contract is crowdfunding to fund a project by contributing users in a decentralized manner. Users can send a transaction to a smart contract address on the blockchain to trigger an action defined in this contract, such as sending back a new token for their contribution, representing their share on the project. [7], [63]

3.6.1 Smart Contract Structure

Smart contracts are a set of functions and state variables written in a high-level language, such as Solidity [63]. The code of such a contract is compiled into low-level bytecode for execution on the Ethereum Virtual Machine (EVM). A local instance of the EVM is being run on each Ethereum node. The Ethereum platform is essentially a distributed replicated virtual machine operating with the same initial state and producing the same final state as its result is deterministic. An address identifies every contract, that is derived from its contract creation transaction. In contrast to externally owned accounts contract accounts do not have private keys and therefore, cannot be owned by their private key. Contract accounts are owned and controlled by the logic of its deployed smart contract code. An essential aspect is that smart contracts are only executed by an initial transaction from an externally owned account. Contracts can then call other smart contracts. They run only upon a transaction triggering its execution either directly or indirectly as part of a chain of contract calls. Smart contracts never run in the background or in parallel. The EVM can be considered as a single threaded machine.

Transactions are atomic and are therefore fully executed if they are error-free. Failed transactions are recorded as failed attempts, and the execution costs for the failed attempt are being deducted from the originating account. An essential and fundamental

architectural design element in Ethereum is gas. Gas is a unit for measuring the required computational and storage resources to perform a transaction on the Ethereum blockchain. It can be considered as the computational fee for performing a transaction on the network paid to the miners. This separate unit allows having a distinction between the actual valuation of the cryptocurrency and the computational cost. It also prevents denial of service attacks and poorly written functions from executing in an endless loop.

After completing a successful or failed transaction, a transaction receipt is created and saved on the blockchain. Transaction receipts store information about the execution of the transaction, including log entries, which refer to events emitted by a respective smart contract. Events can be defined in a smart contract and represent a logging mechanism by emitting these event objects to the transaction log. The transaction log does not cost as much as state variables that rely on the EVM storage directly. Smart contracts cannot access events. External applications can read the transaction logs by listening or searching for these events generated by the smart contract. Therefore, external applications and its user interface can react to changes at the smart contract by listening for these events.

3.7 Decentralized Applications

Decentralized Applications (DApps) are web applications that are partially or entirely decentralized and executed on a distributed computing system. Ethereum as one instance of a DLT became a major platform for developing, building, and running DApps, due to its architecture and concepts as well as the incorporation of a Turing-complete scripting language. The core components of a DApp are besides a decentralized controlling logic based on a smart contract, a decentralized storage system, and communication protocol, as depicted in the following Figure 3.7. Ethereum envisioned from the beginning to reinvent the web, hence the naming of this compound of three core elements as web3 suite. The different aspects of a DApp are briefly described below. [63]

Backend (Smart Contract) Smart contracts store the business logic and associated application status. It can be seen as the backend component of the application. Smart contract code is expensive to deploy and execute, hence it is necessary to identify the essential aspects to be executed on a trustworthy and decentralized execution platform. The inability of code changes and deletion is also a major concern and should be considered when building the application. Because of the cost of deployment and execution, the smart contract logic often depends on external components, such as centralized data. Therefore, there may be concerns about the degree to which this depends on the external system and its trust in them.

Frontend (User Interface) The client-side of the DApp is built upon standard web technologies. Interaction with Ethereum and its smart contract, such as signing and sending messages, managing accounts, is often abstracted from libraries like `web3.js` and a browser extension MetaMask wallet. These make it possible to connect to an Ethereum node and interact with them.

Data Storage The capacity of data storage for DLT, in general, is limited. The main limiting factors for Ethereum are the block gas limit, the amount and cost of the gas, and possible network synchronization issues. Data privacy can also play an important role. Therefore, metadata and data are often stored off the Ethereum blockchain on a data storage platform, either centralized or decentralized. The InterPlanetary File System (IPFS) is one example of such a decentralized data storage platform. It is a P2P hypermedia network protocol for storing and sharing data in a distributed file system. Orbit-DB is built upon IPFS and provides a serverless, distributed P2P database. Ethereum develops its own decentralized data storage protocol named Swarm.

Due to the above-mentioned limiting factors of data storage, concepts and mechanisms are used to minimize the data stored on a chain and, at the same time, to outsource linked data to external storage systems. The digital fingerprint, a hash, of the data is used to reference and anchor data to a DLT in an efficient way. Merkle tree structure can further be used to maximize data being stored, identified by its hash. Chainpoint utilizes the Merkle tree optimization by providing a protocol for anchoring data onto the blockchain and verification proof.

Communication Protocol The communication protocol is responsible for the exchange of messages between applications, different instances of the application, or users of the application. The most popular distributed, P2P messaging protocol for DApps is Whisper, as part of Ethereum.

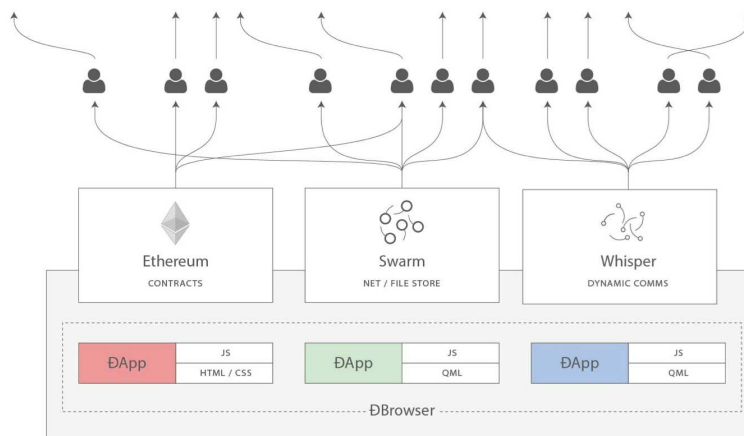


Figure 3.7: The web3suite of a DApp.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

DLT Review

This chapter discusses and reviews several Distributed Ledger Technology (DLT) solutions based on their development adoption, architecture and consensus mechanism.

4.1 Bitcoin

Bitcoin's blockchain was the very first introduced DLT without requiring any central trusted authority [5]. The concept was described and published in 2008, followed by the official release of the Genesis Block transaction, the very first transaction, on January 9, 2009, on the mainnet. The core concept and its architecture are described in detail in section 3.5.6. It remains still one of the most extensive and most proven infrastructures based on the market capitalization, market share, transactions, users, and development activities.

4.2 Ethereum

Ethereum is a distributed platform based on the Blockchain data structure. The main idea behind the release of this new DLT in 2015 was to overcome the limitations of Bitcoin by offering a more generalized DLT and support for Decentralized Applications (DApps) with higher complexity.

“Ethereum does this by building what is essentially the ultimate abstract foundational layer: a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions.” [62]

The consensus algorithm is based on proof-of-work (PoW). The block interval is kept to approximately fifteen seconds to maintain a fast throughput for transactions, confirmations, and contract runtime. Ethereum's core protocol is a native Ethereum Virtual Machine, which enables code execution of arbitrary algorithmic complexity. Each node validating blocks in the network also has to execute the code of smart contracts and other operations. Based on the consensus mechanism, an agreement on the execution result of smart contracts is also being accomplished. Hence, computational steps are costly. Solidity, as a Turing-complete scripting language, provides the possibility to create almost any program in any complexity. It can lead to very long runtimes caused by either intention, bugs, or attacks on the network, such as a Distributed Denial of Service (DDoS) attack. As a prevention and implicit block size limit, a computational fee in the currency of gas is required. This currency unit gas represents the fee per computational step and prevents the non-termination of programs as described in the halting problem. Each transaction needs to define a gas amount to be consumed for processing. The state in Ethereum is based on account objects, which include the current balance. Two types of accounts exist the externally owned account, controlled by the private key of its user, and designated contract accounts controlled by the defined contract. Messages are used to invoke specific functions at the contract account or more general to make state transitions within the Ethereum network, such as a transaction to transfer a specific amount of the currency between accounts.

Ethereum, as a second generation of DLT (section 3.5.5 - Evolution of DLT), provides a highly established platform with vast developer resources, such as frameworks, tools, guides and documentation.

4.3 IOTA

IOTA is a cryptocurrency designed explicitly for the Internet of Things (IoT). Its development started in 2015 followed by the mainnet launch in 2016.

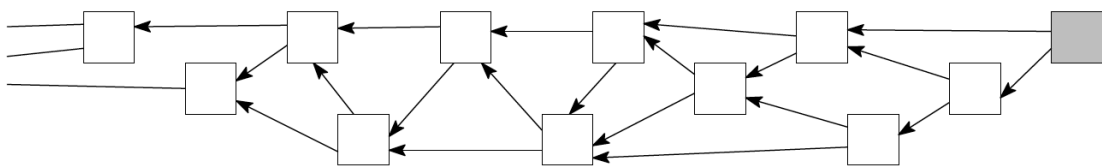


Figure 4.1: IOTA - The Tangle [64], [65]

The public permissionless distributed ledger is based on a Directed Acyclic Graph (DAG) based data structure defined as Tangle (Figure 4.1). In comparison to the Blockchain structure, transactions are stored in a graph data structure instead of a chain of blocks. Vertices in this DAG represent transactions, and edges indicate approvals. A transaction is a simple object consisting of an address, value, signature, tag, and other fields. All

transactions have a path either directly or indirectly to the Genesis Block transaction, the root of this structure. Transactions that are not reachable by other vertices, respectively, transactions are referred to as tips. Tips represent transactions that are not approved yet. Every new transaction has to verify two other transactions. Tips are prioritized over older transactions. For the verification process the client has to compute a minimal version of PoW, similar to Hashcash [64], [66], in order to broadcast the new transaction to the network. This mechanism is being used to avoid spam or Sybil Attack attacks. Both of them require that the attacker broadcasts a high volume of transactions to the network. Weight is being assigned to each vertex (transaction) which is proportional to the effort being made by the node. The weight is used, once the volume of transactions is high enough, for the consensus algorithm. Until the network stabilizes in terms of running nodes and throughput, it works with the help of a coordinator. The coordinator is a centralized undisclosed node operated by the IOTA Foundation to enforce the consensus mechanism at the network. Milestone transactions are created by this coordinator that serve the purpose of validating and confirming transactions, thereby preventing starvation in the network consensus protocol. However, the source code of the coordinator is not publicly available for unknown reasons. This consensus protocol is referenced as Tip Selection Algorithm (TSA) and enforced by the coordinator for now. There is ongoing research to find a replacement for the centralized coordinator and its consensus mechanism [67], [68].

It is currently unclear whether protocol safety and liveness are guaranteed due to the remaining concerns of the undisclosed coordinator operated by the IOTA Foundation [50]. There are also security concerns about the implementation of a custom cryptographic hash algorithm in the IOTA project [69]. Transaction history research has shown that there is no easy access to the entire Tangle data structure since snapshots are saved to keep the size of full nodes to a minimum. Although some database dump files can be accessed, the location of the Genesis Block transaction could not be retrieved.

4.4 Nano

Nano is a cryptocurrency introduced in 2014 and initially known as RaiBlocks [59]. Their sole purpose as stated in their whitepaper [59] is to become a global high-performance currency with a secure and decentralized network and instantaneous transactions with zero fees.

The data structure is designed as a Block Lattice, illustrated in Figure 4.2. It consists of wallets and validator nodes. Every user, respectively account, has its own blockchain (account-chain) as part of a larger DAG. The account chain represents the transaction and balance history for the respective account. Only its account owner can conduct the blockchain update. It is updated asynchronously with the network and therefore enables an instant, feeless and scalable payment solution. Transactions include account balances instead of an amount per transaction. One transaction consists of two blocks, the sender and the receiver publish a matching block, each signed by the respective owner of the

account chain. Send and receive transactions consist of the following fields: previous, work, type, signature. The balance field is only included in the send transaction. To prevent an attack by flooding the system with transactions, each new block published to the network has to populate a valid user-generated PoW value, similar to the Hashcash concept [66]. This calculation lasts milliseconds to seconds and can be precomputed for the next transaction to increase the performance.

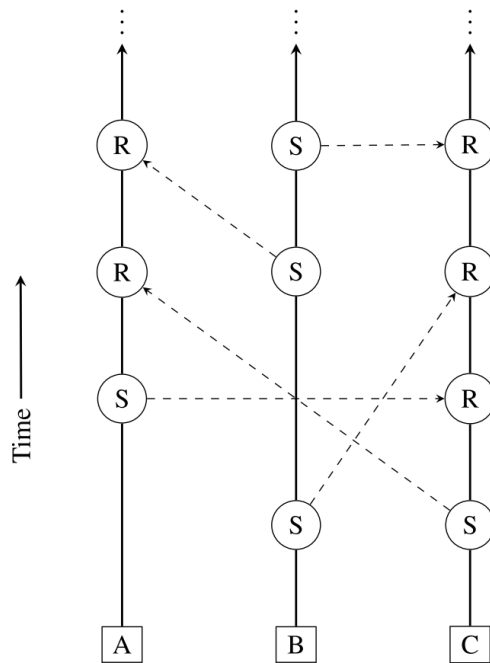


Figure 4.2: Nano Block Lattice with a send (S) and receive (R) block, signed by their chain owner [59]

The consensus protocol is called Open Representative Voting (OPV) by Nano and describes the delegation of their balance as voting weight to selected representatives. This mechanism is also known as the delegated proof-of-stake (DPoS) protocol.

4.5 Comparison

The background section 3.5 on DLT describes in detail the core concepts and properties of such a platform of trust, a distributed system without relying on trusted third parties as intermediaries for establishing trust [39], [40]. The research on this rather new domain is still ongoing, and many changes are discussed along the way. Regardless of the underlying technology, implementation and architecture, DLTs share common characteristics, such as a consensus protocol, the immutability of transactions, and the absence of a central authority. Table 4.1 summarizes and compares several of these characteristics for different DLTs. Also, a comparison of the development activity for every DLT is presented in table 4.2 with reference from its GitHub repository. Taking into account the number of contributors, watchers, and forks, Bitcoin and Ethereum show the highest activity. Furthermore Ethereum and its community provide the most resources for developers, such as guides, documentation, frameworks and tools.

	Bitcoin	Ethereum	IOTA	Nano
Consensus Protocol	PoW	PoW (proof-of-stake (PoS))	Tangle	OPV/DPoS
Block Interval	10 min.	15 sec.	instant	instant
Type of DLT	public	public or private	public or private	public
Created/Genesis Block	2009-01-03	2015-07-30	2016	2014
Data Structure	Blockchain	Blockchain	Tangle/DAG	Block Lattice/DAG
DLT Generation	1	2	3	3
Smart Contract Support	yes (limited)	yes	no	no
Suitable for DApps	no	yes	no	no

Table 4.1: DLTs in comparison

	Bitcoin ¹	Ethereum ²	IOTA ³	Nano ⁴
Commits	23,068	11,531	2,583	4,276
Branches	7	28	7	33
Releases	231	136	62	129
Contributors	684	454	62	56
License	MIT	LGPL-3.0	GPL-3.0	BSD-2-Clause
Watchers	3.5k	2k	144	245
Stars	42.5k	25.5k	1.2k	2.6k
Forks	25.3k	9.3k	423	580
Latest Release	2019-11-24 (0.19.0.1)	2020-02-18 (v1.9.11)	2020-01-05 (v1.8.4)	2019-11-12 (v20.0)
Latest Commit	2020-03-07 (1 day ago)	2020-03-02 (6 days ago)	2020-03-05 (3 days ago)	2020-03-06 (2 days ago)
Pull Request open/closed	353 / 12,389	87 / 4,974	36 / 659	34 / 1,762
Issues open/close	765 / 4498	285 / 4,433	379 / 646	106 / 739
Language	C++	GO	Java	C++
Developer Resources	+	++	+	-

Table 4.2: GitHub development activity for different DLTs - accessed on 2020-03-08

¹<https://github.com/bitcoin/bitcoin>

²<https://github.com/ethereum/go-ethereum>

³<https://github.com/iotaledger/iri>

⁴<https://github.com/nanocurrency/nano-node>



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Design

This chapter describes the design of the accountability management component, which will extend the SmartSociety platform. First, the SmartSociety and its SmartSociety programming framework will be introduced. After that, the functional and non-functional requirements are described at sections 5.3.1 and 5.3.2. The essential part of this chapter is the design of the accountability component itself. An overview of how the accountability component is linked in the context of SmartSociety is provided in a macro view of the architecture. It is followed by a more detailed architectural overview of the accountability component.

5.1 SmartSociety Platform

The SmartSociety project, funded by the European Union, is a collaborative development and research effort between ten universities and institutions. SmartSociety focuses on the Hybrid and Diversity-aware Collective Adaptive System (HDA-CAS), a particular type of Collective Adaptive System (CAS). HDA-CAS is an emerging new class of socio-technical CAS in which humans and machine computing elements complement each other and are considered uniformly under a generic term of peers. The distinction between humans and machines is blurred. Similar to CAS a collective, formed by persistent or short-lived teams of peers, represents the central work unit that achieves their goals collectively.

The SmartSociety platform effectively supports a wide range of collaboration scenarios that exist in today's social computing environments. Collective Based Tasks (CBTs) can be created by external users, and are worked on by a hybrid collective comprised of adaptive peers with humans and machines. The complete lifecycle from task creation, provisioning, negotiation, orchestration to execution is managed by the platform in a smart society application's context. The encapsulated functionality of the application context determines, besides the lifecycle management of the CBT, how collectives are formed, peer participation incentives are created and also how to retain peers. One

working example of such an application is SmartShare. A mobile user contacts this ride-sharing application to either request a ride or offer a ride in the role of a peer by providing a service for the platform as the designated driver for other users. Figure 5.1 shows a high-level model of the core components of the SmartSociety platform architecture.

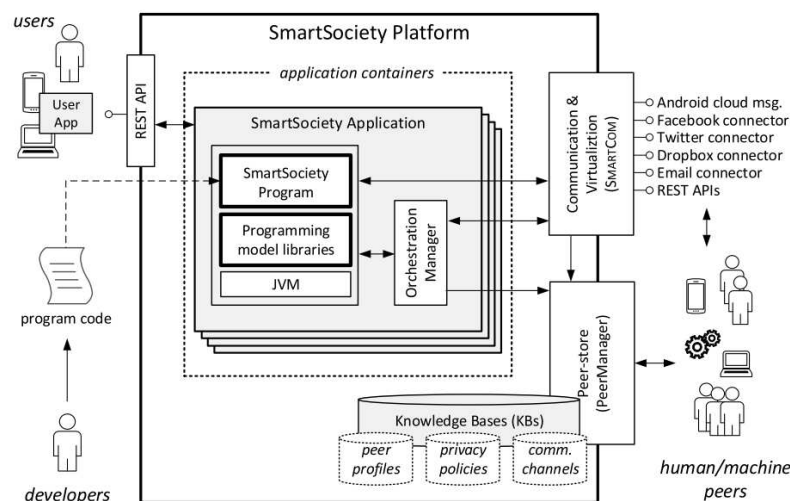


Figure 5.1: The architecture of SmartSociety platform with users and peers. [4]

The left side of Figure 5.1 displays the users, who can use the provided task execution environment as a deployed application. On the right side are the peers for whom the platform provides a collaboration environment with well-defined working conditions.

The following three components are the core of the SmartSociety platform:

Peer Manager (PM) The PM, a central privacy-preserved data storage, maintains and manages information about humans and machines, the peers. The component considers privacy principals by the General Data Protection Regulation (GDPR)¹, a regulation for data protection and privacy enacted by the European Union in May 2018.

Orchestration Manager (OM) The OM component is responsible for the provision and orchestration of collaborative activities between peers.

SmartCom SmartCom [70] is the communication middleware and therefore manages the routing and authentication of messages between the users, the platform, and the peers. It was designed as an independent component, which can be used separately with similar HDA-CASs.

¹<https://eur-lex.europa.eu/legal-content/en/TXT/?uri=CELEX:32016R0679>

5.1.1 SmartSociety Programming Model

The SmartSociety Programming Model is a working implementation of the SmartSociety platform and offers an API for writing SmartSociety applications. Various working scenarios to demonstrate the capabilities of the system are included, such as SmartShare (ride-sharing application) and RQA (an application for collecting and curating answers to requested questions) [71]. In Figure 5.2 the model with an overview of the CBT lifecycle is shown. The CBT is the central construct of the programming model and encapsulates the functionality to keep the state and manage the lifecycle of a collective task. The input for a CBT is defined in an associated TaskRequest, and the outcome of a collective task is represented in a TaskResult object. Every step of the lifecycle can be defined by the programmer of the SmartSociety application with the help of the available programming model libraries.

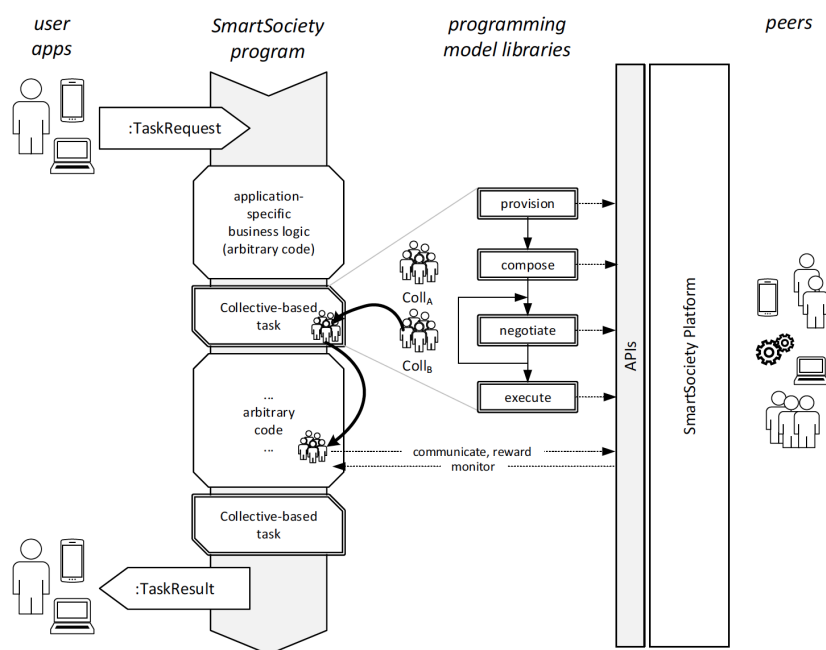


Figure 5.2: SmartSociety Programming model with CBT lifecycle. [71]

An overview and a brief description of the four states of the CBT life cycle are listed below.

provisioning The provisioning phase takes an input collective specified at CBT instantiation to find a set of human peers that are capable of performing the requested task. The result of this process is the provisioned collective that is included as input for the next state.

composition At this stage, task execution plans are compiled, consisting of ordered steps required to process the task and the designated performing peers. This state

delivers as output a list of collective negotiation items associated with composite execution plans and is passed to the subsequent state.

negotiation This stage involves a complex collaborative selection and negotiation process for one or more executing plans from the composition phase. In case that the state is started directly from the provisioning state, the execution plan is implied and implicitly understood by the selected peers. The result of this negotiation state is the only agreed execution plan and the associated collective, which are forwarded to the subsequent state.

execution The execution state handles the actual processing of the agreed execution plan by the ‘agreed’ collective.

In either case, the developer is limited to declaratively specifying the CBT’s type (handlers), the required termination criterion and the Quality of Results (QoR) expectations. The state is ended when the termination criterion evaluates to true. The outcome is ‘success’ or ‘failure’ based on the value of QoR metric.

The lifecycle of a CBT is processed depending on the collaboration model specified by the developer upon instantiation. The two collaboration models are `open_call` and `on_demand` [71]. It is possible to combine the aforementioned collaboration models. An example of such a combination is `open_call` set to false and `on_demand` set to true. Thereby this combination takes into account that the expected optimal collective peers are already provisioned, and the task execution plan is implicitly assumed or known before runtime. Therefore, the composition phase is skipped, and the negotiation phase is trivial, either accepting or denying the task.

5.2 Accountability Model

The design of the accountability management solution is based upon the SmartSociety platform, a HDA-CAS, which provides a framework for handling human-based activities in a collaborative computing environment. The SmartSociety platform is further extended by utilizing smart contracts on a Distributed Ledger Technology (DLT) to have a binding and traceable contract for the human interactions in their activities. To achieve accountability in a distributed collaborative computing system such as SmartSociety, we have to look at the meaning of accountability. Section 3.2 describes the necessity for a transparent, immutable record to track activities and actions as well as the possible identification of the responsible actors assigned to those activities. A DLT system with the concept of smart contracts can provide these described properties and brings the control and responsibility to the collective instead of depending on a trusted third party. Figure 5.3 illustrates the proposed accountability model in a high-level diagram.

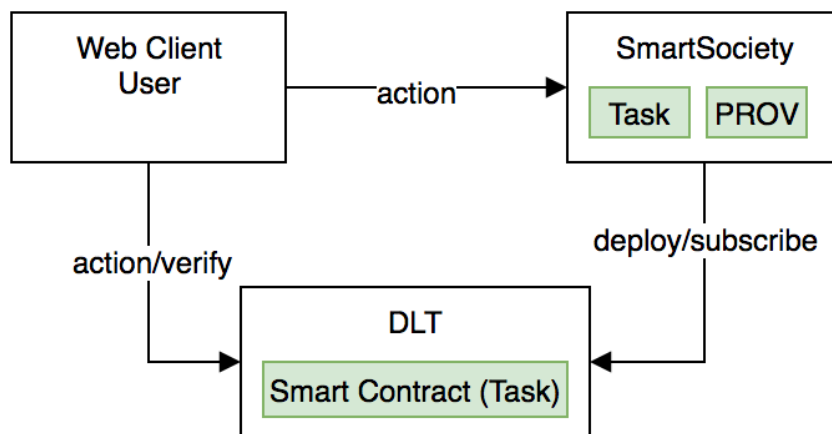


Figure 5.3: High-Level Accountability Model

5.3 Scope and Requirements

To create an accountability management module as an extension to SmartSociety, the prerequisites must be met to limit the scope of this project.

Collaborative Computing Designing a collaborative computing environment is a very complex task itself and is not the main research topic of this project. The SmartSociety is described as a framework for a distributed collaborative computing environment 5.1. SmartSociety will be the basis for our accountability model.

Lifecycle of a Collective Based Task The collective allocation process and the incentive model for fulfilling the task are themselves very complex topics. This is out of scope for this thesis and therefore it is assumed that a certain motivated collective is always found for the negotiation and execution lifecycle of a CBT.

User/Peer Management The user management is a crucial component in a collaborative computing environment. It enables correct identification, forming, and assigning peers to collectives depending on their profile.

The user management is presumed to be provided with sufficient user profiles, incentives, and activity to execute a collective based task successfully. The peer profiles are extended to have an assigned account and access to the DLT being used.

DLT As stated in the background section 3.5 and the review chapter 4 there are many different versions of DLT with different architecture and properties.

The focus remains on one implementation of DLT and that is Ethereum with an underlying blockchain architecture. In order to limit the scope of the implementation of the accountability module, the validity of the properties is assumed by a DLT such as immutability and decentralization. These properties strongly depend on the underlying technology, architecture and the concepts on which they are based. Furthermore, as already mentioned in the preceding description *User/Peer Management*, each user working with the DLT has an identified account mapped to the user in the SmartSociety domain.

PROV There are different possibilities to capture, model, store, and query provenance data. The research of different approaches and their evaluation is out of the scope for this thesis.

A model mapped to the domain of SmartSociety is developed and being used in this implementation as the basis for capturing provenance records.

5.3.1 Functional Requirements

A software prototype for the accountability component was created. The specification of the behavior of this component is defined in the following listed functional requirements. Additional to the specification, the defined functional requirements will serve the purpose of evaluating the prototype's capabilities thoroughly after its implementation and demonstration.

FR-1 The prototype generates provenance data for following actions of a CBTs:

- create Task
- vote Task (accept/reject)
- log Action
- verify Action

FR-2 The prototype stores generated provenance data for retrieval and verification.

FR-3 The prototype stores a checksum of the generated provenance information to the DLT as verification anchor point.

FR-4 The prototype provides access to the generated provenance data with a defined Application Programming Interface (API).

FR-5 The prototype integrates access to an implementation of a DLT and smart contract functionalities, such as the deployment of a smart contract.

FR-6 The prototype allows subscription and processing of triggered events by a smart contract.

FR-7 The prototype must offer an interface for the integration of the component.

5.3.2 Non-Functional Requirements

Non-functional requirements determine the qualities and attributes of the developed system. These constrain the development process and the system to be developed and set external constraints that the system must fulfill. The following subsections define and describe the restrictions for the accountability component as non-functional requirements, as defined by Sameer Paradkar [72].

The accountability component has to be compatible with the SmartSociety programming framework. Hence all the non-functional requirements have to be considered and viewed in the context of the SmartSociety platform.

NFR-1 Accountability

The most essential non-functional requirement of this component is to achieve accountability as per the definition in 3.2.

NFR-3 Security

The accountability component must ensure that only authorized and authenticated users are performing functions within the system. Only authorized users must have access to provenance data. Data communication between the user or external systems and the component itself has to be encrypted.

NFR-4 Extensibility and Maintainability

The requirement of extensibility and maintainability can be achieved by adhering to coding and design standards, best practices, and reference architectures. The system must be composed of loosely coupled components to reduce the impact of changes and upgrades on the overall system. In addition, the documentation of the system must be complete in order to understand the general functionality and possible dependencies on other parts of the system. Continuous integration and delivery also play a crucial role in terms of maintainability and extensibility, as these approaches provide the ability to quickly integrate changes and provide new artifacts that can be deployed at any time through manual release.

NFR-5 Interoperability

The main and crucial aspect of the accountability component is that it is required to be compatible to work with the SmartSociety platform. The runtime environment and dependencies of the implementation of SmartSociety programming framework have to be taken into account for an interoperable accountability component. Provenance information should be stored in the standardized and accessible PROV-N and Resource Description Framework (RDF) file format for full interoperable support in the domain of provenance and the semantic web. Communication from and to external parts of the system should be done in the Javascript Object Notation (JSON) format. Complete documentation of the API should be available. The query of provenance data will be available through a Representational State Transfer (REST) API.

NFR-6 Performance and Scalability

A request should be handled within a reasonable amount of time. The additional computational effort of generating and storing provenance information, as well as the access to DLT should not limit the performance and scalability. The chronological order of the users' actions is a crucial aspect of having reliable provenance information and must not be sacrificed.

The system must handle the same amount of user requests as for the SmartSociety platform.

NFR-6 Usability

Users who interact directly with SmartSociety, DLT, and indirectly with the specified accountability component should be able to use the system easily, and technical details should be kept transparent to the user. Data provided via a REST API by the component must be well documented to facilitate its use. The implementation of a user interface for interaction with the accountability component must be intuitive in its layout and operation to the end-user and in line with the look and feel standards.

5.3.3 Use Cases

Use cases capture the functionalities of a system by analyzing the system requirements.

Figure 5.4 depicts an overview, in Unified Modeling Language (UML), of the involved actors and their interaction in the different parts of the system. In the following subsections, each use case is briefly described based on Alistair Cockburn's concept of short use case documentation [73]. The descriptions are kept short to support an agile and lean process.

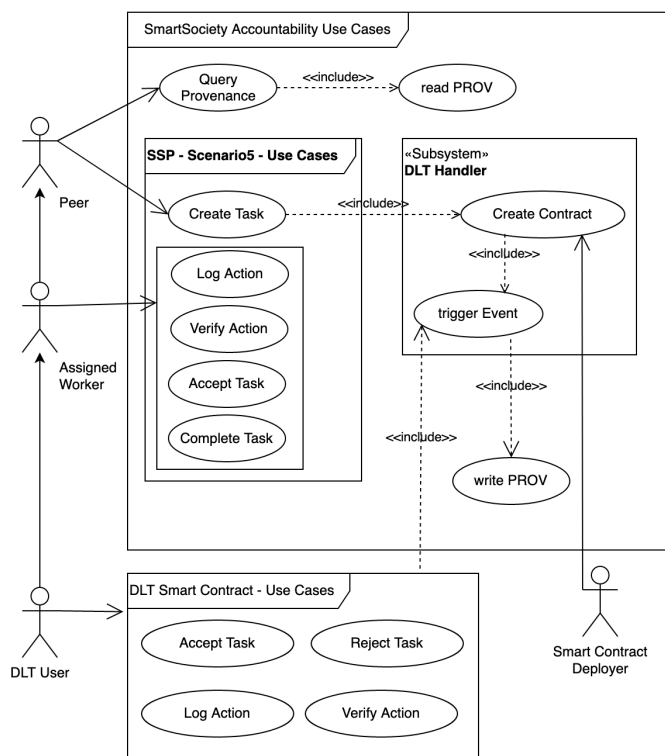


Figure 5.4: UML use case diagram overview of the system

Actors

An actor models a specific role of an external entity in the domain of the represented system [74]. The specification defines the actors interacting with the system and their associated use cases. The external entities can be represented by physical entities such as human users and hardware, but also organizations, internal or external applications.

Peer In this application domain a peer represents a human user that has a valid peer profile configured in the SmartSociety platform.

Assigned Worker An assigned worker has the same properties as a *Peer* and additional is part of the collective assigned for the collective based task.

DLT User The DLT user has an asymmetric key pair that serves as user identification and for interaction with the smart contract on the DLT.

Smart Contract Deployer A wallet address used to deploy smart contracts on the DLT as part of the SmartSociety application.

Peer Creates Task «SmartSociety»

Description: The user creates a collective based task at the application that triggers a linked smart contract deployment at the dlt.

Precondition: The peer is logged in.

Postcondition: The peer sees his created task as confirmation.

Main Success Scenario:

1. A user/peer enters details about the task, such as title and description
2. The peer clicks on create Collective Based Task with its details
3. The peer receives feedback about the creation of the task

Assigned Worker Accepts Task «SmartSociety»

Description: An assigned worker for a task accepts the given task

Precondition: The assigned worker is logged in, and assigned tasks are displayed

Postcondition: Assigned worker accepted the task at SmartSociety application as well at the Smart Contract

Main Success Scenario:

1. An assigned worker is logged in and selects the task to accept
2. The worker clicks on accept task which triggers accept task for smart contract as well
3. The worker receives feedback about the acceptance on the task

Assigned Worker Rejects Task «SmartSociety»

Similar to the use case 5.3.3, only with the command *reject*.

Assigned Worker Logs Action «SmartSociety»

Description: An assigned worker for a task logs an action

Precondition: The assigned worker is logged in and all workers accepted the task

Postcondition: Assigned worker logged/submitted an action for the task at SmartSociety application as well at the Smart Contract

Main Success Scenario:

1. An assigned worker is logged in and selects the task
2. The worker enters an activity and clicks submit log action which triggers log action for smart contract as well
3. The worker receives feedback about the logged action on the task

Assigned Worker Verifies Action «SmartSociety»

Similar to the use case 5.3.3, only with the command *verify*.

Deployer Creates Contract «dlt-handler»

Description: An assigned worker for a task accepts the given task

Trigger: Use case *Peer Creates Task*

Precondition: A Collective Based Task has been created in Smart-Society Application Context

Postcondition: A corresponding Smart Contract has been deployed and the referenced deploy-address and other useful information is returned

Main Success Scenario:

1. The actor *Smart Contract Deployer* deploys a Smart Contract with specified details (from Create Task)
2. SmartSociety is subscribes to all events for the deployed smart contract
3. Before and after deployment PROV data is generated and stored

DLT User Accepts Task «DLT»

Description: An assigned worker for a task accepts the given task

Trigger: Use case *Peer Creates Task*

Precondition: A Smart Contract is deployed

Postcondition: Smart Contract is updated with action (accept)

Main Success Scenario:

1. The Assigned User clicked on Accept Task
2. *DLT user* receives confirmation window to sign a transaction to call accept function of the Smart Contract
3. *DLT user* confirms transaction for accept function call.
4. The transaction is confirmed on the DLT

DLT User Rejects Task «DLT»

Similar to the use case 5.3.3, only with the command *reject*.

DLT User Logs Action «DLT»

Description: An assigned worker for a task logs an action

Trigger: Use case *Peer Logs Action*

Precondition: A Smart Contract is deployed

Postcondition: Smart Contract is updated with logged action

Main Success Scenario:

1. The Assigned User entered an action and clicked on log/submit
2. *DLT user* receives confirmation window to sign a transaction for the smart contract
3. *DLT user* confirms transaction for log action function call.
4. The transaction is confirmed on the DLT

DLT User Verifies Action «DLT»

Similar to the use case 5.3.3, only with the command *verify*.

Handle Events «*dlt-handler*»

Description: When a smart contract is created and deployed, Smart Society is subscribed to events fired by changes on the DLT made by the contract

Trigger: Use cases *Create Contract*, *Accept Task*, *Reject Task*, *Submit Action*, *Verify Action*

Precondition: A Smart Contract is deployed, or a function from the smart contract is called that includes an event

Postcondition: PROV data is generated and stored

Main Success Scenario:

1. Application Event Handler is called when an event is fired (see Trigger).
2. It creates PROV data and stores it

Write PROV

Description This generates provenance data in a standardized format and stores it

Trigger Use case *Handle Event*

Precondition An event is being triggered and handled

Postcondition The event is processed, and PROV data is generated and stored

Main Success Scenario

1. An event is triggered and processed
2. Provenance data is generated depending on the event and details
3. Provenance data is stored for retrieval later on

Read PROV

Description Reads Provenance data for a specific query

Trigger Use case *Query Provenance*

Precondition A query for provenance was called

Postcondition Returns retrieved provenance data

Main Success Scenario

1. A query for provenance has been called
2. Lookup and read provenance records depending on query
3. Return provenance records

Peer Queries Provenance

Description A peer queries (via REST API) provenance data and gets provenance records depending on the query as a result

Precondition Endpoint for query provenance is valid and reachable for peer

Postcondition Provenance data is returned

Main Success Scenario

1. Peer calls query provenance endpoint with specified query
2. Provenance records as result to the query are returned

5.4 Architecture

In the following section an overview of the architecture in context to the SmartSociety and a more detailed view of the accountability component and its architecture are provided.

5.4.1 SmartSociety Application

Figure 5.5 depicts an overview of the accountability component and how it extends SmartSociety. Green boxes illustrate components that are added by this project. The SmartCom Adapter [70] is responsible for the communication between the web client and the SmartSociety application. For that it is required to implement an Input and Output Adapter as defined in the thesis *Virtualizing communication for hybrid and diversity-aware collective adaptive systems* [70], which describes the SmartCom communication middleware.

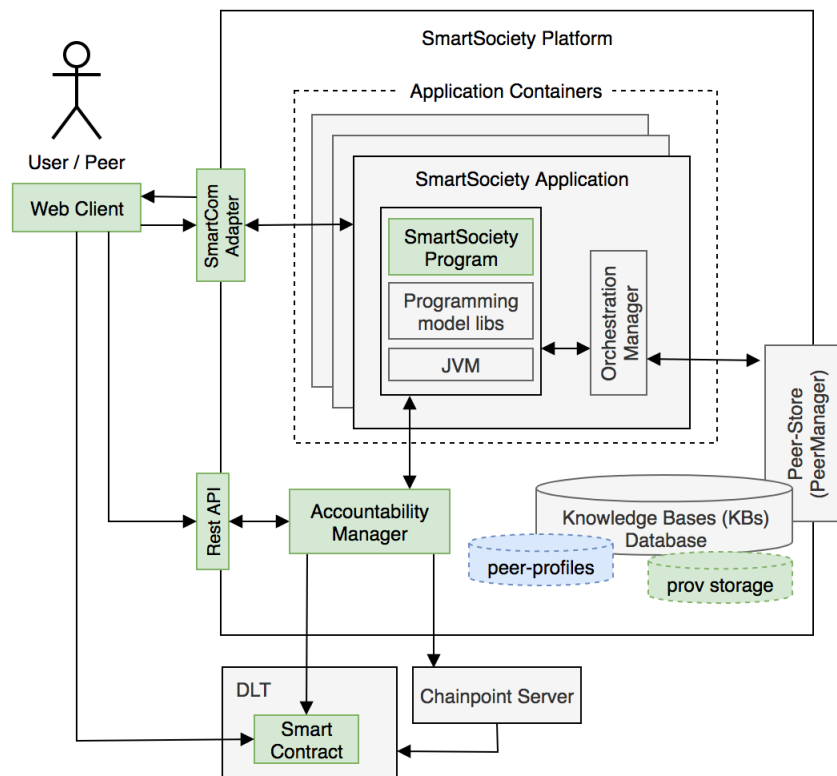


Figure 5.5: Overview of the Accountability Module in context of SmartSociety

The *REST API* is an additional interface providing query functionality for provenance records. A smart contract template will reside in the SmartSociety Platform, and it will be deployed as a smart contract on the DLT also executed there. The SmartSociety Program is an application of the SmartSociety and will serve as a demo for the usage of the

accountability manager. The blue database symbol illustrates the existing *peer-profiles* that are extended for the given use cases 5.3.3.

5.4.2 Accountability Component

The *Accountability Manager* or Manager is displayed in Figure 5.6. The colors represent the different application scopes of SmartSociety Application as a blue box, the Accountability component with orange boxes, and the WebClient with the green box. The WebClient will be able to connect via Input/OutputAdapter with SmartSociety Application, via REST API with the Accountability Module itself and via the DLT if required directly since it can be queried from outside. The smart contract is accessed via the WebClient directly from the peer itself, only the deployment of the smart contract is done by the DLT Manager/Client from the Accountability Module.

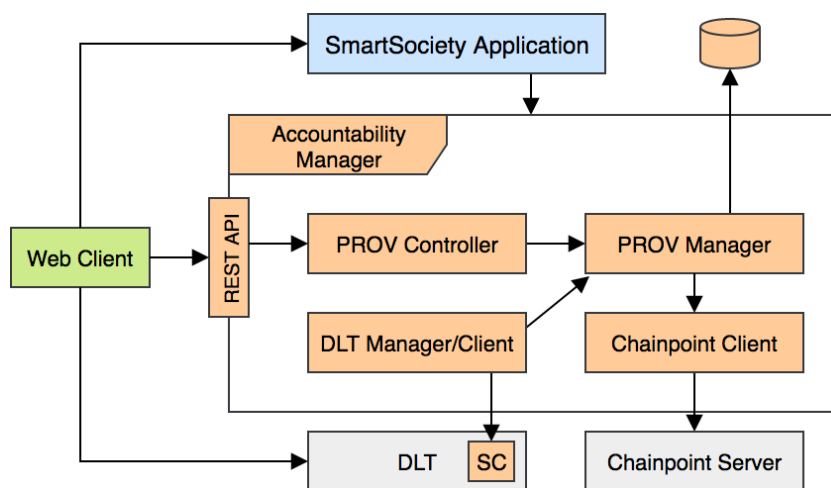


Figure 5.6: Overview of the Accountability Component

Web Client

The *Web Client* will be connected to the SmartSociety application by a SmartCom Input and Output Adapter implementation. It can be based on WebSockets, Hypertext Transfer Protocol (HTTP) long-polling, or any other communication method. The advantage for a WebSocket is that one channel is established and upgraded from HTTP protocol to WebSocket protocol that serves for bidirectional communication between client and server. This connection will be utilized for login to SmartSociety application as well as for retrieving updates for tasks and other useful user relevant information from the application.

Accountability Manager also provides a REST interface within the SmartSociety platform to access provenance data using query capabilities. The web client can access defined end points of this REST interface for querying this data.

The third available connection from the web client is the DLT itself, either through a bridge application such as MetaMask or by running an always connected and synchronized full network node of the DLT. MetaMask is a third-party browser extension that connects to a centralized service that provides the connection to the defined network of a DLT. It keeps the web client lightweight and the user from installing and maintaining a full node.

PROV Controller

The PROV Controller will handle all queries for provenance data from the REST API. It retrieves data depending on the query from the PROV Manager and returns it to the web client in a requested format. To maintain the standard, it should be possible to retrieve data in a standardized format of PROV, such as PROV-N, RDF or similar formats.

PROV Manager

The PROV Manager will manage all provenance generation and storage of the records as well as access and retrieval. The main focus will be to capture every action triggered by an event from the DLT via the DLT Manager/Client. Furthermore, the PROV Manager creates a fingerprint (hash) of the generated PROV document and calls the Chainpoint Client to anchor this hash in a further step for proof-of-existence of this document. The document itself can be stored either within the scope of SmartSociety or also in a distributed storage infrastructure, such as InterPlanetary File System (IPFS).

Chainpoint Client

The Chainpoint Client is responsible for anchoring digital fingerprints of provenance documents, as proof of existence and verification, to the DLT. The following hash of the document's content is submitted to multiple Chainpoint servers to guarantee redundancy. Chainpoint stores each submitted hash in a Merkle tree data structure for efficient storage and anchors only the Merkle tree root hash to the DLT. The Chainpoint proof has to be retrieved within 24 hours and stored locally to be able to verify the document's hash afterward.

DLT Manager Client

This component is responsible for managing every relevant processing that includes DLT. The requirements for this component are to enable communication with smart contracts as well as their deployment to the DLT. Furthermore, subscriptions to defined events on the DLT triggered by the smart contracts need to be maintainable within this component.

One instance of such an implementation is Web3J, a library to integrate applications with Ethereum. The library, as mentioned earlier Web3J, includes functionality to interact with smart contracts using Java.

SmartSociety Application

This application is a demo implementation using the accountability component to demonstrate in a generic scenario how it can be used and its capabilities. It is implemented as an addition to the SmartSociety programming framework.

PROV Storage

The PROV Storage handles all provenance records in multiple possible formats such as RDF, JSON, and PROV-N. In addition, it needs to store also all relevant Chainpoint proofs 5.4.2 with its anchor information of the respective DLT.

5.5 PROV-DM Mapping

This section describes an overview of the defined mapping for the PROV-DM.

5.5.1 Namespaces

Provenance can be captured in different levels of details and perspectives, as described in 3.4.2. The PROV Model is focused on the collective based task represented by a smart contract and its induced activities performed by the DLT users. The verification of provenance records can be easily achieved by retrieval of the transaction history of the DLT. DLT users will sign with their identified address the transaction to invoke a function, such as an activity, at the corresponding smart contract.

The PROV Model is based upon the aforementioned use case diagram at section 5.3.3. An overview of the PROV-DM mappings can be found in the following table 5.11.




Class	Qualified Name (Id)	prov:type	prov:label
	smart: <i>wallet-address</i>	prov:Person	DLT User
	smart: <i>wallet-address</i>	prov:SoftwareAgent	Smart Contract Deployer
	smart: <i>unique-id</i>	smart:Cbt	Collective Based Task
	smart: <i>unique-id</i>	smart:CbtItem	Collective Based Task Item
	smart: <i>wallet-address</i>	smart:Contract	Smart Contract
	smart: <i>unique-id</i>	-	Deployment (Create Task)
	smart: <i>unique-id</i>	-	Accept Task
	smart: <i>unique-id</i>	-	Reject Task
	smart: <i>unique-id</i>	-	Submit Task
	smart: <i>unique-id</i>	-	Verify Task

Table 5.11: PROV-DM Mapping

5.5.2 Namespace

Following namespaces will be used with their defined prefixes.

Prefix	Namespace IRI	Description
prov	http://www.w3.org/ns/prov#	The PROV namespace
smart	http://www.smart-society-project.eu/	Smart Society Platform namespace

Table 5.12: Defined Namespaces

5.5.3 Agent

A DLT user and the Smart Contract Deployer are mapped to the element of a PROV agent. The DLT user is also a registered peer in the smart society platform and has a unique wallet address as a user identifier. The Smart Contract Deployer as an actor is a running software within the smart society application and responsible for the creation of the smart contract representing the CBT.

5.5.4 Entity

In this domain, an entity will be a CBT, including specifically assigned work items and its corresponding smart contract. Finding an appropriate workflow and assignment of the single task items for a collective is out of scope for this thesis and will be presumed as given.

5.5.5 Activity

The activity element is represented by the commonly available actions for the smart contract, such as accept, reject Task and submit, verify an action. Different activity elements will be used to be able to have a complete provenance record.

5.5.6 Interrelation

Interrelations between entities and activities are well defined in PROV. These associations for the differently defined activities in this domain can include *wasGeneratedBy*, *used*, *wasStartedBy*, *wasEndedBy*, *wasAttributedTo*, and *wasAssociatedWith*.

5.5.7 Attributes

Attributes describe the PROV elements in different aspects. In the following description list different attributes and their usage in this model are described.

prov:label Represents a human-readable representation for elements of PROV. One example of such a label attribute is 'Collective Based Task' for an entity representing a CBT.

prov:location The location attribute can be used to describe geographic (ISO-19112) or non-geographic places such as the system where the smart contract is deployed and is located.

prov:role A role is a function of an agent or entity with respect to an activity. An agent can have a specific role for its responsibility. In this setup, the role 'Deployer' is used to describe the role of the deployer.

prov:type The type attribute provides additional typing information of a PROV construct.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Implementation

This chapter describes the implementation details of the working prototype ¹ for the accountability management component. The technologies used for the implementation are discussed, and integral parts of the implementation are selected for a detailed description.

6.1 Technology Stack

The prototype for the accountability component is written in Java. It maximizes compatibility as other components, and the overall framework implementation of SmartSociety programming framework is also written in Java. As the dependency injection framework the lightweight solution is being used. It is an implementation of the JSR-330 and can be used for standalone applications in JavaSE environments.

An essential part of the accountability component requires the creation, management and export of provenance information based on the PROV standard. This PROV component should support different PROV encodings for achieving a high interoperable system. The World Wide Web Consortium (W3C) provides a list² of different implementations utilizing the PROV standard and the support of different types of PROV encodings. These implementations include applications, services, and complete open-source frameworks. The open-source framework ProvToolbox³ is also listed on the website mentioned above provided by the W3C. It is a Java-based framework and supports all PROV representations as well as other standardized formats such as Resource Description Framework (RDF), Extensible Markup Language (XML) and Javascript Object Notation (JSON).

Apache Jena Fuseki is an open-source SPARQL server to serve RDF data over Hypertext Transfer Protocol (HTTP). Fuseki is a subproject of the open-source semantic web

¹<https://bitbucket.org/alexanderp-tu/smartacc/src/master/>

²<https://www.w3.org/TR/prov-implementations/>

³<https://github.com/lucmoreau/ProvToolbox>

framework Apache Jena and developed as a servlet running in a preconfigured Jetty web server as the container. The provenance information can be imported as RDF encoded files, and a graph is built out of this data that can be stored and queried with SPARQL, a RDF query language.

Ethereum was used as an widely adopted and supported implementation of a Distributed Ledger Technology (DLT) with smart contract capabilities. The Java implementation of the Ethereum protocol EthereumJ⁴ was used that allows interaction with the DLT Ethereum.

Chainpoint^{5,6} is an open standard for anchoring data to DLTs such as Bitcoin and Ethereum. Since there is no Java implementation, a custom implementation has been made of the chainpoint client to be able to connect and submit document hashes to the Chainpoint services. The document hash of provenance recorded files that are submitted to the Chainpoint network for anchoring the hash eventually to the DLT as proof-of-existence.

Apache Maven is used for the build process and the dependency management. It includes unit tests and was configured to create a deployable fat-jar file with all required dependencies included. It enables a fast and efficient approach to include and make use of the component.

The table 6.1 lists all relevant technologies that have been used with this implementation.

Name	Version	License	Note
Java	1.8	GPL	
HK2	2.6.0	CDDL, GPLv2	Lightweight DI-Framework
PROV-Toolbox	0.9.0	MIT License	Library for W3C PROV data
Web3J	4.5.5	Apache-2.0 License	Library for Ethereum integration
MongoDB	3.2.2	Apache-2.0 License	Library to interact with MongoDB instance
Flapdoodle	2.2.0	Apache-2.0 License	Embedded MongoDB instance
SLF4J	1.7.7	MIT License	Simple Logging Facade
JUnit	4.11	CPL	Testing Framework
Mockito	2.23.4	MIT License	Mocking Framework for Unit Tests
Jena Fuseki	3.13.1	Apache-2.0 License	SPARQL Server serving RDF data via HTTP
Maven	3.5.3	Apache-2.0 License	Build Automation Tool
Solidity	0.5.12	GPLv3	OOP Language to create smart contracts
Ganache	2.1.1	GPLv3	Personal (in-memory) Ethereum blockchain
Go Ethereum	1.9.6	LGPLv3	Implementation of Ethereum protocol in Go

Table 6.1: Technology Stack of SmartAcc

⁴<https://github.com/ethereum/ethereumj>

⁵<https://chainpoint.org/>

⁶<https://github.com/chainpoint>

6.2 Package Structure

The component's packages are structured around features to provide a clear grouping of related parts of the system. Following the structure is briefly summarized.

at.ac.tuwien.dsg.smartacc This is the root of all packages. It contains the `Main.java` class and the `ProvenanceApplication.java` which can be used as the entry point for any external usage. It will inject all necessary services and provides the basic functionality of the accountability component.

at.ac.tuwien.dsg.smartacc.chainpoint It contains all relevant source code for handling the anchoring of document hashes by Chainpoint. A client is implemented, which handles communication with Chainpoint network. It allows to submit hashes and retrieve proofs.

at.ac.tuwien.dsg.smartacc.db All database relevant implementation parts are included here. It uses MongoDB as a database.

at.ac.tuwien.dsg.smartacc.eth This package includes a web3j service to connect the component to an Ethereum blockchain. It can deploy smart contracts and also execute also transactions on the blockchain.

at.ac.tuwien.dsg.smartacc.prov The prov packages contain the provenance service to create, read, and write the defined PROV-DM. Furthermore, it includes the Apache Jena Fuseki server for retrieval of provenance information.

6.3 Sequence Diagram

Figure 6.1 illustrates the sequence diagram for creating and accepting a task through the SmartAcc application.

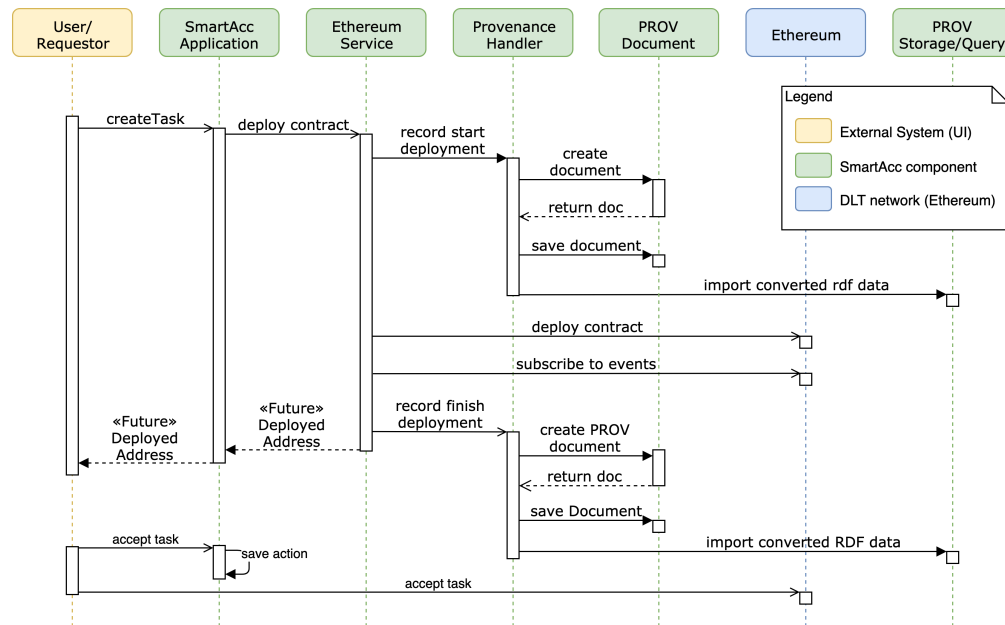


Figure 6.1: SmartAcc - Sequence diagram

6.4 SmartAcc - Accountability Component

The following subsections contain descriptions of the main implementation details of the developed accountability component.

6.4.1 Bootstrap the Application

Bootstrapping the SmartAcc component can be achieved by extending the abstract class *SmartAccBootstrap*, which defines application startup and shutdown functionality of the component. It includes optional configuration for very fine-grained logging possibility as well as setting a client for the MongoDB connection. The fine-grained logging will impact the performance and must be used with caution. The MongoDB client enables the possibility to use an external existing mongo client with this component.

6.4.2 Task

The Task data transfer object includes all required information for the deployment and provenance of the corresponding CBT originating from SmartSociety.

6.4.3 SmartAccApplication

This is the main application of the accountability component. It extends the abstract application bootstrap class 6.4.1 and implements the defined functionality. The implemented functions call the Ethereum service which process transactions to the DLT Ethereum. The provenance and anchoring functionality is called from the DLT module directly.

6.4.4 DLT Module

The DLT module resides in the package *at.ac.tuwien.dsg.smartacc.eth* and includes a *EthereumService*, which is configured by the *EthereumServiceConfig* class and its corresponding values retrieved from the global file *application.properties*. The *EthereumService* includes functionality to deploy a contract with a default Ethereum address or a provided Ethereum wallet file, subscribe to events triggered by the smart contract on the Ethereum blockchain. The injected *provenanceService* handles the provenance generation. Due to the nature of DLTs in general and Ethereum, especially with a 15-second block interval, the function for the contract deployment returns the result as a *CompletableFuture<String>* that is set to the deployed address of the contract once the deployment transaction has been processed.

Besides the deployment itself, there is also the provenance recording functionality which is triggered by specific smart contract events at the DLT. These provenance functions are responsible for creating and updating PROV documents as well as saving them for later retrieval. Furthermore, in the next step, the document's hash is submitted to the Chainpoint servers for anchoring of the hash as a proof of existence.

By subscribing to defined events triggered by the deployed smart contract, it is possible to react to specified events and utilize the accountability component as the responsible provenance record keeping. The events are automatically mapped from the smart contract's language to the Java class representation by the Web3J library.

To interact with the smart contract without leaving the JVM it is possible to generate Java wrappers for smart contracts written in Solidity. Web3J enables auto generation of the wrapped classes by following two commands displayed in listing 6.1 and 6.2.

```
1 $ solc <contract>.sol --bin --abi --optimize -o <output-dir>/
```

Listing 6.1: Compilation of the smart contract File (.sol).

```
1 $ web3j solidity generate -b /path/to/<smart-contract>.bin -a /path/to/<smart-contract>.abi -o /path/to/src/main/java -p com.your.organisation.name
```

Listing 6.2: Command to generate wrapper code for the compiled Solidity file.

6.4.5 Provenance Module

The provenance module uses the `ProvToolbox`⁷ library to create and update W3C PROV documents. Once the document has been created or updated with provenance information its meta-data is saved to the MongoDB and the converted RDF file is saved to the Fuseki SPARQL server for query access. Furthermore the document hash is created and submitted to the Chainpoint servers for anchoring to the DLT as proof of existence.

ProvenanceService

The *ProvenanceService* handles all the provenance related tasks as described at the above subsection 6.4.5. First, a provenance document in PROV-N representation is created with the provided details of the `taskDTO` object. As a next step, it is saved locally for further processing. The created document is then written to the stream in Turtle format, a representation for RDF data models. Finally, the converted Turtle format is imported to the Apache Fuseki SPARQL server for later data query and retrieval.

6.4.6 Anchoring Module

Chainpoint is a service for anchoring document hashes to Bitcoin and Ethereum blockchain as proof of existence. The integrity and state of a file can be verified by this mechanism. The implementation allows to submit document hashes to the Chainpoint servers for processing and anchoring document hashes to the blockchain eventually. A minimized proof can be retrieved almost immediately when the chainpoint nodes processed the hash submission request. The full proof can be retrieved and saved for later verification. The complete proof is only available for one day, after this time it is only possible to verify the complete proof by the Chainpoint service.

6.4.7 Smart Contract - Solidity

The implementation of the smart contract was accomplished with Solidity, a scripting language for the DLT Ethereum. The smart contract represents the CBT with respect to the SmartSociety. For every `create Task` function call a smart contract is being created by calling the constructor with designated values of the CBT. The creator of the smart contract is also the owner, and it is possible to set access restrictions for function calls of the contract by defining modifiers. Furthermore, events can be defined and triggered to utilize the EVM logging facilities that store the included arguments in the transaction log.

Listing 6.3 depicts some of the defined events at the implemented smart contract. The concept of the transaction log data structure and events, in general, is described in more detail at the background chapter 3.6.1 A smart contract is written in the turing-complete language Solidity and with Web3J it is possible to generate Java based wrapper objects for direct interaction with the contract from within the JVM.

⁷<http://lucmoreau.github.io/ProvToolbox/>

```

1 event LogWorkerAdded(address _worker);
2 event LogTaskAccepted(bytes16 _taskId, address _voter);
3 event LogTaskRejected(bytes16 _taskId, address _voter);
4 event LogActionAdded(bytes16 _taskId, bytes16 _actionId, address _addedBy,
    uint _actionCount);
5 event LogActionVerified(bytes16 _taskId, bytes16 _actionId, address
    _verifiedBy, uint _verifierCount);
6 event LogContractCreated(bytes16 _taskId, address _requester, address
    _deployer, uint _registeredWorkerCount);

```

Listing 6.3: Defined Events of the Task contract.

Modifiers

Modifiers can be used to add access restrictions to functions of the smart contract. In listing 6.4 a modifier of the smart contract `TaskContract.sol` is depicted.

```

1 modifier onlyRegisteredWorker {
2 require(registeredWorkers[msg.sender].isRegistered, "msg.sender is not
    included at registeredWorker array.");
3
4 }

```

Listing 6.4: A defined modifier of the Task contract.

6.4.8 Database Module

The database module handles all the database relevant functionality. At the moment a MongoDB service is used that can be either instantiated with its own database client or it can be constructed with reference to an external database instance connected with the provided client. DAOs are responsible for saving and querying data from the database and can be injected wherever needed in `smartAcc`.

6.4.9 PROV Query API

The embeddable Apache Jena Fuseki Sparql server implements the provenance query interface. The `importData` function reads the stream data and creates an RDF model, that can be loaded into the Fuseki server. Clients can access a provided web interface of Fuseki to query the data or also manage the server in general. Furthermore, SPARQL over HTTP is supported to read, create, and update RDF datasets of the server.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Evaluation

This chapter describes an implemented scenario (scenario-5) for the SmartSociety programming framework by demonstrating the functionality of the accountability component.

First, the general scenario is outlined, and an overview of the implementation details are presented. After that, the execution of the implemented scenario is described in detail. Finally, the requirements defined in design chapter 5.3.1 and 5.3.1 are evaluated and discussed as outlined in “Design Science in Information Systems Research” [3].

7.1 Setup

This section describes the setup of the implemented scenario ¹ used for demonstration purposes and evaluation.

7.1.1 SmartAccTask

The SmartSociety programming framework (see section 5.1.1) includes different scenarios to demonstrate the functionality of SmartSociety (see section 5.1). Including scenarios such as a hybrid collective question-answer service AskSmartSociety! and SmartShare a ride-sharing application.

To demonstrate and evaluate the functionality of the accountability component SmartAcc 6 a new scenario SmartAccTask application is created in the context of SmartSociety programming framework. SmartAccTask can be considered an on-demand generic task request application. A user can request a task, and the platform will find the most suitable peers/collectives that are optimally capable of executing the requested Collective Based Task (CBT). The accountability component transparently records all actions of the human peers involved in the task in PROV representation. The complete scenario consists of a

¹<https://bitbucket.org/alexanderp-tu/smartacctask/src/master/>

user interface to interact with the application, and the implemented SmartSociety task application scenario SmartAccTask that uses the SmartAcc accountability component. The implemented scenario was intentionally kept simple for demonstration purposes. An overview of the SmartSociety architecture and its application context is shown in Figure 5.5 in the chapter Design.

7.1.2 Technology Stack

Table 7.1 lists all relevant technologies that have been used for the implementation of the working scenario SmartAccTask. The frontend is based on responsive elements defined by Bootstrap and uses web3.js to interact with the deployed smart contract by the SmartSociety platform. MetaMask is a browser extension and enables the user to confirm transactions to interact with smart contracts in Ethereum from within the browser. In the development setup, Ganache is being used as an in-memory local Ethereum node. The web application is written with plain JavaScript and minimal dependencies required. Node.js and the installed package lite-server act as the local development web server and serve the web application. The scenario application SmartAccTask is written in Java as the underlying SmartSociety programming framework is also implemented in Java. Dependencies of the scenario are the SmartSociety programming framework, SmartCom as communication middleware for SmartSociety, SmartAcc as the included accountability component (see chapter 6), and java-websocket for establishing a websocket server at the backend acting as communication channel connected by the frontend the web application.

The scenario application was run on the following test machine setup: a MacBook Pro (Retina, 13-inch, Early 2015), Intel Core i5 2x2.7 GHz (5257U) with MacOS Catalina 10.15.2 (64-bit) and the 64-bit browser Chrome Version 79.0.3945.130.

Name	Version	License	Note
Node.js	12.12.0	MIT License	JavaScript runtime environment
npm	6.13.2	Artistic License 2.0	node package manager
lite-server	2.5.4	MIT License	Node.js web server to serve the application
jQuery	3.4.1	MIT License	required by Bootstrap
Bootstrap	4.3.1	MIT License	responsive user interface elements
web3.js	1.2.0	GPLv3	interaction with Ethereum
Solidity	0.5.12	GPLv3	solidity compiler (solc) for smart contract
truffle	5.1.3	MIT License	framework for smart contract development
MetaMask	7.7.9	MIT License	browser extension connecting to Ethereum network
Ganache	2.1.1	MIT License	local in-memory Ethereum network
Java	1.8		
SmartSociety programming framework	n/a	n/a	
SmartCom	1.0-SNAPSHOT	n/a	
SmartAcc	1.0-SNAPSHOT	n/a	
java-websocket	1.3.9	MIT License	

Table 7.1: Task Application Scenario Technology Stack

7.2 SmartAccTask Application

The SmartAccTask application extends the existing SmartSociety programming framework by an additional scenario implementation. Details of the SmartSociety application implementation are described in the next subsections.

7.2.1 Peers

The registered peers in the peer manager are loaded from the `Peers.json` file. One example of such a peer defined in JSON format is depicted in the listing 7.1. The `channelType` is specified as `WebSocket` since the peer is connected with a websocket from the web client to the backend application SmartAccTask. Another critical defined key-value pair is `ethereumIdentity`, which is the associated identity of this peer on the Ethereum network. It corresponds to the registered public address and is used for interactions on the Ethereum network.

```

1 {
2   "name": "peer1",
3   "channelType": "WebSocket",
4   "channel": "peer1@localhost",
5   "role": "HumanWorker",
6   "ethereumIdentity": "0x6C7155A4A8222D478E08de251f0933923DdE57Ce"
7 }

```

Listing 7.1: One peer definition from the `Peers.json` file.

7.2.2 Communication Adapters

The SmartCom communication middleware handles the communication between client and SmartSociety. An input and output adapter have to be defined for incoming and outgoing messages from and to the connected peers. The class `WebSocketInputAdapter` extends the `InputPushAdapter` class and implements additionally also the interface `OutputAdapter` which are both defined structures by the SmartCom component.

7.2.3 TaskRequest and Plan

Every request for a task by a user will trigger the creation of an `S5TaskRequest` object that extends `TaskRequest` defined by the SmartSociety. Additional to the `S5TaskRequest`, there are a `Task` object and a `TaskAction` object which are DTOs for the data exchange between the frontend and the backend. The `Task` includes all relevant details for the requested task, such for example an unique `taskId` as UUID, name, description, and the `deployedContractAddress` that specifies the address of the deployed smart contract `TaskContract` which represents the requested `Task`. S5 stands for scenario 5 since this implementation is the fifth scenario for the SmartSociety programming framework.

7.2.4 Task Lifecycle

The task lifecycle is processed when the task has been requested and the task runner of the SmartSociety has been started. For the description of each lifecycle state see also 5.1.1. In the following subsections the different implemented states in the order of processing are described.

Provisioning The provisioning process is implemented by implementing the *ProvisioningHandler* of the SmartSociety. This state is the starting point of the task's lifecycle and includes as inputs the *input collective*, the `taskRequest` and the *ApplicationContext*. In a normal situation, the matching of the suitable peers for the task is done here. For the demonstration, the entire input collective with all human peers is persisted here and returned.

Composition The interface *CompositionHandler* defines the implementation details for the next state, the *composition*. It includes as inputs the *ApplicationContext*, the *provisioned collective*, and the `taskRequest`. Since we are running the collaboration model of `on_demand` and `open_call` the task execution plan is implicitly assumed at this stage. The accountability component is called with the given `taskRequest` and its details.

Negotiation The *NegotiationHandler* is responsible for the negotiation process of the proceeding execution plan and its `taskRequest`. It also implements a *NotificationCallback* to handle negotiation relevant messages. Once the peers accepted or rejected the plan to execute the task, this will return the collective with the accepted plan. If rejected, it will end the task.

Execution The *ExecutionHandler* handles the execution process of the task and also requires the *NotificationCallback* to process messages relevant to the task execution. Peers can submit their action when done and verify action items from other peers. The more verifications have been registered for a single action item the higher the probability that the task has been completed successfully as per definition.

7.2.5 Web Client

The web client is written in JavaScript using graphical user interface elements defined by Bootstrap. The peer uses his registered public address in Ethereum as the identity to interact with the SmartAccTask application. On the startup of the web client, a websocket connection is trying to be setup to the websocket server at the SmartAccTask application. When connected successfully the Ethereum identity of the current logged in user is displayed at the top right of the web client application. For the connection, the user has to have MetaMask installed, which enacts as a bridge to the Ethereum network. MetaMask offers the option to easily switch between users.

7.3 Functional Evaluation

The functionality of the `smartacc` application as defined with functional requirements at section 5.3.1 is documented at the following subsection 7.3.1 - Execution. The setup and implementation of the scenario enables users to interact with the SmartSociety Task platform and by that use the `smartacc` module.

In this demonstration, there are two peers assigned to a task. The task is created by the first peer, which also acts as a worker for the task. As soon as the first peer is sending the task request, `smartacc` creates a corresponding smart contract for this task, which thereby can be called up by users directly without the third party of SmartSociety. Both peers accept the task, that is confirmed by the `smartacc` backend as well as by their own transaction made on the Ethereum blockchain for the created smart contract representing the task. As soon as both peers accepted the task it switches from the negotiation phase to the execution phase. One worker logs an action performed for the task, that again is processed by the SmartAcc application as well as the smart contract on the Ethereum blockchain. Each function call at the smart contract triggers an event, that is processed by the SmartAcc application for provenance record. Every interaction with the application is generating provenance information recorded in a document as well as in a document store for the possibility of retrieval of the provenance data through an API.

The module SmartAcc² was evaluated with the help of an additional implemented scenario, Scenario5³, in the SmartSociety programming framework. This is an independent working example for demonstration purposes. The execution of this additional implemented scenario is presented in the following subsections.

²<https://bitbucket.org/alexanderp-tu/smartacc/src/master/>

³<https://bitbucket.org/alexanderp-tu/smartacctask/src/master/>

7.3.1 Execution

Initial Page

When the web application is called for the first time, the user has to confirm at the MetaMask browser extension a connection request for the Ethereum user account by the local application Scenario5 (Figure 7.1a). Once this is confirmed, the user will be logged in at the application, showing by the logged-in Ethereum account at the top right corner as depicted in Figure 7.1b.

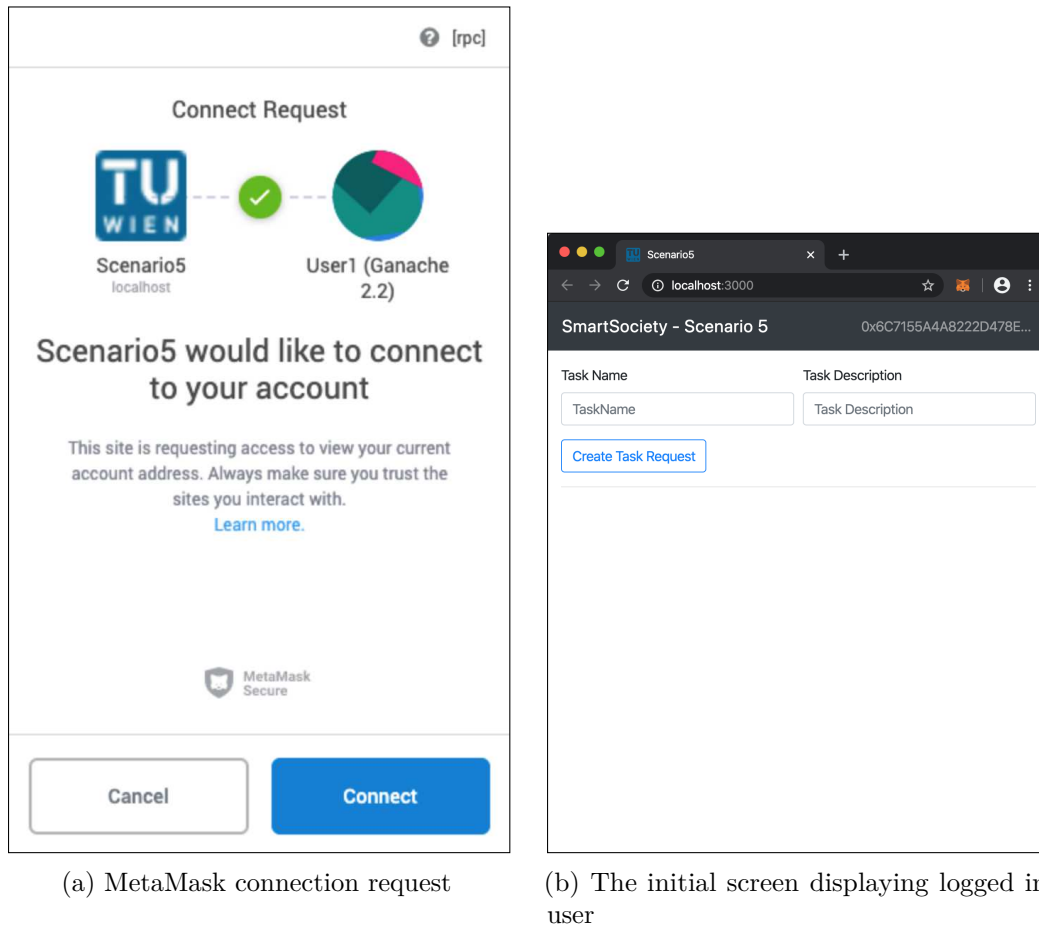


Figure 7.1: Initial Screens

Create Task

A user is creating a sample task. The created task is deployed as a smart contract by the SmartAcc application in the background. The deployed contract address and location is returned to the user interface and displayed after the task has been created and deployed. After this initial deployment, users can interact with the contract through the MetaMask extension as described in the following subsections. The state of the task lifecycle is after its creation at the 7.2.4 phase awaiting acceptance or rejection of the task. Figure 7.2 shows both screens the task creation step and the state with its details after the task has been created.

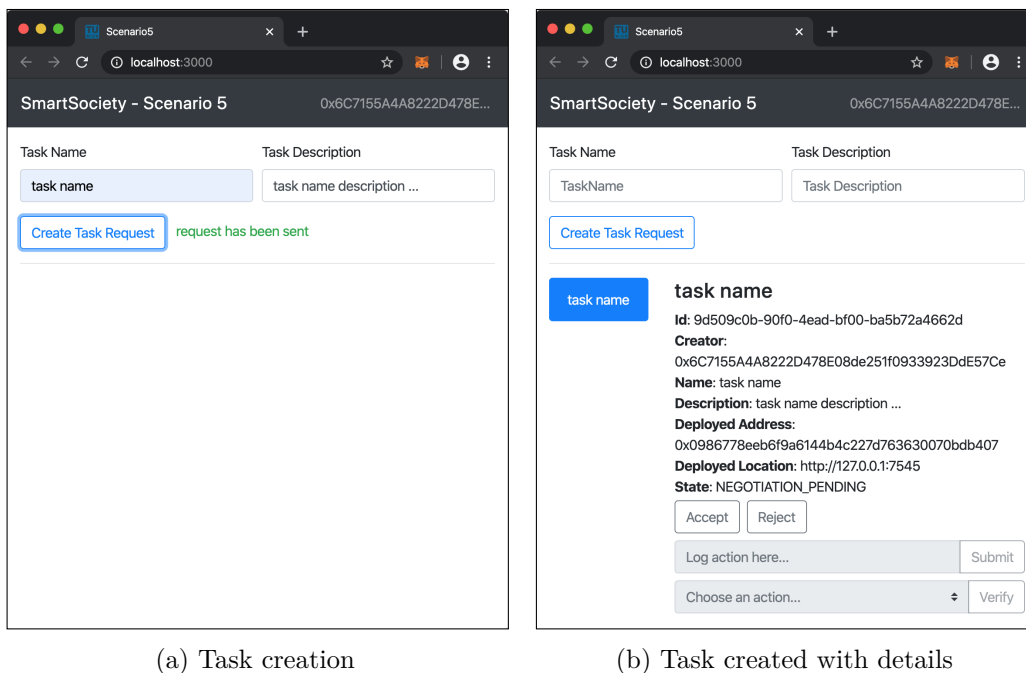


Figure 7.2: Task creation and its displayed details

Task Negotiation

In the negotiation phase of the collaborative task, both peers accept the task and additionally confirm their acceptance with a transaction that is sent to the respective smart contract for this task using the browser extension MetaMask. As illustrated in Figure 7.5, accepting the task at the smart contract is independent of the backend implementation and is only triggered and confirmed by the client, respectively, the working peer.

7. EVALUATION

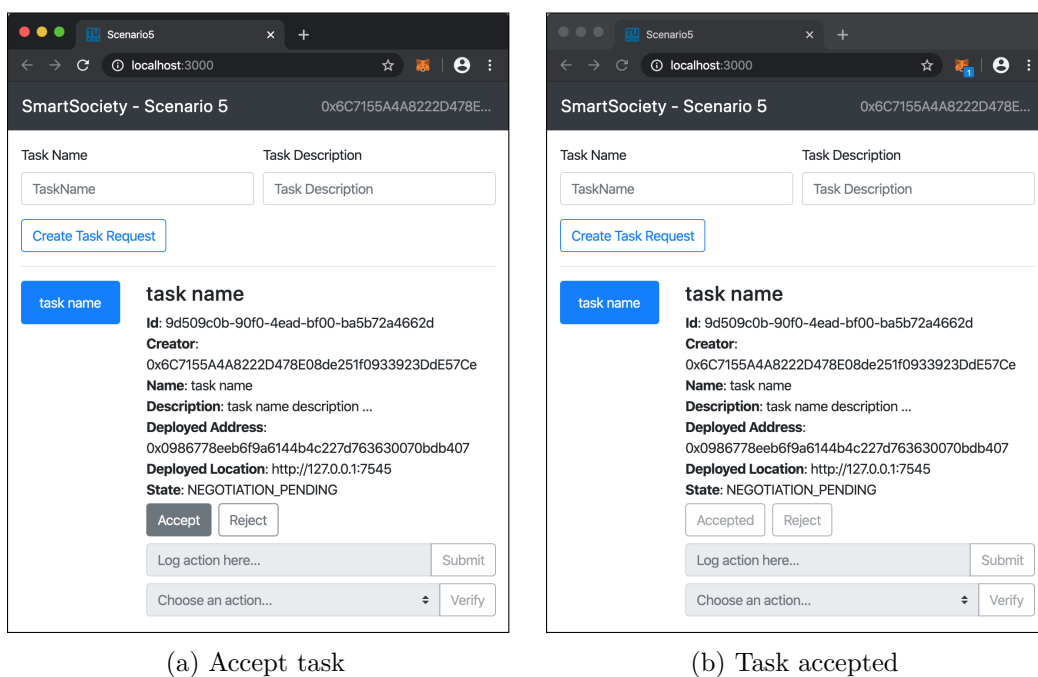


Figure 7.3: Task by User accepted

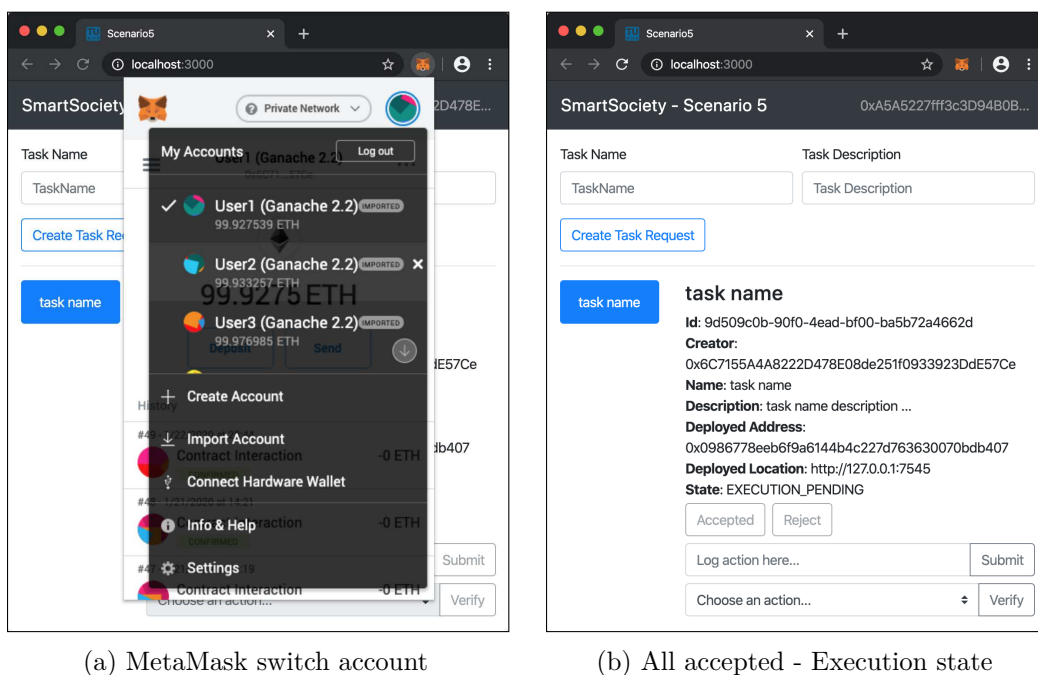


Figure 7.4: All users accepted and new Task state (Execution)

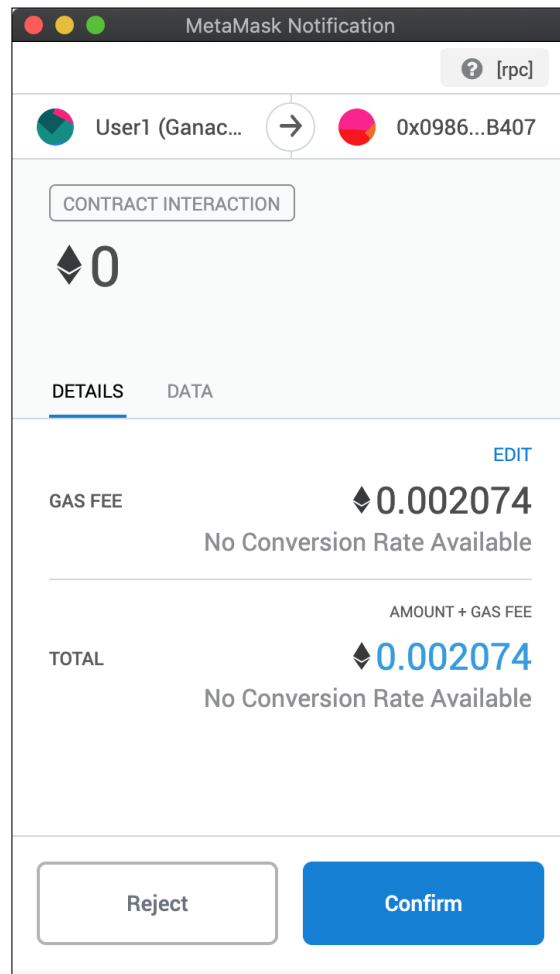
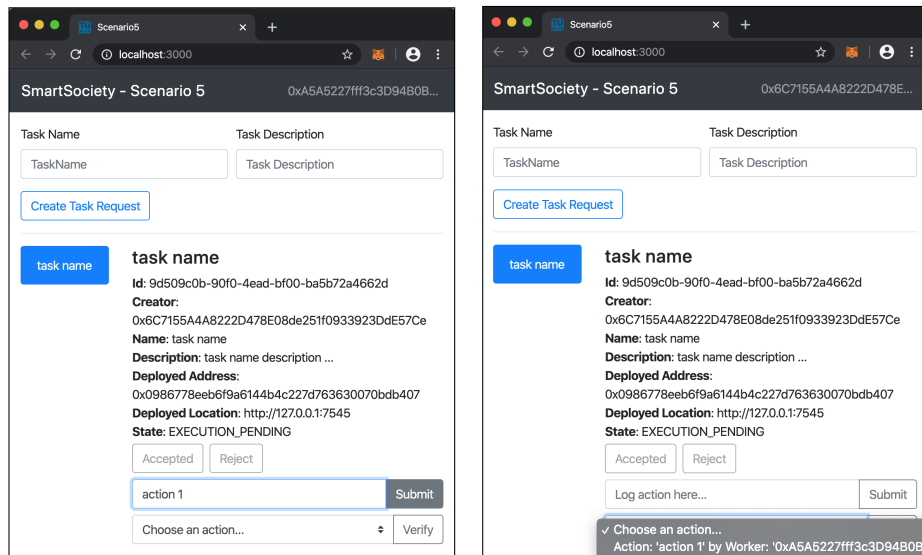


Figure 7.5: MetaMask - Transaction confirmation for accepting the Task

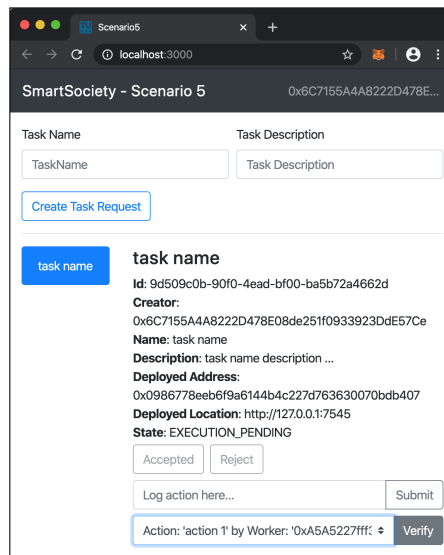
Task Execution

The next stage is the task execution. Peers can submit to the SmartSociety as well to the smart contract, their action with a briefly described textual representation. The submitted actions are displayed at the frontend and can be verified only by other peers that have not submitted this action. These steps are illustrated in the following Figure 7.6 with its subfigures.



(a) Accept Task

(b) Task accepted



(c) MetaMask switch account

Figure 7.6: Task accepted and new state

Task Completed

The task is either completed after a set time or when all other peers have verified the task actions submitted by others. The final state of the task at the user interface is displayed at Figure 7.7.

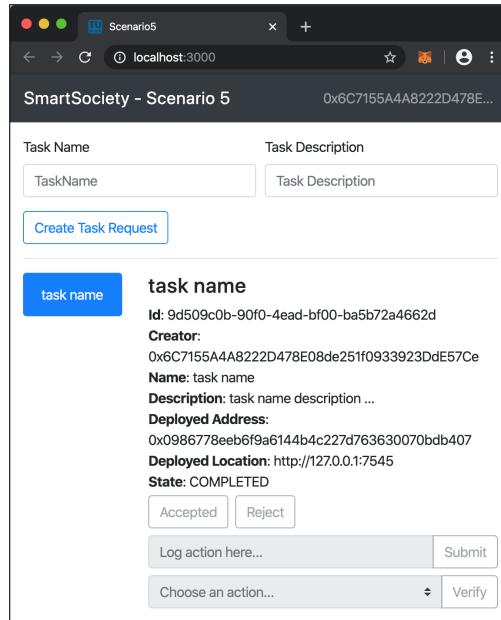


Figure 7.7: Task state completed

7.3.2 Generated Provenance Data

The provenance graph is created based on the PROV standard with each action taken by the user at the application. In the example, when the user accepts the task, both ways are processed and triggering a provenance log of the action, by the backend of the SmartSociety application itself and also by the user registering the smart contract event of their acceptance or rejection of the task at the contract.

PROVN Document

The generated document for the provenance information is in PROV-N format. An excerpt of the document is displayed at the listing 7.3. At the beginning of the document, the qualified namespaces are defined. It enables a unique identification system for different parts of the ontology. As an example the activity with the unique qualified identifier `dlttx:0x4f224a0f9125e60b0bf108431cf79ce92618ce327b...` describes an accept activity by an Ethereum user account. The qualified identifier resolves into `http://www.dlt-explorer.org/tx/0x4f224a0f9125e60b0bf10...` which would enable the user to lookup this transaction on the blockchain for an existing DLT explorer enabling any user to lookup this transaction. In this example and scenario, a DLT explorer is assumed under the specified URI for demonstration purposes.

Provenance Graph

The generated provenance document 7.3 can be used to create the corresponding provenance graph. The `provconvert` tool is capable of converting a PROV-N document into other formats, such as for example a PDF. The illustration of the provenance graphscenario-provn-file7.8 was generated by the `provconvert` command depicted in Listing 7.2.

```
1 provconvert -infile 9d509c0b-90f0-4ead-bf00-ba5b72a4662d.provn -outfile 9
   d509c0b-90f0-4ead-bf00-ba5b72a4662d.pdf
```

Listing 7.2: Comand to generate PROV graph from provn File

Provenance Query

The provenance information can also be queried by the client using SPARQL commands with the provided user interface of the included Apache Jena framework, which contains the corresponding provenance data as RDF.

```

1 document
2 prefix dlt <http://www.dlt-explorer.org/address/>
3 prefix smartaction <http://www.smartsocietyplatform.org/task/action/>
4 prefix smartpeer <http://www.smartsocietyplatform.org/peer/>
5 prefix smartcollective <http://www.smartsocietyplatform.org/collective/>
6 prefix dltx <http://www.dlt-explorer.org/tx/>
7 prefix smarttask <http://www.smartsocietyplatform.org/task/>
8 prefix smart <http://www.smartsocietyplatform.org/ns/#>
9 entity (smarttask:9d509c0b-90f0-4ead-bf00-ba5b72a4662d , [ prov:type = 'prov:
    Collection' , prov:type = 'prov:Plan' , prov:type = 'smart:Task' ])
10 entity (smartcollective:collectiveId , [ prov:type = 'smart:Collective' ])
11 entity (smartaction:ce0510f8-de86-4ecd-8d97-7ba916b91dd3 , [ prov:type = 'smart
    :Task-Action' ])
12 entity (dlt:0x0986778eeb6f9a6144b4c227d763630070bdb407 , [ prov:type = 'smart:
    Smart-Contract' , prov:location = "http://127.0.0.1:7545" %% xsd:anyURI
    ])
13 activity (dltx:0
    x4f224a0f9125e60b0bf108431cf79ce92618ce327b06634f30f97dc85df61897
    ,2020-01-22T20:44:39.000+01:00,2020-01-22T20:44:39.000+01:00 , [ prov:type
    = 'smart:Accept-Task' , prov:label = "acceptByDltEvent" ])
14 activity (dltx:0
    x61f300e4e5319d2b97b2243e0d3ceacb598ab416b162ea83c32d35ca5714297a
    ,2020-01-22T20:45:06.000+01:00,2020-01-22T20:45:06.000+01:00 , [ prov:type
    = 'smart:Accept-Task' , prov:label = "acceptByDltEvent" ])
15 ...
16 ...
17 agent (dlt:0x6C7155A4A8222D478E08de251f0933923DdE57Ce , [ prov:type = 'prov:
    Person' , prov:type = 'smart:Worker' , prov:type = 'smart:DLT-Identity' ])
18 used (smart:createTask , smartcollective:collectiveId , -)
19 used (smart:deployment , smarttask:9d509c0b-90f0-4ead-bf00-ba5b72a4662d
    ,2020-01-22T20:44:10.221+01:00)
20 ...
21 wasAttributedTo (smartcollective:collectiveId , smartpeer:0
    xA5A5227fff3c3D94B0B080b47aeC9fD94364689F , [ prov:type = 'smart:memberOf'
    ])
22 wasEndedBy (smart:deployment , dlt:0x0986778eeb6f9a6144b4c227d763630070bdb407
    , - ,2020-01-22T20:44:11.514+01:00)
23 wasEndedBy (smart:addAction-ce0510f8-de86-4ecd-8d97-7ba916b91dd3 , smartaction
    :ce0510f8-de86-4ecd-8d97-7ba916b91dd3 , - , -)
24 actedOnBehalfOf (dlt:0x6C7155A4A8222D478E08de251f0933923DdE57Ce , smartpeer:0
    x6C7155A4A8222D478E08de251f0933923DdE57Ce , -)
25 actedOnBehalfOf (dlt:0xA5A5227fff3c3D94B0B080b47aeC9fD94364689F , smartpeer:0
    xA5A5227fff3c3D94B0B080b47aeC9fD94364689F , -)
26 endDocument

```

Listing 7.3: Excerpt from the generated PROVn file

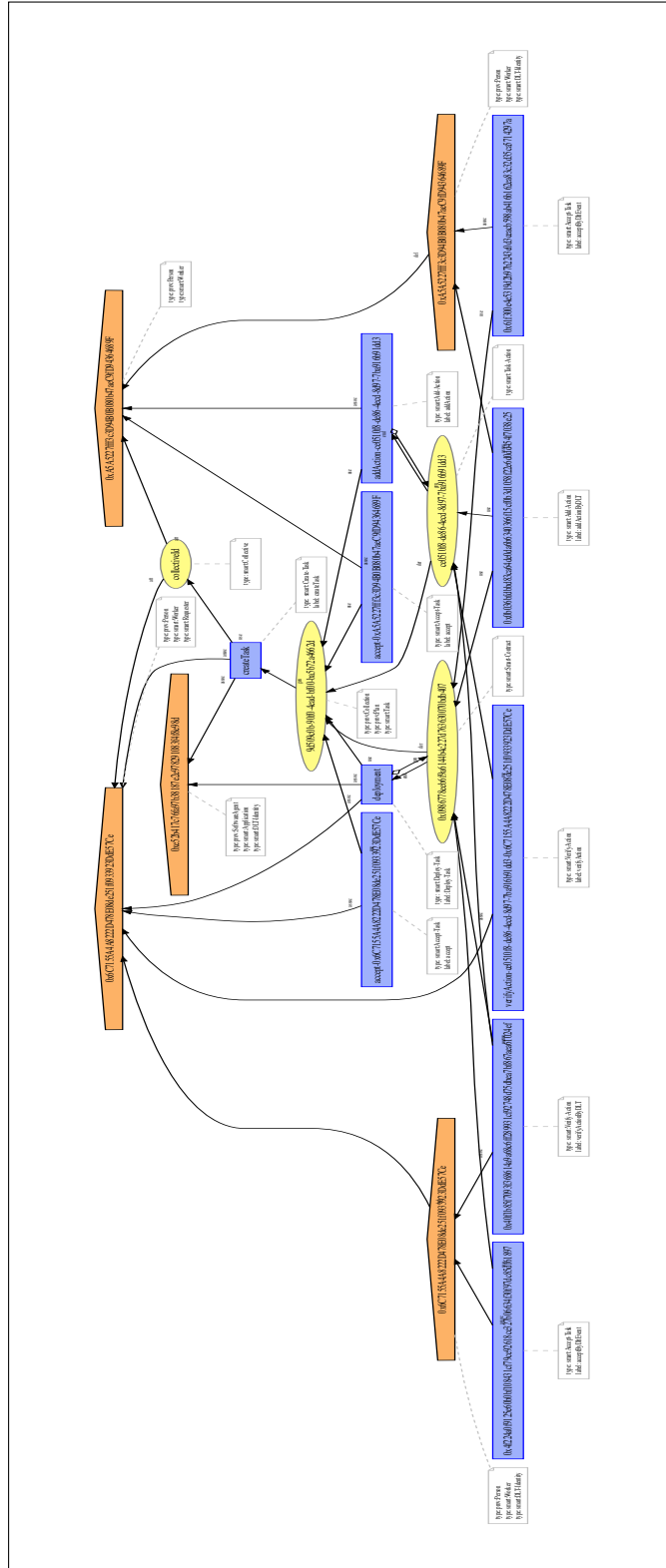


Figure 7.8: Provenance graph represented as PDF.

Conclusion

The first part of the conclusion encloses a detailed summary of the master's thesis, its research questions, elaboration and discussion. Followed by an overview of the limitations and implications of the thesis. The last section gives an outlook for future work within this research topic.

8.1 Summary

This thesis presents an approach on how to provide accountability in a collaborative computing environment with human activities based on a Distributed Ledger Technology (DLT). The definition and key concepts of provenance and accountability have been introduced as well as the technical background on DLT.

An accountable system, as by definition in subsection 3.2.1, can be achieved with the help of provenance and DLT. Provenance itself is a fundamental aspect of an accountable system and its design, because it mostly consists of an explicit representation of past events in order to understand and trace the origin of decisions, data, and activities. As described in this thesis (see subsection 3.3.3), reliable provenance has certain characteristics, such as, tamper-resistant, to keep the integrity of the recorded data, authenticity, to identify ownership of the data and reliable collection mechanisms that are trustworthy and accurate. The PROV standard was briefly described with its core elements and structure in the background chapter 3.4.2. The complex and rather new DLT with its different types, generations and implemented architectures is not fully explored yet and still under substantial development, continually refined and adapted. During the research and work of this thesis, multiple improvements and substantial changes have been released for the underlying DLT. This rapid pace of change reflects this rather new and complex research domain of DLT. The comparison of different DLTs in chapter DLT Review shows the essential implementation and structural differences of how to achieve a DLT with the claim of preserving immutability (see also table 4.1). The consensus protocol

proof-of-work (PoW) is more accurately defined as “Mutable-By-Hashing-Power” as described in 3.5.3 - Properties of DLT.

Ethereum, with its long history, high level of support and large developer space with a variety of existing tools and libraries, as well as its versatility through the introduction of a turing-complete scripting language, Solidity, was selected for the implementation part. Cryptographically enforced agreements, such as smart contracts in Ethereum have been utilized to achieve the ability to track the transactions and by that the activities of the involved users.

The implemented accountability model as defined in section 5.2 was considered as an extension of SmartSociety, a Hybrid and Diversity-aware Collective Adaptive System (HDA-CAS) and serving as the distributive collaborative computing environment for human activities.

For evaluation and demonstration purposes the SmartSociety programming framework was extended by a working scenario that utilizes the implemented accountability component. The architecture of the SmartSociety programming framework with its implemented application scenario is illustrated in section 5.4 - Architecture. A demonstration of this implemented scenario was documented and described as an evaluation of usability in chapter 7 - Evaluation. The peers can interact with the SmartSociety programming framework and confirm their actions on the deployed smart contract connected to a Collective Based Task (CBT) of the SmartSociety. Human activities are being captured by the accountability module and written as PROV-N and Resource Description Framework (RDF) format to a file and a database (Apache Jena) to have a web-based query option with SPARQL commands.

8.2 Limitations

The implemented accountability component, as well as the implemented scenario with its user interface, are considered to be prototypes. Non-functional requirements such as authentication and authorization have not been implemented as a complete solution for the sake of simplicity and for demonstration purposes. Therefore, the implementation is by no means production-ready.

The implemented smart contract has not been deployed to the live main network of Ethereum. It was deployed and tested only at the test network Rinkeby and the local in-memory network Ganache. The smart contract was kept simple in its functionality as upgrades, updates, and security of smart contract is out of scope for this thesis due to time and resource constraints.

A smart contract interaction by sending a transaction to the network can be very costly, depending on how much calculations are performed and how much memory is required. This has to be considered before utilizing such an approach since costs arise for the peers and the application for collaboration in this setup. Once the transaction is confirmed,

the operations must be performed on every node of the Ethereum distributed computing platform. This is always expensive and can also be seen as inefficient.

The public wallet address in Ethereum serves as the defined identity of the represented SmartSociety peer. Proper identity management has not been implemented.

Although the architecture and implementation of the accountability component was designed to be extendable, maintainable, and interoperable, it was built considering the SmartSociety environment and its dependencies.

The provenance capturing mechanism has been implemented directly in this application. This could be further enhanced by implementing the declarative approach, PROV templating, for recording and generation of provenance data.

8.3 Implications

The proposed accountability model based on the Ethereum and its smart contracts emerges as a possible candidate to solve the problem of providing accountability in a distributed computing environment without the essential requirement of having a trusted third party. The defined provenance record based on the application itself and the events stored transparently and publicly available on the DLT enable trustworthiness and correctness of human activities, attribute actions to individual contributors and thereby track misconduct.

The provenance recording in PROV standardized format is capable of representing actions and activities of human peers and the application in regard to SmartSociety. With the standardized format, it is possible to convert the representation to other formats and also generate a graphic representation of the provenance information.

As the domain of DLT is rather new to research and science, its security as well as further improvements regarding usability have yet to be determined. New types of DLT which are considered as the third generation are even newer and include different underlying structures that might affect the properties of DLT in general. As this area is still being intensively researched and is not being fully adopted, it must still be treated with caution.

8.4 Outlook

There are many different implementations of DLT and just as many questions on this topic. Among other things, research is currently working on the most reliable and efficient consensus algorithm and on other integral components of such a distributed system.

The PoW consensus protocol could be completely replaced by proof-of-stake (PoS) in the near future. More research is required about the implementation details and reliability of such PoS protocols in DLT without sacrificing its properties. The incentive plays an essential role for keeping the network protocol active and secure. Furthermore, research

8. CONCLUSION

focused at efficiency and reliability of smart contracts particularly in Ethereum and its costs from an economic point of view is required.

The PROV Data Model

In the following subsections, common essential elements of PROV-DM and each component will be briefly described, including an example in the notational PROV-N syntax for better understanding. An overview of all types and relations with their corresponding PROV-N expression is shown in Table A.2.

A.1 Common Elements

Common essential elements used within the PROV data model are listed in this subsection.

Namespace A namespace is represented by an Internationalized Resource Identifier (IRI) ¹. Namespaces can be assigned to specific prefixes that can be used as an abbreviation standing for the declared namespace. A default namespace can also be set, which does not require a prefix. The PROV namespace with the prefix `prov` is referenced by the IRI `http://www.w3.org/ns/prov#`. For example, a referenced element with `prov:entity` would be equivalent to the absolute IRI of `http://www.w3.org/ns/prov#entity`.

Qualified Name A qualified name is always subject to a namespace. It consists of a local name and an optional prefix defining the mapped namespace. When the prefix is not present, then it refers to the default namespace. PROV-DM specifies that a qualified name can be mapped to an IRI by concatenating the IRI associated with the prefix and the local part.

Identifier An identifier is described by a qualified name and uniquely identifies the element within the scope of the namespace. The identifier is mandatory for the elements `Entity` (A.2), `Activity` (A.2) and `Agent` (A.4). For other elements it is optional to have an identifier.

¹<https://www.ietf.org/rfc/rfc3987.txt>

Attributes and their Values The PROV data model includes a predefined set of attributes in the PROV namespace that enables metadata to be added to an object. An overview of the predefined attributes is presented in Table A.1. Attribute values are constants defined as a type of string, time, number, qualified name, IRI, and encoded binary data. The recommended types by the PROV standard are either RDF-compatible or qualified names.

Attribute	Allowed In	value
prov:label	any construct	A Value of type xsd:string
prov:location	Entity, Activity, Agent, Usage, Generation, Invalidation, Start, and End	A Value
prov:role	Usage, Generation, Invalidation, Association, Start, and End	A Value
prov:type	any construct	A Value
prov:value	Entity	A Value

Table A.1: PROV-DM Attributes Overview [21]

Reserved Attributes for Extensibility The PROV namespace includes a set of reserved attributes (`prov:type`, `prov:role`, `prov:location`) that allow designers to extend and adapt their model to their specific application or domain.

```

1  activity(ex:work)
2  entity(ex:laptop4, [prov:type='ex:Computer', prov:location='
   Workplace A'])
3  used(ex:work, ex:laptop4, [prov:role='day-to-day machine'])

```

Listing A.1: An entity of type Computer at a defined location used in a specific role in a work activity.[21]

A.2 Component 1: Entities and Activities

The first component is focused on entity and activity and their related interrelations, such as `Used`, `WasGeneratedBy`, `WasStartedBy`, `WasEndedBy`, `WasInvalidatedBy` and `WasInformedBy`.

Entity An entity represents either a digital, physical, or conceptual object type, that can be real or imaginary. It includes a mandatory `identifier` (`id`) and can have multiple `attribute-value` pairs describing additional information (see Listing A.2). An example of an entity in PROV-N with the type of a document is represented in Listing A.3. For PROV data visualizations, entities are depicted as yellow ovals (see Figure 3.2).

```

1  entity(id, [attr1=val1, ...])

```

Listing A.2: PROV-N definition of entity.[21]

```
1 entity(tr:WD-prov-dm-20111215, [prov:type="document", ex:version="2"]
   )
```

Listing A.3: An entity of type document in the second version.[21]

Activity An activity describes a process over a period of time that may include generating, using, consuming, processing, transforming, modifying or relocating entities. In PROV-N it includes a mandatory identifier `id` and optional `startTime`, `endTime` and `attributes` as attribute-value pairs (see Listing A.4). The illustration of activities in PROV are denoted as purple rectangles (see Figure 3.2).

```
1 activity(id, startTime, endTime, [attr1=val1, ...])
```

Listing A.4: PROV-N definition of activity.[21]

```
1 activity(a1, 2011-11-16T16:05:00, 2011-11-16T16:06:00, [ ex:host="
   server.example.org", prov:type='ex:edit' ])
```

Listing A.5: An example of an activity.[21]

Generation (*wasGeneratedBy*) Generation is used for the completion of the creation of a new entity by an activity. In PROV-N syntax only the entity `id` is mandatory and the remaining fields `id`, `activity`, `time` and `attribute-value` pairs are optional with the restriction that at least one of them must be present (see Listing A.6).

```
1 wasGeneratedBy(id; e, a, t, [attr1=val1, ...])
```

Listing A.6: PROV-N definition of `wasGeneratedBy`. [21]

```
1 wasGeneratedBy(e1, a1, 2001-10-26T21:32:52, [ex:port="p1"])
2 wasGeneratedBy(e, -, 2001-10-26T21:32:52)
```

Listing A.7: Two generations of an entity. The latter one without an activity. [21]

Usage (*used*) Usage refers to the starting process of utilizing an entity by an activity. The PROV-N syntax for `used` has a mandatory `activity id` field and optional fields: `id`, `entity (id)`, `time`, and `attribute-value` pairs with the restriction that at least one of the optional fields must be present (see Listing A.8). It is possible that a reference to a given entity may appear in multiple usages that share a given activity identifier.

```
1 used(id; activity, entity, time, [attr1=val1, ...])
```

Listing A.8: PROV-N definition of `used`. [21]

```
1 used(a1, e1, 2011-11-16T16:00:00, [ex:parameter="p1"])
2 used(a1, e2, 2011-11-16T16:00:01, [ex:parameter="p2"])
```

Listing A.9: Two entities used by one activity. [21]

Communication (*wasInformedBy*) A communication `wasInformedBy` represents the exchange of some unspecified entity between two activities. The PROV-N syntax for `wasInformedBy` must have set the `informed activity2` and `informant activity1` fields and has further optional fields: `id` and `attribute-value` pairs (see Listing A.10). It can be read as `activity1` was informed by `activity2` by an unspecified entity.

```
1 wasInformedBy(id; activity2, activity1, [attr1=val1, ...])
```

Listing A.10: PROV-N definition of `wasInformedBy`. [21]

```
1 activity(a1, [prov:type="traffic regulations enforcing"])
2 activity(a2, [prov:type="fine paying"])
3 wasInformedBy(a2, a1)
```

Listing A.11: Example: Activity `a2` was informed by activity `a1` and generated implicitly an unspecified entity. [21]

Start (*wasStartedBy*) Start `wasStartedBy` marks the trigger, an entity, to set off an activity. The activity exists only when it is started. Any usage, generation or invalidation relations for an activity follows the start of the activity. The PROV-N syntax for `wasStartedBy` must have set the `activity2-id` referencing the started activity and has further optional fields: `id`, `trigger (entity-id)`, `starter (activity1-id)`, `time` and `attribute-value` pairs (see Listing A.12). At least one of the optional fields has to be present.

```
1 wasStartedBy(id; activity2-id, entity-id, activity1-id, time, [attr1
   =val1, ...])
```

Listing A.12: PROV-N definition of `wasStartedBy`. [21]

```
1 entity(e1, [prov:type="email message"])
2 activity(a0, [prov:type="write"])
3 wasGeneratedBy(e1, a0)
4 wasStartedBy(a1, e1, a0, 2011-11-16T16:05:00)
```

Listing A.13: Example of starting an activity (`a1`) write with a generated entity (`e1`). [21]

End (*wasEndedBy*) In contrast to *Start A.2* `wasEndedBy` marks the end of an activity triggered by an entity. The terminated activity ceases to exist after it was ended. The PROV-N syntax for `wasEndedBy` must have set the `activity (id)` referencing the ended activity and has further optional fields: `id`, `trigger (entity id)`, `ender (activity id)`, `time` and `attribute-value` pairs (see Listing A.14). At least one of the optional fields has to be present.

```
1 wasEndedBy(id; activity, trigger, ender, time, [attr1=val1, ...])
```

Listing A.14: PROV-N definition of `wasEndedBy`. [21]

```

1 entity(e1, [prov:type="approval document"])
2 activity(a1, [prov:type="Editing"])
3 wasEndedBy(a1, e1, -, -)

```

Listing A.15: An example of ending the activity (a1 Editing) by the trigger entity (e1).[21]

Invalidation (*wasInvalidatedBy*) Invalidation, represented by `wasInvalidatedBy`, sets an existing entity to no longer exist by a specified activity. The lifecycle of an entity starts with the *Generation A.2* and ends with `wasInvalidatedBy`. An entity may be invalidated or ceasing to exist for different reasons. For example, the entity was destroyed, consumed or expired. The PROV-N syntax for `wasInvalidatedBy` must have an identifier (`id`) set for the invalidated entity. Additional optional fields are an `id`, an activity that invalidated the entity `activity` (`id`), `time` and `attributes` with attribute-value pairs. (see Listing A.16). At least one of the optional fields has to be present.

```

1 wasInvalidatedBy(id; entity, activity, time, [attr1=val1, ...])

```

Listing A.16: PROV-N definition of `wasInvalidatedBy`.[21]

```

1 entity(ex:File1)
2 activity(ex:delete)
3 wasInvalidatedBy(ex:File1, ex:delete, 2018-09-03T01:31:00, [ex:
  reason="No longer required."])

```

Listing A.17: Example of invalidated entity by an activity. [21]

A.3 Component 2: Derivations

Derivation (*wasDerivedFrom*) Derivation (`wasDerivedFrom`) represents a transformation of an entity into another updated or new entity. The level of details of the derivation process can be set as required, depending on the provided fields. The PROV-N syntax for `wasDerivedFrom` must have at least following fields: `generatedEntity` (`id`) and the `usedEntity` (`id`). Besides the mandatory fields, there are optional fields: an identifier `id`, an activity that uses and generates the entities `activity` (`id`), `generation` (`id`) for the generation of `generatedEntity`, `usage` (`id`) for using the `usedEntity` (`id`) and the `attributes` with attribute-value pairs (see Listing A.18). There are three subtypes of `wasDerivedFrom`, defined as `hadPrimarySource`, `wasQuotedFrom`, and `wasRevisionOf`.

```

1 wasDerivedFrom(id; generatedEntity, usedEntity, activity, generation
  , used, [attr1=val1, ...])

```

Listing A.18: PROV-N definition of `wasDerivedFrom`.[21]

```

1  entity(ex:File2 , [prov:type="PNG"])
2  entity(ex:Dataset , [prov:type="Excel" , ex:description="Dataset for
   Finance"])
3  wasDerivedFrom(ex:File2 , ex:Dataset)

```

Listing A.19: Example of starting an activity (a1) write with a generated entity (e1). [21]

Derivation Revision (*prov:Revision*) Revision describes the derivation of an entity into a revised entity, which is based on the original entity. The type `prov:Revision` has to be set at the `wasDerivedFrom` relation (see Listing A.20).

```

1  wasDerivedFrom(generatedEntity , usedEntity , [prov:type='prov:
   Revision'])

```

Listing A.20: Example of `wasDerivedFrom` with the type `Revision`. [21]

Derivation Quotation (*prov:Quotation*) The derivation relation of type `Quotation` can be used to copy some or all of an entity and reference the original entity as the source of the content. To specify a derivation as a quote from a source, the type `prov:Quotation` has to be set (see Listing A.21).

```

1  wasDerivedFrom(generatedEntity , usedEntity , [prov:type='prov:
   Quotation'])

```

Listing A.21: Example of `wasDerivedFrom` with the type `Quotation`. [21]

A.4 Component 3: Agents, Responsibility, and Influence

Agent An agent can have the responsibility for an activity (`wasAssociatedWith` A.4), entity (`wasAttributedTo`) or another agent (`actedOnBehalfOf`). In the PROV namespace following predefined agent types are available: `prov:Person`, `prov:SoftwareAgent`, and `prov:Organization`. The PROV-N syntax for an agent must have an identifier (`id`) and can have `attributes` with attribute-value pairs set (see Listing A.22).

```

1  agent(id , [attr1=val1 , ...])

```

Listing A.22: PROV-N definition of an Agent.[21]

```

1  agent(ex:Alice , [ex:employee="1234" , ex:name="Alice" , prov:type='
   prov:Person'])

```

Listing A.23: Example of an Agent with additional attributes set. [21]

Attribution (*wasAttributedTo*) The attribution relation describes the assignment of an agent to an entity. This relation is useful when the activity for assigning an agent is unknown. The PROV-N syntax for `wasAttributedTo` has the mandatory fields of an entity (`id`) and an agent (`id`). Additional optional fields can be set, such as an `id` and `attributes` with attribute-value pairs. (see Listing A.24).

```
1 wasAttributedTo(id; entity , activity , [attr1=val1 , ...])
```

Listing A.24: PROV-N definition of `wasAttributedTo`. [21]

```
1 agent(ex:Alice , [prov:type='Person' ])
2 entity(ex:Book1 , [prov:type='Book' ])
3 wasAttributedTo(ex:Book1 , ex:Alice , [prov:type="authorship"])
```

Listing A.25: Example of authorship (Alice) attribution of an entity (Book1). [21]

Association (*wasAssociatedWith*) An agent can be assigned responsibility to activity by `wasAssociatedWith`. The PROV-N syntax for `wasAssociatedWith` has only one mandatory field, which is an activity (`id`). Additional optional fields are an `id`, an agent that is responsible, a plan that the agent relied on, and `attributes` with attribute-value pairs. (see Listing A.26). At least one of the optional fields has to be set. A plan is modeled as an entity with a type of plan (`prov:Plan`). With the concept of a plan, it is possible to have a set of actions or steps assigned to one or more agents to achieve a goal.

```
1 wasAssociatedWith(id; activity , agent , plan , [attr1=val1 , ...])
```

Listing A.26: PROV-N definition of `wasAssociatedWith`. [21]

```
1 agent(ex:Alice , [prov:type='Person' ])
2 activity(ex:a , [prov:type="workflow execution"])
3 wasAssociatedWith(ex:a , ex:Alice , ex:wf)
4 entity(ex:wf , [prov:type='prov:Plan' , ex:label="Workflow 1"])
```

Listing A.27: Example of authorship (Alice) attribution of an entity (Book1). [21]

Delegation (*actedOnBehalfOf*) The delegation relation describes the assignment of authority and responsibility to an agent by itself or another agent. The PROV-N syntax for `actedOnBehalfOf` has the mandatory fields of an agent `delegate` (`id`) and a responsible agent (`id`). Additional optional fields can be set, such as an `id`, the delegated activity and `attributes` with attribute-value pairs. (see Listing A.28).

```
1 actedOnBehalfOf(id; delegate , responsible , activity , [attr1=val1 , ...])
```

Listing A.28: PROV-N definition of `actedOnBehalfOf`. [21]

```

1 agent(ex:Alice, [prov:type='Person'])
2 agent(ex:Bob, [prov:type='Person'])
3 activity(ex:a, [prov:type="signing contract"])
4 wasAssociatedWith(ex:a, ex:Alice, ex:c)
5 entity(ex:c, [prov:type='prov:Plan', ex:label="Contract 1"])
6 actedOnBehalfOf(ex:Bob, ex:Alice, ex:a)

```

Listing A.29: An example of delegating responsibility for signing a contract from one person/agent to another. [21]

Influence (*wasInfluencedBy*) The Influence relation describes an entity, activity, or agent which was affected by another entity, activity, or agent. The PROV-N syntax for `wasInfluencedBy` describe two mandatory fields, the `influencer` (entity, activity or agent) and the `influencee` (entity, activity, agent). Additional optional fields are the `id` and the `attributes` with attribute-value pairs (see Listing A.30).

```

1 wasInfluencedBy(id; influencee, influencer, [attr1=val1, ...])

```

Listing A.30: PROV-N definition of `wasInfluencedBy`. [21]

```

1 agent(w3:Consortium, [prov:type='Organization'])
2 entity(ex:prov-dm, [prov:type='ex:Document', ex:label="PROV data
  model"])
3 wasInfluencedBy(ex:prov-dm, w3:Consortium)

```

Listing A.31: An example of a document which was influenced by the World Wide Web Consortium (W3C). [21]

A.5 Component 4: Bundles

Bundle Bundle is a specific type of `entity` A.2 and represents a named collection of provenance descriptions. This structure provides the mechanism to describe the provenance of provenance. To bundle different descriptions, it is put into a structure starting with `bundle id` and ends with `endBundle`. To describe the provenance of this bundle, an `entity` with the bundle id has to be set (see Listing A.32). An example can be found in Listing A.33.

```

1 bundle id
2 ...
3 endBundle
4 entity(id, [prov:type='prov:Bundle', attr1=val1, ... ] )

```

Listing A.32: PROV-N definition of a Bundle. [21]


```

1 bundle bob:bundle1
2   entity(ex:report1, [prov:type="report", ex:version=1])
3   wasGeneratedBy(ex:report1, -, 2012-05-24T10:00:01)
4   endBundle
5
6   entity(bob:bundle1, [prov:type='prov:Bundle'])
7   wasGeneratedBy(bob:bundle1, -, 2012-05-24T10:30:00)
8   wasAttributedTo(bob:bundle1, ex:Bob)

```

Listing A.33: Example of a bundle of descriptions and its provenance. [21]

A.6 Component 5: Alternate Entities

This component enables linking of entities with two specific relations.

Specialization (*specializationOf*) This relation describes a specialized entity that shares all aspects of another entity but also has additional distinct aspects. In particular, aspects include a time period and a context associated with the entity. The PROV-N syntax defines *specializationOf* to have mandatory a specific entity called *infra* and a general entity *supra*, for which the *infra* is based on (see Listing A.34).

```

1 specializationOf(infra, supra)

```

Listing A.34: PROV-N definition of a specializationOf.[21]

```

1 specializationOf(ex:bbcNews2012-03-23, bbc:news/)

```

Listing A.35: Example of a specific entity, including a time aspect, which is based on *bbc:news/*. [21]

Alternate (*alternateOf*) Alternate describes two entities with aspects of the same thing. The entities may have the same aspects or different and further may overlap in time or not. The PROV-N syntax defines *alternateOf* with two entity identifiers, *alternate1* and *alternate2* (see Listing A.36).

```

1 alternateOf(alternate1, alternate2)

```

Listing A.36: PROV-N definition of a specializationOf.[21]

```

1 entity(ex:bbcNews2012-03-23)
2 entity(ex:bbcNews2012-03-26)
3 alternateOf(ex:bbcNews2012-03-23,ex:bbcNews2012-03-26)

```

Listing A.37: Example of two alternative entities which are both a specialization of *bbc:news/*. [21]

A.7 Component 6: Collections

Collection Collection is an entity with the `prov:type` of either `prov:EmptyCollection` or `prov:Collection` (see Listing A.38). It defines a structure that may include multiple entities as members of the collection. The relation of membership is defined next A.7.

```
1 entity(c0, [prov:type='prov:EmptyCollection'])
2 entity(c1, [prov:type='prov:Collection'])
```

Listing A.38: An empty collection (c0) and a collection with unknown content (c1). [21]

Membership (*hadMember*) The membership relation describes the association of an entity to a collection. It includes a referenced `collection` (id) and the `entity` (id) to be included into the referenced collection. The definition in PROV-N is depicted in Listing A.39

```
1 hadMember(collection, entity)
```

Listing A.39: PROV-N definition of `hadMember`. [21]

```
1 entity(e0)
2 entity(e1)
3
4 entity(c, [prov:type='prov:Collection']) // c is a collection, with
   unknown content
5 hadMember(c, e0)
6 hadMember(c, e1)
```

Listing A.40: Example of a collection (c) that includes e0 and e1 as members. [21]

A.8 Prov-DM Types and Relations

Type or Relation Name	Representation in the PROV-N notation	Component
Entity	entity(id, [attr1=val1, ...])	Component 1: Entities/Activities
Activity	activity(id, st, et, [attr1=val1, ...])	
Generation	wasGeneratedBy(id;e,a,t,attrs)	
Usage	used(id;a,e,t,attrs)	
Communication	wasInformedBy(id;a2,a1,attrs)	
Start	wasStartedBy(id;a2,e,a1,t,attrs)	
End	wasEndedBy(id;a2,e,a1,t,attrs)	
Invalidation	wasInvalidatedBy(id;e,a,t,attrs)	Component 2: Derivations
Derivation	wasDerivedFrom(id; e2, e1, a, g2, u1, attrs)	
Revision	... prov:type='prov:Revision' ...	
Quotation	... prov:type='prov:Quotation' ...	
Primary Source	... prov:type='prov:PrimarySource' ...	Component 3: Agents, Responsibility, Influence
Agent	agent(id, [attr1=val1, ...])	
Attribution	wasAttributedTo(id;e,ag,attr)	
Association	wasAssociatedWith(id;a,ag,pl,attrs)	
Delegation	actedOnBehalfOf(id;ag2,ag1,a,attrs)	
Plan	... prov:type='prov:Plan' ...	
Person	... prov:type='prov:Person' ...	
Organization	... prov:type='prov:Organization' ...	
SoftwareAgent	... prov:type='prov:SoftwareAgent' ...	
Influence	wasInfluencedBy(id;e2,e1,attrs)	Component 4: Bundles
Bundle constructor	bundle id description_1 ... description_n endBundle	
Bundle type	... prov:type='prov:Bundle' ...	Component 5: Alternate
Alternate	alternateOf(alt1, alt2)	
Specialization	specializationOf(infra, supra)	Component 6: Collections
Collection	... prov:type='prov:Collection' ...	
EmptyCollection	... prov:type='prov:EmptyCollection' ...	
Membership	hadMember(c,e)	

Table A.2: PROV-DM Types and Relations [21]

A.9 UML Prov-DM Component Overviews

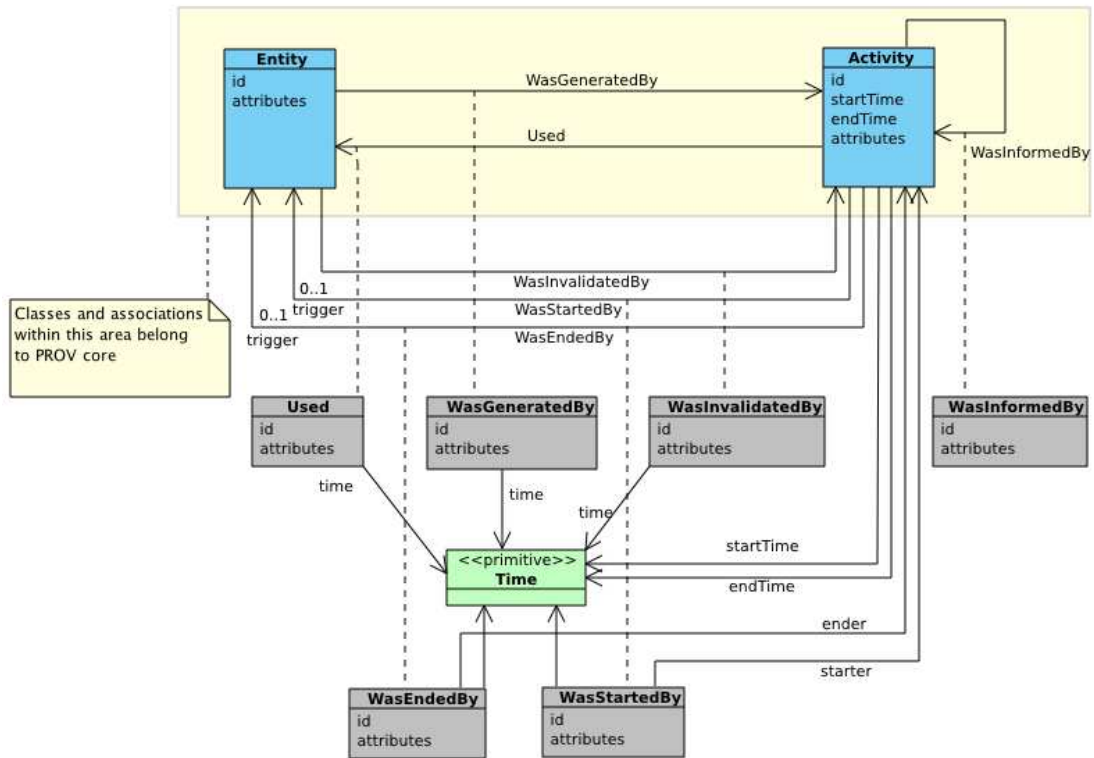


Figure A.1: Unified Modeling Language (UML) Overview Component 1: Entities and Activities [21]

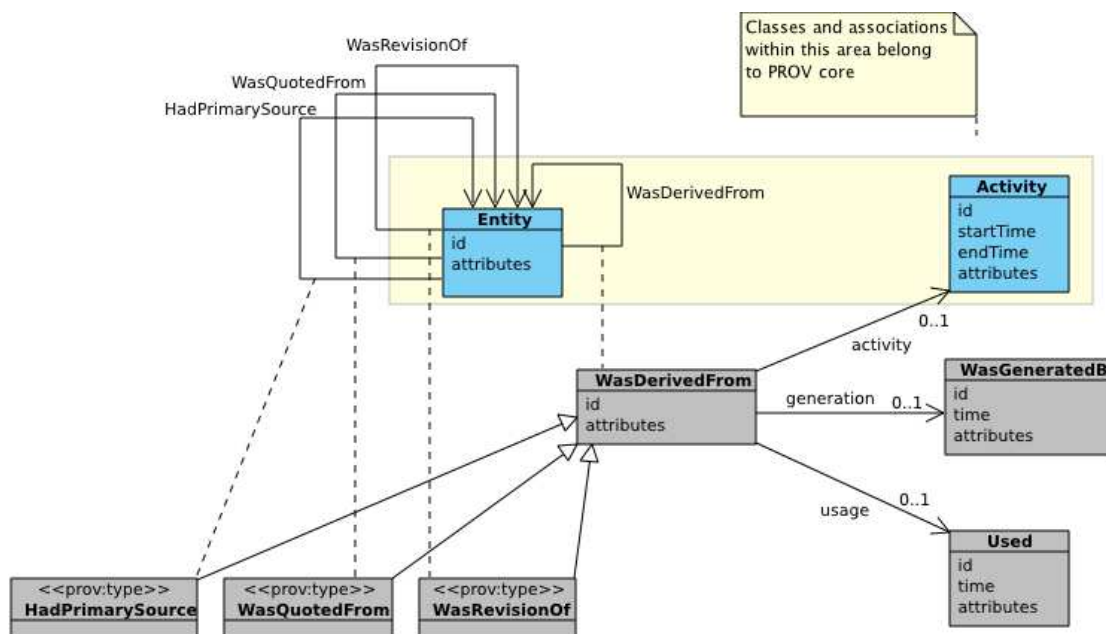


Figure A.2: UML Overview Component 2: Derivation [21]

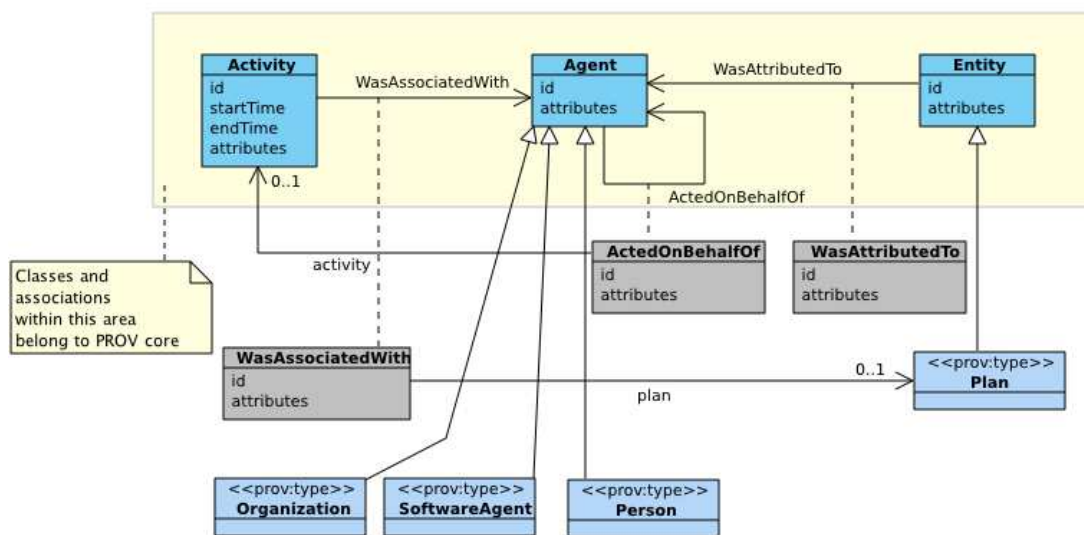


Figure A.3: UML Overview Component 3: Agents and Responsibility [21]

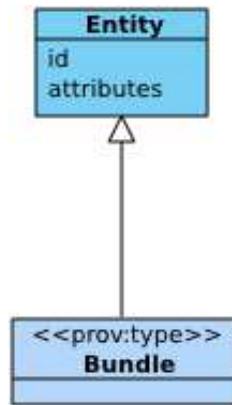


Figure A.4: UML Overview Component 4: Bundle [21]

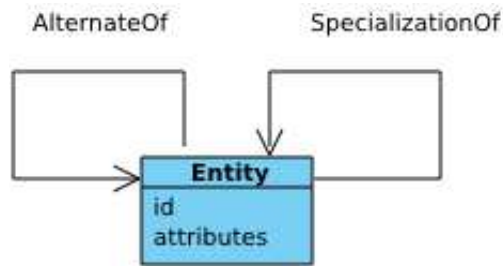


Figure A.5: UML Overview Component 5: Alternates [21]

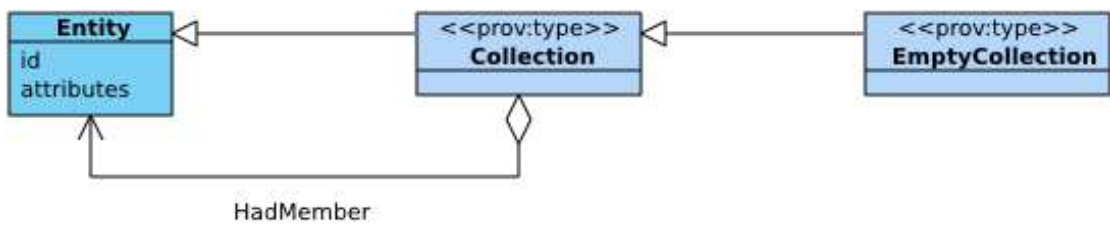


Figure A.6: UML Overview Component 6: Collections [21]

Type	Core concept
prov:Bundle	Entity
prov:Collection	Entity
prov:EmptyCollection	Entity
prov:Organization	Agent
prov:Person	Agent
prov:Plan	Entity
prov:PrimarySource	Derivation
prov:Quotation	Derivation
prov:Revision	Derivation
prov:SoftwareAgent	Agent

Table A.3: PROV-DM predefined types (valid values for prov:type) [21]



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

DLT

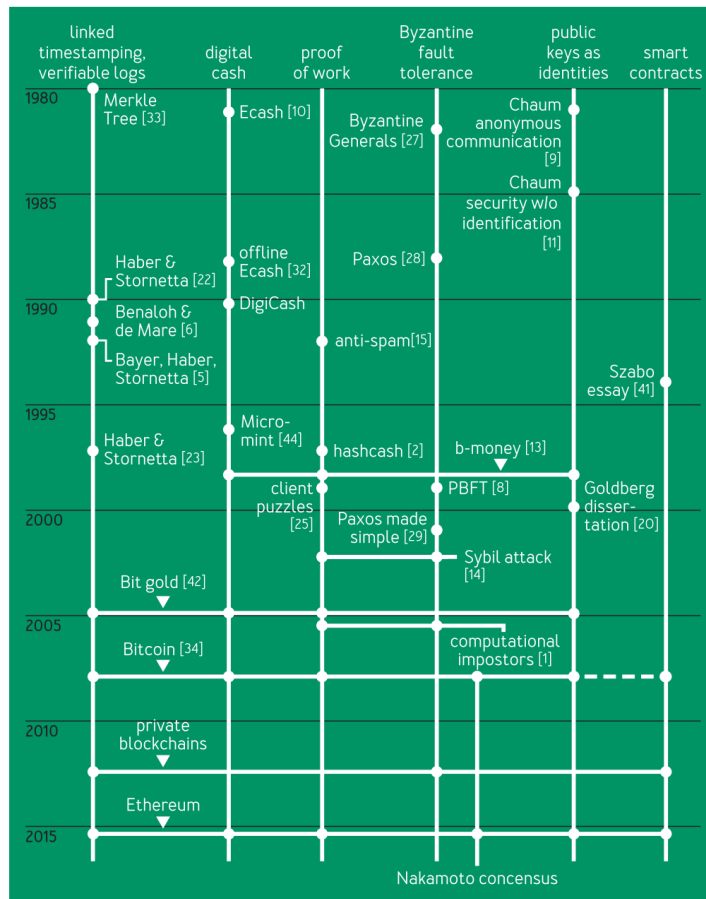


Figure B.1: Chronology of fundamental concepts used in DLT [41]



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

3.1	Overview of provenance characteristics [23]	10
3.2	Provenance Core Concepts (PROV-DM Types/Relations) [29]	12
3.3	PROV graph for writing and editing a document (author's view)[21]	14
3.4	Bitcoin transactions - The chain of ownership [5]	20
3.5	Block Structure [5]	21
3.6	Linked list of blocks - Blockchain [5]	21
3.7	The web3suite of a Decentralized Application (DApp).	27
4.1	IOTA - The Tangle [64], [65]	30
4.2	Nano Block Lattice with a send (S) and receive (R) block, signed by their chain owner [59]	32
5.1	The architecture of SmartSociety platform with users and peers. [4]	36
5.2	SmartSociety Programming model with CBT lifecycle. [71]	37
5.3	High-Level Accountability Model	39
5.4	UML use case diagram overview of the system	43
5.5	Overview of the Accountability Module in context of SmartSociety	49
5.6	Overview of the Accountability Component	50
6.1	SmartAcc - Sequence diagram	60
7.1	Initial Screens	70
7.2	Task creation and its displayed details	71
7.3	Task by User accepted	72
7.4	All users accepted and new Task state (Execution)	72
7.5	MetaMask - Transaction confirmation for accepting the Task	73
7.6	Task accepted and new state	74
7.7	Task state completed	75
7.8	Provenance graph represented as PDF.	78
A.1	UML Overview Component 1: Entities and Activities [21]	94
A.2	UML Overview Component 2: Derivation [21]	95
A.3	UML Overview Component 3: Agents and Responsibility [21]	95
A.4	UML Overview Component 4: Bundle [21]	96

A.5 UML Overview Component 5: Alternates [21]	96
A.6 UML Overview Component 6: Collections [21]	96
B.1 Chronology of fundamental concepts used in DLT [41]	99

List of Listings

6.1	Compilation of the smart contract File (.sol).	61
6.2	Command to generate wrapper code for the compiled Solidity file. . . .	61
6.3	Defined Events of the Task contract.	63
6.4	A defined modifier of the Task contract.	63
7.1	One peer definition from the Peers.json file.	67
7.2	Comand to generate PROV graph from provn File	76
7.3	Excerpt from the generated PROVN file	77
A.1	An entity of type Computer at a defined location used in a specific role in a work activity.[21]	84
A.2	PROV-N definition of entity.[21]	84
A.3	An entity of type document in the second version.[21]	85
A.4	PROV-N definition of activity.[21]	85
A.5	An example of an activity.[21]	85
A.6	PROV-N definition of wasGeneratedBy.[21]	85
A.7	Two generations of an entity. The latter one without an activity. [21] .	85
A.8	PROV-N definition of used.[21]	85
A.9	Two entities used by one activity. [21]	85
A.10	PROV-N definition of wasInformedBy.[21]	86
A.11	Example: Activity a2 was informed by activity a1 and generated implicitly an unspecified entity. [21]	86
A.12	PROV-N definition of wasStartedBy.[21]	86
A.13	Example of starting an activity (a1) write with a generated entity (e1). [21]	86
A.14	PROV-N definition of wasEndedBy.[21]	86
A.15	An example of ending the activity (a1 Editing) by the trigger entity (e1).[21]	87
A.16	PROV-N definition of wasInvalidatedBy.[21]	87
A.17	Example of invalidated entity by an activity. [21]	87
A.18	PROV-N definition of wasDerivedFrom.[21]	87
A.19	Example of starting an activity (a1) write with a generated entity (e1). [21]	88
A.20	Example of wasDerivedFrom with the type Revision. [21]	88
A.21	Example of wasDerivedFrom with the type Quotation. [21]	88

A.22 PROV-N definition of an Agent.[21]	88
A.23 Example of an Agent with additional attributes set. [21]	88
A.24 PROV-N definition of wasAttributedTo.[21]	89
A.25 Example of authorship (Alice) attribution of an entity (Book1). [21]	89
A.26 PROV-N definition of wasAssociatedWith.[21]	89
A.27 Example of authorship (Alice) attribution of an entity (Book1). [21]	89
A.28 PROV-N definition of actedOnBehalfOf.[21]	89
A.29 An example of delegating responsibility for signing a contract from one person/agent to another. [21]	90
A.30 PROV-N definition of wasInfluencedBy.[21]	90
A.31 An example of a document which was influenced by the W3C. [21]	90
A.32 PROV-N definition of a Bundle.[21]	90
A.33 Example of a bundle of descriptions and its provenance. [21]	91
A.34 PROV-N definition of a specializationOf.[21]	91
A.35 Example of a specific entity, including a time aspect, which is based on <i>bbc:news/</i> . [21]	91
A.36 PROV-N definition of a specializationOf.[21]	91
A.37 Example of two alternative entities which are both a specialization of <i>bbc:news/</i> . [21]	91
A.38 An empty collection (c0) and a collection with unknown content (c1). [21]	92
A.39 PROV-N definition of hadMember.[21]	92
A.40 Example of a collection (c) that includes e0 and e1 as members. [21]	92

List of Tables

3.1	PROV-DM Component Overview [21]	14
4.1	DLTs in comparison	33
4.2	GitHub development activity for different DLTs - accessed on 2020-03-08	33
5.11	PROV-DM Mapping	53
5.12	Defined Namespaces	54
6.1	Technology Stack of SmartAcc	58
7.1	Task Application Scenario Technology Stack	66
A.1	PROV-DM Attributes Overview [21]	84
A.2	PROV-DM Types and Relations [21]	93
A.3	PROV-DM predefined types (valid values for prov:type) [21]	97



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Glossary

- Apache Jena** An open source framework for semantic web or linked-data applications ¹. 57, 63, 76, 80
- Apache-2.0 License** A permissive free software license defined by the Apache Software Foundation. 58
- Artistic License 2.0** A free and open source software license. 66
- Base58Check** Base58 is a text based binary encoding format based on 58 characters developed for Bitcoin and other cryptocurrencies. The representation is human readable, compact and includes error detection. 19
- Bitcoin** Bitcoin is a DLT based on the Blockchain structure, introduced in 2008. ix, 15, 18, 22, 29, 33, 58, 62, 107, 110
- Block Lattice** A directed acyclic graph data structure defined for Nano. 31, 33, 110
- Blockchain** Blockchain refers to a specific structure and type of DLT. The most prominent example is Bitcoin. 29, 30, 33, 107
- Bootstrap** An open source front-end component library to build responsive web applications ². 66, 68
- Byzantine Fault Tolerance** A property of a distributed system that tolerates the class of failures described with the Byzantine Generals Problem. 15, 21, 22, 110, 113
- Byzantine Generals Problem** Describes in computer science in an abstract way the problem of coping with component failure or corrupted components in a reliable system. 21, 23, 107
- Chainpoint** An open standard ³ for anchoring data with help of DLT to create a timestamp proof. 27, 51

¹<https://jena.apache.org/>

²<https://getbootstrap.com/>

³<https://chainpoint.org/>

- Common Development and Distribution License** A free and open-source software license, produced by Sun Microsystems and based on the Mozilla Public License. 113
- Common Public License** An open source software license published by IBM. 113
- DAO** Data Access Object is a design pattern that provides an abstract interface to some type of persistence mechanism. 63
- Decentralized Application** A web application that is partially or entirely decentralized and executed on a distributed computing system such as DLT. 3, 26, 29, 101, 113
- delegated proof-of-stake** A consensus protocol ⁴ based on PoS. 23, 32, 113
- DOM** Document Object Model is an application programming interface for an HTML or XML document. The representation is a tree structure with each node representing a part of the document. 109
- double spending** Double spending refers in digital cash systems to the attempt of spending the same single digital token in more than one transaction. 15, 22
- ECMAScript** A scripting language specification designed by the standards organization Ecma International. 109
- Ethereum** A distributed ledger technology with a blockchain data structure and support of the scripting language Solidity. ix, xi, 18, 22, 23, 25–27, 29, 30, 33, 40, 51, 58, 59, 61, 62, 66–70, 76, 80–82, 108–112
- Ethereum Virtual Machine** Ethereum Virtual Machine for executing the native script language Solidity on the Ethereum blockchain. 25, 30, 112, 113
- externally owned account** An externally owned account is a type of account in Ethereum that is controlled by its private key. 25, 30
- Fast Probabilistic Consensus** A possible future consensus protocol for IOTA super-seeding Tip Selection Algorithm (TSA). 113
- Ganache** A local in-memory Ethereum blockchain for development purpose. Ganache is part of the Truffle suite of tools and formerly known as TestRPC ⁵. 66, 80
- Genesis Block** The initial or the first block of a blockchain structure. In general this is pre-defined and hardcoded into the client software. 20, 29, 31

⁴https://de.wikipedia.org/wiki/Proof_of_Stake

⁵<https://github.com/trufflesuite/ganache>

- Git** A distributed version-control system for tracking changes in source code during software development. 109
- GitHub** A software company that provides hosting of software development versioning control using Git. 3, 33, 105
- halting problem** The halting problem describes in computability theory the problem of determining whether the program will finish running or continue to run forever. 30
- HK2** A light-weight and dynamic dependency injection framework. 57
- InterPlanetary File System** A decentralized peer-to-peer (P2P) hypermedia protocol, often used in conjunction with DLT. 27, 51, 110, 114
- IOTA** A directed acyclic graph based distributed ledger technology ⁶. 30, 31, 33, 101, 108, 111, 112
- Java** A general-purpose computer programming language that is object-oriented. 66
- java-websocket** A websocket client and server implementation written in Java ⁷. 66
- JavaScript** An interpreted, dynamic scripting language based on ECMAScript. 66, 68, 110, 112
- jQuery** A javascript library for Hypertext Markup Language (HTML) DOM tree traversal and manipulation ⁸ ⁹. 66
- Lesser General Public License** A free-software license defined and published by the Free Software Foundation. 114
- lite-server** Lightweight development web server for Node.js. 66
- mainnet** The Mainnet is referenced as the production environment for DLTs. 29, 30
- Merkle tree** Merkle tree refers to a hash tree data structure. 15, 19, 20, 27, 51
- MetaMask** A browser extension that helps to interact with the Ethereum network without running a full node. 26, 51, 66, 68, 70–74, 101
- MIT License** A permissive free software license originated at the Massachusetts Institute of Technology. 58, 66

⁶<https://www.iota.org/>

⁷<https://github.com/TooTallNate/Java-WebSocket>

⁸<https://jquery.com/>

⁹<https://github.com/jquery/jquery>

MongoDB A cross-platform document-oriented database program using the Javascript Object Notation (JSON) format. 58–60, 62, 63

Nano A decentralized cryptocurrency based on the Block Lattice data structure ¹⁰. 31–33, 107, 110

Node Package Manager A package manager for JavaScript. It is the default package manager for Node.js. 114

Node.js A JavaScript runtime environment. 66, 109, 110

Open Provenance Model The Open Provenance Model is a specification that was defined in 2008 to represent a model of artifacts in the past that explains how they were derived. 12, 114

Open Representative Voting The consensus protocol used in Nano. 32, 114

Orbit-DB A serverless, distributed P2P database, built upon InterPlanetary File System (IPFS). 27

peer-to-peer A distributed shared network or system architecture based on peers that are directly connected with no intermediary entities. ix, xi, 15, 18, 109, 114

Practical Byzantine Fault Tolerance An improved consensus algorithm of Byzantine Fault Tolerance (BFT) where members of the network are partially trusted. 23, 114

proof-of-stake A consensus protocol ¹¹ to secure a DLT based on a particular type of stake or quota within the system. 22, 33, 81, 114

proof-of-work A consensus protocol ¹² based on the computing effort to secure a DLT. At the moment implemented at Bitcoin and Ethereum. 15, 22, 30, 80, 114

PROV The W3C specified a model in a series of documents, described as PROV, for working with provenance data in a standardized way. ix, xi, 3, 10, 12–14, 51, 53–55, 57, 58, 61, 62, 65, 76, 79, 81, 83–85, 88, 101, 110, 111

PROV-DM A standardized generic data model for the PROV specification. 12–14, 53, 59, 83, 93, 97, 101, 105, 110

PROV-N An human-readable file format representing the PROV-DM. 12, 42, 51, 52, 62, 76, 80, 83–92

¹⁰<https://nano.org/en>

¹¹https://en.wikipedia.org/wiki/Proof_of_stake

¹²https://en.wikipedia.org/wiki/Proof_of_work

provconvert A tool to convert between different PROV representations ¹³. 76

Resource Description Framework A metadata data model defined as a W3C specification. It provides a conceptual description of information for web resources. 11, 42, 57, 80, 114

SHA-256 A cryptographic hashing function generating a 256-bit (32 bytes) digest for an input. 22

smart contract A smart contract is a computer protocol that is designed to digitally facilitate, verify, or enforce a contract. ix, xi, 2, 26, 39, 41, 44, 45, 47, 50, 51, 54, 58, 61, 62, 66, 69, 76, 80–82

SmartAcc The implemented accountability component prototype ¹⁴ developed in this thesis. 58, 60, 65, 66, 69, 71, 105

SmartAccTask The implemented SmartSociety accountability task scenario ¹⁵ developed during this thesis for evaluation purposes. 65–68

SmartCom Communication middleware ¹⁶ for SmartSociety. 36, 49, 50, 66, 67

SmartSociety An European Union funded research and development project for a HDA-CAS. 2–4, 35–37, 39–45, 49, 50, 60, 62, 65–69, 74, 76, 80, 81, 101, 111

SmartSociety programming framework A working implementation of the SmartSociety model with multiple demo scenarios available ¹⁷ [71]. ix, xi, 3, 4, 35, 41, 42, 52, 57, 65–67, 69, 80

Solidity A scripting language that is turing-complete and used with Ethereum. 25, 30, 61, 62, 66, 80, 108

SPARQL A query language for the file format RDF. 57, 58, 62, 63, 76, 80

Stellar Consensus Protocol A consensus protocol ¹⁸ for Stellar, an implementation of a DLT. 24, 114

Swarm Ethereum’s decentralized data storage protocol. 27

Sybil Attack An attack in computer security that undermines a reputation system by creating multiple identities. 22, 31

Tangle A directed acyclic graph based data structure defined for IOTA. 30, 31, 33

¹³<https://github.com/lucmoreau/ProvToolbox/wiki/provconvert>

¹⁴<https://bitbucket.org/alexanderp-tu/smartacc/src/master/>

¹⁵<https://bitbucket.org/alexanderp-tu/smartacctask/src/master/>

¹⁶<https://github.com/tuwiendsg/SmartCom>

¹⁷<https://gitlab.com/smartsociety/programming-framework>

¹⁸<https://www.stellar.org/papers/stellar-consensus-protocol>

Tip Selection Algorithm A consensus protocol for IOTA. 31, 108, 115

truffle A development environment and testing framework for blockchains using the Ethereum Virtual Machine (EVM) ¹⁹. 66

Turtle A file format for RDF data models. 62

Web Ontology Language 2 OWL2 is an ontology language for the Semantic Web with formally defined meaning. 13, 114

web3.js A Ethereum Application Programming Interface (API) in JavaScript. 66

Web3J A Java library enabling interactions of applications with Ethereum. 51, 61, 62

websocket A computer communication protocol providing bi-directional, full-duplex communication channels over a single Transmission Control Protocol (TCP) connection. 67, 68, 109

Whisper Ethereum's decentralized communication protocol for DApps. 27

¹⁹<https://www.trufflesuite.com/>

Acronyms

- API** Application Programming Interface. 41–43, 49–51, 69, 112
- BFT** Byzantine Fault Tolerance. 15, 17, 21, 22, 110, 113, *Glossary*: Byzantine Fault Tolerance
- CAS** Collective Adaptive System. 7, 35
- CBT** Collective Based Task. 35, 37–39, 41, 54, 60, 62, 65, 80, 101
- CDDL** Common Development and Distribution License. 58, 113, *Glossary*: Common Development and Distribution License
- CPL** Common Public License. 58, 113, *Glossary*: Common Public License
- DAG** Directed Acyclic Graph. 18, 30, 31, 33
- DApp** Decentralized Application. 3, 26, 27, 29, 33, 101, 112, 113, *Glossary*: Decentralized Application
- DDoS** Distributed Denial of Service. 30
- DLT** Distributed Ledger Technology. ix, xi, 2–5, 7, 15–23, 25–27, 29, 30, 33, 39–44, 46, 47, 49–54, 58, 61, 62, 76, 79, 81, 99, 102, 105, 107–111
- DPoS** delegated proof-of-stake. 23, 32, 33, 113, *Glossary*: delegated proof-of-stake
- DTO** Data Transfer Object. 67
- EVM** Ethereum Virtual Machine. 25, 26, 30, 62, 112, 113, *Glossary*: Ethereum Virtual Machine
- FPC** Fast Probabilistic Consensus. 113, *Glossary*: Fast Probabilistic Consensus
- GDPR** General Data Protection Regulation. 5, 36
- GPL** General Public License. 58, 66

HDA-CAS Hybrid and Diversity-aware Collective Adaptive System. 35, 36, 39, 80, 111

HTML Hypertext Markup Language. 109

HTTP Hypertext Transfer Protocol. 13, 50, 57, 58, 63

IoT Internet of Things. 18, 30

IPFS InterPlanetary File System. 27, 51, 110, 114, *Glossary*: InterPlanetary File System

IRI Internationalized Resource Identifier. 83, 84

JSON Javascript Object Notation. 42, 52, 57, 67, 110

JSR-330 Java Specification Request 330. 57

JVM Java Virtual Machine. 61, 62

LGPL Lesser General Public License. 58, 114, *Glossary*: Lesser General Public License

npm Node Package Manager. 66, 114, *Glossary*: Node Package Manager

OPM Open Provenance Model. 12, 114, *Glossary*: Open Provenance Model

OPV Open Representative Voting. 32, 33, 114, *Glossary*: Open Representative Voting

OWL2 Web Ontology Language 2. 13, 114, *Glossary*: Web Ontology Language 2

P2P peer-to-peer. ix, xi, 15, 17–19, 21, 27, 109, 110, 114, *Glossary*: peer-to-peer

PBFT Practical Byzantine Fault Tolerance. 23, 24, 114, *Glossary*: Practical Byzantine Fault Tolerance

PDF Portable Document Format. 76

PoS proof-of-stake. 22, 23, 33, 81, 108, 114, *Glossary*: proof-of-stake

PoW proof-of-work. 15, 22, 23, 30–33, 80, 81, 114, *Glossary*: proof-of-work

RDF Resource Description Framework. 11–13, 42, 51, 52, 57, 58, 62, 63, 76, 80, 84, 111, 112, 114, *Glossary*: Resource Description Framework

REST Representational State Transfer. 42, 43, 49–51

SCP Stellar Consensus Protocol. 24, 114, *Glossary*: Stellar Consensus Protocol

SHA Secure Hash Algorithm. 19

TCP Transmission Control Protocol. 112

TSA Tip Selection Algorithm. 31, 108, 115, *Glossary*: Tip Selection Algorithm

UML Unified Modeling Language. 43, 94–96, 101, 102

URI Uniform Resource Identifier. 76

UUID Universally Unique Identifier. 67

W3C World Wide Web Consortium. 12, 13, 57, 58, 62, 90, 104, 110, 111

XML Extensible Markup Language. 11–13, 57



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

- [1] Z. Xiao, N. Kathiresshan, and Y. Xiao, “A survey of accountability in computer networks and distributed systems”, *Security and Communication Networks*, vol. 9, no. 4, pp. 290–315, 2016, ISSN: 19390114. DOI: 10.1002/sec.574.
- [2] D. Catteddu, M. Felici, G. Hogben, A. Holcroft, E. Kosta, R. Leenes, C. Millard, M. Niezen, D. Nuñez, N. Papanikolaou, S. Pearson, and D. Pradelles, “Towards a Model of Accountability for Cloud Computing Services Proposed Definitions of Accountability in the Cloud”, *International Workshop on Trustworthiness, Accountability and Forensics in the Cloud (TAFC)*, pp. 21–30, 2013, ISSN: 2079-2247. [Online]. Available: <http://dimacs.rutgers.edu/archive/Workshops/TAFC/TAFC-preproceedings.pdf#page=21>.
- [3] A. R. Hevner, S. M. March, J. Park, and S. Ram, “Design Science in Information Systems Research”, *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004, ISSN: 02767783. DOI: 10.2307/25148625.
- [4] O. Scekic, D. Miorandi, T. Schiavinotto, D. I. Diochnos, A. Hume, R. Chenu-Abente, H. L. Truong, M. Rovatsos, I. Carreras, S. Dustdar, and F. Giunchiglia, “SmartSociety - A Platform for Collaborative People-Machine Computation”, *Proceedings - 2015 IEEE 8th International Conference on Service-Oriented Computing and Applications, SOCA 2015*, pp. 147–154, 2016. DOI: 10.1109/SOCA.2015.10.
- [5] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008. [Online]. Available: <https://bitcoin.org/bitcoin.pdf> (accessed on May 16, 2020).
- [6] A. M. Antonopoulos, *Mastering Bitcoin: Programming the Open Blockchain*, 2nd. O’Reilly Media, Inc., 2017, p. 408, ISBN: 978-1491954386.
- [7] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, “SoK: Research perspectives and challenges for bitcoin and cryptocurrencies”, in *Proceedings - IEEE Symposium on Security and Privacy*, vol. 2015-July, IEEE, 2015, pp. 104–121, ISBN: 9781467369497. DOI: 10.1109/SP.2015.14.

- [8] D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. J. Sussman, “Information accountability”, *Communications of the ACM*, vol. 51, no. 6, pp. 82–87, 2008, ISSN: 00010782. DOI: 10.1145/1349026.1349043.
- [9] R. Neisse, G. Steri, and I. Nai-Fovino, “A Blockchain-based Approach for Data Accountability and Provenance Tracking”, in *ARES '17 Proceedings of the 12th International Conference on Availability, Reliability and Security*, Reggio Calabria, Italy: ACM, 2017, 14:1–14:10, ISBN: 9781450352574. DOI: 10.1145/3098954.3098958. arXiv: 1706.04507.
- [10] M. Martin and A. Schreiber, “Trustworthy Provenance Recording using a blockchain-like database Master’s Thesis”, PhD thesis, Leipzig University, 2017. [Online]. Available: <http://elib.dlr.de/111772/1/thesis.pdf>.
- [11] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, “MedRec: Using blockchain for medical data access and permission management”, *Proceedings - 2016 2nd International Conference on Open and Big Data, OBD 2016*, pp. 25–30, 2016. DOI: 10.1109/OBD.2016.11.
- [12] S. Dustdar, S. Nastic, and O. Scekic, “A novel vision of cyber-human smart city”, *Proceedings - 4th IEEE Workshop on Hot Topics in Web Systems and Technologies, HotWeb 2016*, pp. 42–47, 2016. DOI: 10.1109/HotWeb.2016.16.
- [13] A. Coronato, V. De Florio, M. Bakhouya, and G. Di Marzo Serugendo, “Formal modeling of socio-technical collective adaptive systems”, *Proceedings - 2012 IEEE 6th International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2012*, pp. 187–192, 2012, ISSN: 1949-3673. DOI: 10.1109/SASOW.2012.40.
- [14] S. Anderson, N. Bredeche, A. Eiben, G. Kampis, and M. van Steen, *Adaptive Collective Systems – Hearing Black Sheep*. Bookprints, 2013, ISBN: 9782746669307.
- [15] S. Kernbach, T. Schmickl, and J. Timmis, “Collective Adaptive Systems: Challenges Beyond Evolvability”, 2011. arXiv: 1108.5643.
- [16] L. Moreau, “The Foundations for Provenance on the Web”, *Foundations and Trends® in Web Science*, vol. 2, no. 2-3, pp. 99–241, 2010, ISSN: 1555-077X. DOI: 10.1561/18000000010.
- [17] A. R. Yumerefendi and J. S. Chase, “The Role of Accountability in Dependable Distributed Systems”, in *Proceedings of the First Conference on Hot Topics in System Dependability*, Yokohama, Japan: USENIX Association, 2005, p. 3.
- [18] K. J. Lin, J. Zou, and W. Yan, “Accountability Computing for E-society”, *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pp. 34–41, 2010, ISSN: 1550-445X. DOI: 10.1109/AINA.2010.167.

- [19] J. Freire, D. Koop, E. Santos, and C. T. Silva, “Provenance for computational tasks: A survey”, *Computing in Science and Engineering*, vol. 10, no. 3, pp. 11–21, 2008, ISSN: 15219615. DOI: 10.1109/MCSE.2008.79.
- [20] Y. L. Simmhan, B. Plale, and D. Gannon, “A survey of data provenance in e-science”, *ACM SIGMOD Record*, vol. 34, no. 3, pp. 31–36, 2005, ISSN: 0163-5808. DOI: 10.1145/1084805.1084812.
- [21] L. Moreau and P. Missier, *PROV-DM: The PROV Data Model*, 2013. [Online]. Available: <http://www.w3.org/TR/2013/REC-prov-dm-20130430/> (accessed on May 16, 2020).
- [22] W.-C. Tan, “Research problems in data provenance”, *IEEE Data Engineering Bulletin*, vol. 27, pp. 45–52, 2004.
- [23] S. M. S. Da Cruz, M. L. M. Campos, and M. Mattoso, “Towards a taxonomy of provenance in Scientific Workflow Management Systems”, *SERVICES 2009 - 5th 2009 World Congress on Services*, no. PART 1, pp. 259–266, 2009. DOI: 10.1109/SERVICES-I.2009.18.
- [24] Y. S. Tan, R. K. Ko, and G. Holmes, “Security and data accountability in distributed systems: A provenance survey”, *Proceedings - 2013 IEEE International Conference on High Performance Computing and Communications, HPCC 2013 and 2013 IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2013*, pp. 1571–1578, 2013. DOI: 10.1109/HPCC.and.EUC.2013.221.
- [25] P. Groth and L. Moreau, *PROV-Overview*, 2013. [Online]. Available: <https://www.w3.org/TR/prov-overview/> (accessed on May 16, 2020).
- [26] L. Moreau and P. Groth, “Provenance: An Introduction to PROV”, *Synthesis Lectures on the Semantic Web: Theory and Technology*, vol. 3, no. 4, pp. 1–129, 2013, ISSN: 2160-4711. DOI: 10.2200/S00528ED1V01Y201308WEB007.
- [27] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. Van Den Bussche, “The Open Provenance Model core specification (v1.1)”, *Future Generation Computer Systems*, vol. 27, pp. 743–756, 2011. DOI: 10.1016/j.future.2010.07.005.
- [28] P. Missier, K. Belhajjame, and J. Cheney, “The W3C PROV family of specifications for modelling provenance metadata”, *Proceedings of the 16th International Conference on Extending Database Technology - EDBT '13*, p. 773, 2013. DOI: 10.1145/2452376.2452478. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=2452376.2452478>.

- [29] T. Lebo, S. Sahoo, and D McGuinness, *PROV-O: The PROV Ontology*, 2013. [Online]. Available: <http://www.w3.org/TR/2013/REC-prov-o-20130430/> (accessed on May 16, 2020).
- [30] J. Cheney, P. Missier, and L. Moreau, *Constraints of the PROV Data Model*, 2013. [Online]. Available: <https://www.w3.org/TR/2013/REC-prov-constraints-20130430/> (accessed on May 16, 2020).
- [31] L. Moreau and P. Missier, *PROV-N: The Provenance Notation*, 2013. [Online]. Available: <https://www.w3.org/TR/2013/REC-prov-n-20130430/> (accessed on May 16, 2020).
- [32] G. Klyne and P. Groth, *PROV-AQ: Provenance Access and Query*, 2013. [Online]. Available: <https://www.w3.org/TR/2013/NOTE-prov-aq-20130430/> (accessed on May 16, 2020).
- [33] K. Belhajjame, H. Deus, D. Garijo, G. Klyne, P. Missier, S. Soiland-Reyes, and S. Zednik, *PROV Model Primer*, 2013. [Online]. Available: <https://www.w3.org/TR/2013/NOTE-prov-primer-20130430/> (accessed on May 16, 2020).
- [34] L. Moreau, H. Hua, C. Tilmes, and S. Zednik, *PROV-XML: The PROV XML Schema*, 2013. [Online]. Available: <http://www.w3.org/TR/2013/NOTE-prov-xml-20130430/> (accessed on May 16, 2020).
- [35] D. Chaum, “Blind Signatures for Untraceable Payments”, in *Advances in Cryptology*, Boston, MA: Springer US, 1983, pp. 199–203, ISBN: 978-1-4757-0604-8, 978-1-4757-0602-4. DOI: 10.1007/978-1-4757-0602-4_18.
- [36] L. Law, S. Sabetr, and J. Solinas, “How to Make a Mint: The Cryptography of Anonymous Electronic Cash”, *American University Law Review*, vol. 46, no. 4, pp. 1131–1162, 1996.
- [37] N. Szabo, “Formalizing and securing relationships on public networks”, *First Monday*, vol. 2, no. 9, 1997, ISSN: 13960466. DOI: 10.5210/fm.v2i9.548.
- [38] —, *Trusted Third Parties are Security Holes*, 2001. [Online]. Available: <http://nakamotoinstitute.org/trusted-third-parties/> (accessed on May 16, 2020).
- [39] D. Tapscott and A. Tapscott, *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World*. Portfolio, 2016, p. 384, ISBN: 978-1101980132.

- [40] Economist, “The promise of the blockchain | The trust machine”, *The Economist*, pp. 2–4, 2015, ISSN: 0013-0613. [Online]. Available: <https://www.economist.com/leaders/2015/10/31/the-trust-machine> (accessed on May 16, 2020).
- [41] A. Narayanan and J. Clark, “Bitcoin’s academic pedigree”, *Communications of the ACM*, vol. 60, no. 12, pp. 36–45, 2017, ISSN: 00010782. DOI: 10.1145/3132259.
- [42] F. Tschorsch, “Bitcoin and Beyond : A Technical Survey on Decentralized Digital Currencies”, vol. 18, no. 3, pp. 2084–2123, 2016, ISSN: 1553-877X. DOI: doi:10.1109/COMST.2016.2535718.
- [43] I. Osipkov, E. Y. Vasserman, N. Hopper, and K. Yongdae, “Combating double-spending using cooperative P2P systems”, *Proceedings - International Conference on Distributed Computing Systems*, 2007, ISSN: 1063-6927. DOI: 10.1109/ICDCS.2007.91.
- [44] A. Deshpande, K. Start, L. Lepetit, and S. Gunashekar, “Distributed Ledger Technologies/Blockchain: Challenges, opportunities and the prospects for standards”, *Overview report The British Standards Institution (BSI)*, no. May, pp. 1–34, 2017. [Online]. Available: https://www.bsigroup.com/LocalFiles/zh-tw/InfoSec-newsletter/No201706/download/BSI_Blockchain_DLT_Web.pdf (accessed on May 16, 2020).
- [45] M. Walport *et al.*, “Distributed ledger technology: Beyond block chain”, *UK Government Office for Science*, pp. 1–88, 2016. [Online]. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf (accessed on May 16, 2020).
- [46] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis, “Consensus in the Age of Blockchains”, 2017. arXiv: 1711.03936.
- [47] D. Conte de Leon, A. Q. Stalick, A. A. Jillepalli, M. A. Haney, and F. T. Sheldon, “Blockchain: properties and misconceptions”, *Asia Pacific Journal of Innovation and Entrepreneurship*, vol. 11, no. 3, pp. 286–300, 2017, ISSN: 2071-1395. DOI: 10.1108/APJIE-12-2017-034.
- [48] E. Politou, F. Casino, E. Alepis, and C. Patsakis, “Blockchain Mutability: Challenges and Proposed Solutions”, *IEEE Transactions on Emerging Topics in Computing*, 2019, ISSN: 21686750. DOI: 10.1109/TETC.2019.2949510. arXiv: 1907.07099.

- [49] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends”, in *2017 IEEE International Congress on Big Data (BigData Congress)*, 2017, pp. 557–564, ISBN: 9781538619964.
DOI: 10.1109/BigDataCongress.2017.85.
- [50] D. T. T. Anh, M. Zhang, B. C. Ooi, and G. Chen, “Untangling Blockchain: A Data Processing View of Blockchain Systems”, *IEEE Transactions on Knowledge and Data Engineering*, vol. 4347, no. c, pp. 1–20, 2018, ISSN: 10414347.
DOI: 10.1109/TKDE.2017.2781227. arXiv: 1708.05665.
- [51] V. Buterin, *On Public and Private Blockchains*, 2015.
[Online]. Available: <https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/> (accessed on May 16, 2020).
- [52] EWF, *Energy Web Foundation – blockchain and digital security in energy*, 2017.
[Online]. Available: <http://energyweb.org/> (accessed on May 16, 2020).
- [53] J. Kolla, *Decoding the evolution of Blockchain 3.0*, New Delhi, 2018.
[Online]. Available: <https://www.livemint.com/Technology/OIb3LaLJ2pdwAGMRCiGLhI/Decoding-the-evolution-of-Blockchain-30.html> (accessed on May 16, 2020).
- [54] S. Lee, *Explaining Directed Acyclic Graph (DAG), The Real Blockchain 3.0*, 2018.
[Online]. Available: <https://www.forbes.com/sites/shermanlee/2018/01/22/explaining-directed-acyclic-graph-dag-the-real-blockchain-3-0/> (accessed on May 16, 2020).
- [55] S. Haber and W. S. Stornetta, “How to time-stamp a digital document”, *Journal of Cryptology*, vol. 3, no. 2, pp. 99–111, 1991, ISSN: 09332790.
DOI: 10.1007/BF00196791. arXiv: arXiv:1211.2549v2.
- [56] D. Bayer, S. Haber, and W. S. Stornetta, “Improving the Efficiency and Reliability of Digital Time-Stamping”, in *Sequences II*, 1993, pp. 329–334, ISBN: 978-1-4613-9323-8.
DOI: 10.1007/978-1-4613-9323-8_24.
- [57] L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem”, *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982. DOI: 10.1145/357172.357176.
- [58] L. S. Sankar, M. Sindhu, and M. Sethumadhavan, “Survey of consensus protocols on blockchain applications”, *2017 4th International Conference on Advanced Computing and Communication Systems, ICACCS 2017*, 2017.
DOI: 10.1109/ICACCS.2017.8014672.

- [59] C. Lemahieu, *Nano: A Feeless Distributed Cryptocurrency Network*, 2017. [Online]. Available: https://content.nano.org/whitepaper/Nano_Whitepaper_en.pdf (accessed on May 16, 2020).
- [60] SteemIt, “Steem Bluepaper”, 2017. [Online]. Available: <https://steem.io/steem-bluepaper.pdf> (accessed on May 16, 2020).
- [61] D. Larimer, *DPOS Consensus Algorithm - The Missing White Paper*, 2017. [Online]. Available: <https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper> (accessed on May 16, 2020).
- [62] V. Buterin, *Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform*, 2014. [Online]. Available: <https://github.com/ethereum/wiki/wiki/White-Paper> (accessed on May 16, 2020).
- [63] A. M. Antonopoulos and G. Wood, *Mastering Ethereum: Building Smart Contracts and DApps*. O’reilly Media, 2018, p. 385, ISBN: 9781491971949.
- [64] S. Popov, *The Tangle*, 2018. [Online]. Available: https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf (accessed on May 16, 2020).
- [65] D. Schiener, *A Primer on IOTA (with Presentation) – IOTA*, 2017. [Online]. Available: <https://blog.iota.org/a-primer-on-iota-with-presentation-e0a6eb2cc621> (accessed on May 16, 2020).
- [66] A. Back, *Hashcash -A Denial of Service Counter-Measure*, 2002. [Online]. Available: <http://hashcash.org/hashcash.pdf> (accessed on Mar. 26, 2020).
- [67] S. Popov, H. Moog, D. Camargo, A. Caposelle, V. Dimitrov, A. Gal, A. Greve, B. Kusmierz, S. Mueller, A. Penzkofer, O. Saa, W. Sanders, L. Vigneri, W. Welz, and V. Attias, *The Coordicide*, 2020. [Online]. Available: https://files.iota.org/papers/20200120_Coordicide_WP.pdf (accessed on Apr. 5, 2020).
- [68] Iota, *IOTA Coordicide*. [Online]. Available: <https://coordicide.iota.org/> (accessed on Apr. 5, 2020).
- [69] E. Heilman, N. Narula, G. Tanzer, J. Lovejoy, M. Colavita, M. Virza, and T. Dryja, *Cryptanalysis of Curl-P and Other Attacks on the IOTA Cryptocurrency*, Cryptology ePrint Archive, Report 2019/344, 2019. [Online]. Available: <https://eprint.iacr.org/2019/344> (accessed on Mar. 26, 2020).

- [70] P. Zeppezauer, O. Scekcic, H. L. Truong, and S. Dustdar, “Virtualizing communication for hybrid and diversity-aware collective adaptive systems”, PhD thesis, 2015, pp. 56–67, ISBN: 9783319228846.
DOI: 10.1007/978-3-319-22885-3_6.
- [71] O. Scekcic, T. Schiavinotto, D. I. Diochnos, M. Rovatsos, H. L. Truong, I. Carreras, and S. Dustdar, “Programming Model Elements for Hybrid Collaborative Adaptive Systems”, *Proceedings - 2015 IEEE Conference on Collaboration and Internet Computing, CIC 2015*, pp. 278–287, 2016, ISSN: 2168-6750.
DOI: 10.1109/CIC.2015.17.
- [72] S. Paradkar,
Mastering Non-Functional Requirements : Analysis, architecture, and assessment. Packt Publishing Ltd., 2017, p. 230, ISBN: 978-1788299237.
- [73] A. Cockburn, *Writing effective use cases*. Addison-Wesley Professional, 2000, p. 304, ISBN: 978-0201702255.
- [74] Object Management Group,
Unified Modeling Language: Specification. Version 2.5.1. formal/17-12-05, 2017. [Online]. Available: <https://www.omg.org/spec/UML/2.5.1/> (accessed on Mar. 30, 2019).