

A Blockchain-driven Approach for Secure and Scalable Provenance Management in Open Data Systems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Stoyan Staynov, BSc.

Registration Number 11815146

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr. Emanuel Sallinger

Assistance: DI Markus Nissl

Vienna, 3rd May, 2023



Stoyan Staynov



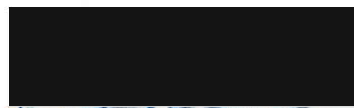
Emanuel Sallinger

Erklärung zur Verfassung der Arbeit

Stoyan Staynov, BSc.

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

Wien, 23. April 2023



Stoyan Staynov

Acknowledgements

I would like to seize this moment to convey my appreciation to those people who have been with me and supported me throughout my academic journey and the creation of this thesis.

I am particularly grateful to my advisors, Markus Nissl and Emanuel Sallinger, for always being accessible to assist me whenever I encountered difficulties in my work (sometimes on extremely short notice), for the invaluable feedback, and excellent communication while writing this diploma thesis.

Abstract

As the availability of Linked Open Data (LOD) expands and leads to the creation of more derived and aggregated data, the demand for increasing the trust factor associated with this data has grown. Provenance tracking has shown to be a feasible approach to increasing reliability of data, however, a uniform approach for handling provenance data of LOD at a global level has not yet been established.

This presents a challenging issue as data provenance frequently possesses numerous domain-specific characteristics. This further enforces the limited interoperability of current solutions across different data management systems. In order to address these problems and achieve a unified and interoperable solution for provenance tracking, it is essential to develop an architecture that can cater to the varied needs of different domains and data catalogs while maintaining a high degree of adaptability and scalability.

In recent years, decentralized networks have gained popularity with the emergence of blockchain and distributed ledger technologies. These technologies offer a promising approach to addressing the challenges associated with provenance tracking by providing a secure and tamper-resistant platform for storing and managing provenance information.

In this thesis, we propose a platform capable of multi-domain support that leverages state-of-the-art research regarding blockchain-based storage and tamper-resistant management of provenance information. The solution can be used with different existing data management systems for LOD, regardless of their underlying technology or implementation. In this way we address current limitations in the field of provenance tracking and enhance data quality and trustworthiness across various data management and analytical use cases. A key emphasis is placed on data systems powered by knowledge graphs, which possess the capability to make inferences and offer a more in-depth comprehension of the data.

Our approach has been evaluated using a scenario-based framework, revealing its ability to support high-level data granularity as well as addressing fundamental requirements for the correct operation of such a system. Additionally, we have presented an in-depth analysis of the expenses associated with running our proposed solution, considering both on-chain and off-chain aspects. Through this analysis, we have effectively illustrated the versatility of our approach across various domains and scenarios, outperforming many existing state-of-the-art methods.

Contents

Abstract	vii
Contents	ix
1 Introduction	1
1.1 Motivation and Problem Statement	1
1.2 Aim and Expected Results	2
1.3 Research Questions and Main Results	3
1.4 Methodology	5
1.5 Structure	6
2 Preliminaries	9
2.1 Graph Networks	9
2.2 Data Provenance	14
2.3 Blockchain	23
2.4 Data Catalogs	28
2.5 Related Work	30
3 Design	39
3.1 Requirements	39
3.2 Architecture	43
3.3 Summary	48
4 Implementation	49
4.1 Technology Stack	49
4.2 Smart Contract	54
4.3 Middleware Agents	62
5 Evaluation	69
5.1 Requirements Analysis	69
5.2 Cost Efficiency	74
6 Conclusion and Future Work	81
6.1 Research Questions	82
	ix

6.2 Future Work	83
List of Figures	85
List of Tables	87
Bibliography	91

Introduction

1.1 Motivation and Problem Statement

The exponential growth in data in enterprises and institutions [AN19] has created a need for effective and efficient data management that allow for tracking provenance information. In detail, *provenance tracking* is the process of backtracking the origin of that data as far as practicable. One aspect of what is usually referred to as *provenance information* is semantic linking to a data source [Cag19]. Such information includes details about where the data came from, who created it, and any transformations that have been applied to it.

To tackle the issue of provenance tracking, the use of *data catalogs* has become a widespread practice in order to organize and manage large amounts of data efficiently [ESM⁺21]. Yet such catalogs miss the ability to ensure trustworthiness of the cataloged data [HC20].

For this, the integration of a *knowledge graph* (KG) into data catalogs can significantly enhance the catalogs' ability to draw inferences and provide a more comprehensive understanding of the data [Ran18]. In a broad sense, KGs acquire and integrate information into an ontology and apply a (scalable) reasoner to derive new knowledge [EW16]. In the setting of provenance tracking, this means that with the reasoner we are able to derive new insights and relationships between data resources that may not be immediately apparent through manual exploration.

The benefits of (KG-powered) data catalogs are especially notable when dealing with *open data*, where data access (and sometimes editorial rights) is provided to the broad public. However, the increasing complexity and volume of public data pose new challenges in terms of legitimacy. There is evidence supporting the notion that using systems, capable of tracking provenance of data resources can enhance the trustworthiness of the data [LST⁺17].

To ensure the security and trust of provenance information, it is essential that it is stored in a tamper-resistant manner [ZKS⁺17]. Integrating data provenance solutions with smart contracts, computer programs that allow users to deploy and execute procedural instructions, can enhance the validity and reliability of data [LST⁺17, RK18].

Across literature, various implementations of storing provenance for organized data in smart contracts exist [KH21, DA20, RK18, LST⁺17]. However, to the best of our knowledge, current work share at least one of the following limitations:

- solve a domain-specific problem;
- utilize a specific platform in terms of blockchain and/or data catalog;
- have limited interoperability with existing data management systems;

These limitations leave a gap to be filled by a platform capable of *multi-domain* support, creating the dire need of a global solution supporting state-of-the-art research regarding blockchain-based storage and tamper-resistant management of provenance information, as well as the integration of Knowledge Graphs for improved data discovery, navigation, and quality. Such a solution can be used with different existing data catalogs, regardless of their underlying technology or implementation, and can help address the current limitations in the field of provenance tracking.

A potential way to overcome this limitation is to integrate state-of-the-art knowledge graph systems with smart contracts. Smart contracts provide the abilities to store and manage provenance information in a tamper-resistant and transparent manner. Knowledge graphs, on the other hand, provide a semantic representation of data, which can support data integration and interoperability, and enable automated reasoning. By combining the strengths of smart contracts and knowledge graphs, it is possible to develop a solution that can efficiently store and manage provenance information, enhance data quality and trustworthiness, and support a wide range of data management and analytical use cases.

1.2 Aim and Expected Results

Considering the limitations found in existing approaches to provenance tracking, there is a pressing need to address the challenges in ensuring data trustworthiness, interoperability, and domain-agnostic support. To this end, the integration of state-of-the-art knowledge graph systems with smart contracts is explored in the subsequent section, with the aim of developing a comprehensive solution for managing provenance information across various domains and data catalogs.

As outlined in the problem statement, most current solutions for storing and managing provenance information of cataloged open data in data catalogs share the drawback that they solve a domain-specific problem. In order to address this limitation, our

research aims to explore how we can extend these solutions to overcome this drawback. Key aspects of this investigation include the role of Smart Contracts in providing this functionality, the automation of the process, and the potential for further extending the contract to integrate additional domain knowledge.

Our initial aim is to extend existing solutions in collaborative Linked Open Data to support sufficient provenance information. By doing so, we anticipate enhanced data quality, trustworthiness, and interoperability across diverse LOD sources. The adoption of standardized vocabularies, best practices, and provenance-aware applications will enable users to better understand the origin and history of resources, encouraging a more transparent and reliable ecosystem. Bearing these considerations in mind, we also seek to address the platform-specific constraints of current approaches.

An additional shared drawback to current solutions is the utilization of a specific platform in terms of used blockchain. Depending on the consensus mechanism enforced by the blockchain, the solution could have a significant impact on energy consumption, leading not only to significant operational expenses but also large environmental footprint.

As a result of this, we aim to explore different authority and consensus mechanisms for a blockchain-based system storing provenance information of LOD, we expect to identify solutions that maintain data integrity while minimizing energy consumption. Alternative consensus algorithms, such as Proof of Stake, Delegated Proof of Stake, or Practical Byzantine Fault Tolerance, may offer a more energy and/or cost efficient approach compared to traditional Proof of Work, addressing ethical concerns regarding resource usage.

Finally, weaknesses have been identified in current solutions in terms of interoperability with existing systems, accommodating the growing volume of data, and compatibility with various data catalogs. In our last objective our goal is to determine the most optimal standards for our solution in terms of provenance vocabularies, smart contract standards, data models and overall best practices and guidelines.

The aim of this thesis is used to formulate the context of our research questions, presented in the next section. Additionally, specific expected results are formulated following each question.

1.3 Research Questions and Main Results

The research direction and expected results in the next section are defined in the form of Research Questions (**RQs**) and Sub Research Question (**SRQs**), which encapsulate the details tackled in the fragments of the topic.

In the thesis, we are going to address three important research questions. First, we focus on overcoming the drawback of existing solutions when dealing with domain-independent data.

RQ 1. How can we extend existing solutions in collaborative Linked Open Data to support sufficient provenance information as means of preserving provenance information in a globally unique way?

- **SRQ 1.1.** What role can Smart Contracts play in order to provide this functionality?
- **SRQ 1.2.** How can this process be automated?
- **SRQ 1.3.** How can we further extend the contract and integrate additional domain knowledge?

To address this question, we develop a versatile multi-domain solution that utilizes the blockchain for storing and managing provenance information securely. Our architecture prioritizes extensibility and adaptability, enabling easy integration with modern data catalogs through an API-based approach. Additionally, our solution also focuses on enabling automation where possible, with the aim to reduce administrative overhead. However, automatic smart contract deployment remains a challenge. By utilizing modern standards, we were able to maintain contract extensibility, allowing for incorporation of domain-independent knowledge.

Then, we explore how can we improve operational expenses, while preserving a minimal environmental footprint, a consideration often overlooked by modern solutions.

RQ 2. Which authority and consensus mechanisms are available to ensure the integrity of such a system? How can we preserve ethical considerations w.r.t energy consumption in such a solution?

To tackle this question, we conduct a detailed study on authority and consensus mechanism, carefully evaluating each one for its suitability in our solution. Ultimately, the Proof of Stake (PoS) mechanism was chosen for our solution, as it provides for an optimal balance between the security and integrity demands of our solution, while providing for a reduced environmental footprint.

Lastly, we aim to overcome the weaknesses of current solutions when addressing monetary costs and interoperability with existing systems.

RQ 3. What standards and tools are suitable to implement a universal solution, expandable to multiple domains with considerations of usability, scalability and flexibility?

- **SRQ 3.1.** How does the cost factor vary in each of them?
- **SRQ 3.2.** How does the storage factor vary in each of them? Benefits w.r.t on/off/mix-chain provenance storage?

By combining on-chain and off-chain provenance storage, we have developed a system capable of serving different industries, further justified by the data model and technology stack utilized by our solution. We have further enforced our claims, by providing a detailed evaluation on core properties together with cost factors of our proposed solution.

1.4 Methodology

In this work, we employ Design Science as a methodology for conducting research and developing artifacts in order to address the identified problem and achieve the stated objectives.

1. **Literature review** By conducting a literature review we are able to identify the key concepts, trends and most importantly gaps in current state-of-the-art on cataloged open data, blockchain, and provenance information. We conduct a detailed evaluation and interpretation on the research information presented. Particular effort is put in identifying any biases or limitations that may affect the credibility or validity of the information, especially in the context of blockchain systems.

This allows us to attain greater insight on these topics and provide an optimal direction for answering the research questions. Furthermore, we are able to gain a better understanding how the technological concepts employed by our approach can be combined.

2. **Design & Implementation** In this methodology we employ a structured approach to designing and implementing a proof-of-concept in order to demonstrate the effectiveness of the solution.

During the design phase, an architecture blueprint of a system for efficient storage and management of provenance information for cataloged open data in a tamper-resistant manner is established. The main goal is to discover an appropriate solution for recording, coordinating, and gathering provenance information within the blockchain, while also enabling authorized access for utilization of this information.

Subsequently, during the implementation phase, a solution is devised, drawing upon the findings of the preceding research and architectural design. This phase also includes a detailed explanation of multiple technological choices made and the rationale behind them.

3. **Evaluation** Lastly, the evaluation of this work is conducted by employing two techniques, namely (1) conducting multiple experiments by using the proof of concept (PoC) implementation and (2) a comparison with the current state-of-the-art.

The experimental design of this study is intended to address research questions which have not been sufficiently answered by the conducted research. The experimental

framework relies on quantitative data retrieved from various components of the proposed system. Specifically, one of the key components we leverage for data retrieval is the blockchain, due to its ability to capture and store a vast amount of information that is well-suited for detailed analysis. Subsequently, those experiments are evaluated, and the findings are interpreted.

The comparative analysis in this study is based on multiple use-cases, which are identified during the requirements definition of the thesis. The use cases are chosen based on an evaluation framework and performance of several different systems is evaluated against each other.

1.5 Structure

The remaining thesis is structured as follows:

Chapter 2: Preliminaries

The second chapter provides information regarding the meaning and semantics of the primary technologies referenced in this work. A specific provenance model, identified as most suitable for the means of this thesis, is presented and explored in detail. Furthermore, extra emphasis is put on distributed ledgers and smart contracts as essential solutions to answering parts of the research challenges. Lastly, literature in the fields mentioned above are examined. In addition, sources of usage and implementation of platforms aiming to achieve provenance tracking on distributed ledgers are covered. Considerable attention is paid to system utilizations and techniques w.r.t. choices of Blockchain systems, provenance models and argumentation around those decisions.

Chapter 3: Design

The content provided in the design chapter aims to present a structural blueprint for the system, which directs future implementation efforts. This blueprint includes information about identified requirements that the solution must fulfill, along with an in-depth exploration of the architectural design.

Chapter 4: Implementation

In the implementation chapter, details on the development of a proof-of-concept implementation of the proposed solution are provided. Furthermore, aspects such as the implementation procedure, the selected technology stack, and the resulting outcomes are covered.

Chapter 5: Evaluation

In the evaluation chapter, an informed discussion is conducted about how the solution addresses general and quality-related system properties within the data provenance

domain. Moreover, a scenario-based evaluation is provided to showcase the capabilities of the solution compared to the current state-of-the-art. Finally, an analysis of the specific costs associated with using this approach for storing provenance information on the blockchain is presented.

Chapter 6: Conclusion and Future Work

The last chapter provides a summary of the findings and offers a perspective on potential future research directions to enhance the solution.

Preliminaries

In this chapter, we describe the necessary definitions of concepts that are encountered throughout this thesis. We assume that the reader is familiar with the basic idea behind the World Wide Web (WWW) [BCL⁺94]. The rest of this chapter is organized as follows.

First, in Section 2.1, we explore the field of graph networks, more concretely Linked Data and knowledge graphs. In Section 2.2, we introduce the concept of data provenance. We highlight the key types of provenance and extend the research by presenting specifications and data models in the shape of the widely adopted W3C PROV standard [MBC13]. Then, in Section 2.3, we provide some necessary context regarding distributed ledgers and differences between existing digital representations of such (blockchains). Next, in Section 2.4 we describe characteristics and state-of-the-art in data catalogs. Finally, we explore related work on blockchains, provenance, and Linked Data in Section 2.5.

2.1 Graph Networks

In this section, we provide an overview of graph networks, also called graphs. In Subsection 2.1.1 this concept is explored in the context of Linked (Open) Data, together with characteristics and popular data models. Later in Subsection 2.1.2, we introduce the concept of knowledge graphs. We provide context on what benefits knowledge provides in the scope of graphs and explore the building blocks of such a structure.

Graph networks are mathematical structures used to represent relationships between pairs of objects [net21]. These structures date back to the 18th century and have many applications in computing and information technologies. A graph consists of vertices (in our domain also known as entities, nodes or points) connected by edges (also known as relationships, links or lines). There are two types of graphs: undirected graphs, in which the edges symmetrically connect two vertices, and directed graphs, in which the edges asymmetrically connect two vertices.

1. Use URIs as names for things.

Example 1: URI

An example of a URI is any URL such as the URI for research overview at TU Vienna: <https://www.tuwien.at/en/research>

2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards such as RDF.
4. Include links to other URIs so that they can discover more things.

Figure 2.1: The four principles of Linked Data [BHB09]

2.1.1 Semantic Web & Linked (Open) Data

The Semantic Web, as described by Tim Berners-Lee, is a web of data that can be processed by machines. This enables easy access and navigation between resources in similar contexts. The term is described by Berners-Lee et al. [BLHL01] as:

Definition 2.1.1 (Semantic Web). “an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.”

Linked Data [BL], as originally introduced by the same author, has the aim to create links between multi-source data elements such as entities, concepts, and properties.

In vision of the author, the ultimate objective was to achieve a machine-readable web of data i.e. *Semantic Web*, while *Linked Data* is the means by which this goal can be achieved. Throughout the evolution of the WWW, these two concepts have become interchangeable [Yu14].

Linked Data. Linked Data can take the form of resources or literal values. At the same time, subjects and predicates are represented as resources, each of which can be identified by a unique international resource identifier (IRI). The IRI is an internet protocol standard that expands the Uniform Resource Identifier (URI) protocol by increasing the permitted characters. A URI is a unique sequence of characters that identifies a logical or physical resource on the web. A well-known example of an IRI are Uniform Resource Locators (URL), otherwise known as web addresses.

Linked Data is built around a set of four design principles, summarised in Figure 2.1, where data is linked in the form of resources or literal values. By leveraging HTTP URIs, it allows users to look up names and obtain meaningful information using established standards like RDF. Additionally, Linked Data incorporates connections to other URIs,

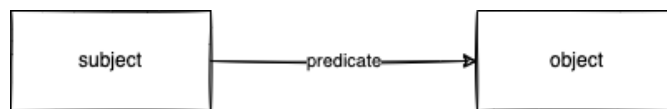


Figure 2.2: An RDF triple statement

enabling the discovery of more related content. In practice, the Resource Description Framework (RDF) data model frequently supports Linked Data.

Resource Description Framework. RDF is a standard for representing and exchanging structured data on the web so that it can be navigated and exchanged between applications without loss of context. At its core, the RDF model represents data as a set of triples, also called facts, where each triple takes the format $\langle subject, predicate, object \rangle$, where *subject* and *object* are the two resources which share a relation. Resources often are referenced throughout multiple triples and share many connections. There is no limit to how many subjects, predicates, and objects are required. However, the minimum requirement for a correct structure is one. Facts are then stored in this format, making them easy to navigate for machines. The structure of an RDF triple statement is shown in [Figure 2.2](#).

Semantic Web Browser. As each RDF entity contains an IRI, this IRI is perfectly fitted for navigating between different resources. For this, specific kinds of browsers have been developed. The Semantic Web browser allows users to navigate the interconnected web of data in similar way that a standard web browser allows users to visit standard text sites through links. However, instead of the *conventional HTML* document format, Semantic Web browsers navigate various data sources using *RDF*, which is heavily reliant on metadata to define the semantics, or meanings, and definitions of data links [\[QK04\]](#). Therefore, Semantic Web browsers are also called hyperdata browsers, analogous to hypertext browsers.

Linked Open Data. A specific branch of link data is Linked Open Data, which uses only open standards, such as RDF. It represents Linked Data, which is publicly open and can be distributed by everybody without restrictions. Linked Open Data is about the open provisioning of (structured) datasets and knowledge on the Web [\[CZA⁺18\]](#). A key idea of Linked Open Data is that additional value is generated by interlinking structured data. According to a study conducted in 2014 [\[SBP14\]](#), the Linked Open Data cloud contained a total of 31 billion RDF triples, interlinked with multiple data sources by approximately 504 million RDF links.

Furthermore, Linked Data *search engines* exist in order to search and find Linked Data on the web. Such search engines use techniques like semantic search, which allows them to understand the meaning and context of the data they are searching for, in order to return more relevant results [\[HHU⁺11\]](#).

Definition	Year	Source
"A mathematical structure with vertices as knowledge units connected by edges that represent the prerequisite relation"	1974	E. Marchi & O. Miguel [MM74]
"[...] systems exist, [...], which use a variety of techniques to extract new knowledge, in the form of facts, from the web. These facts are interrelated, and hence, recently this extracted knowledge has been referred to as a knowledge graph."	2013	Pujara et al. [PMGC13]
"Knowledge graphs could be envisaged as a network of all kind things which are relevant to a specific domain or to an organization. They are not limited to abstract concepts and relations but can also contain instances of things like documents and datasets."	2014	Semantic Web Company [sem14]
"A knowledge graph (i) mainly describes real world entities and their interrelations, organized in a graph, (ii) defines possible classes and relations of entities in a schema, (iii) allows for potentially interrelating arbitrary entities with each other and (iv) covers various topical domains."	2016	Paulheim [Pau17]
"We define a Knowledge Graph as an RDF graph. An RDF graph consists of a set of RDF triples where each RDF triple (s, p, o) is an ordered set of the following RDF terms: a subject $s \in U \cup B$, a predicate $p \in U$, and an object $U \cup B \cup L$. An RDF term is either a URI $u \in U$, a blank node $b \in B$, or a literal $l \in L$."	2016	Farber et al. [FBMR18]
"A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge."	2016	Ehrlinger et al. [EW16]

Table 2.1: Selected definitions of knowledge graph.

2.1.2 Knowledge Graph

Recently, an increasing interest in knowledge graphs has been observed in the enterprise domain and can be found in media, healthcare, construction, etc. [\[GPVW17\]](#). As knowledge graphs are found in such many industries, most people have already unknowingly used a knowledge graph while performing a specific task. One such case is when using the Google search engine. Adding context to specific search results by providing images, news, and videos is an excellent example of how such systems enrich our interactions.

There have been many differences in coming to a consensus on the precise definition of a knowledge graph. A selected list of definitions is shown in [Table 2.1](#), together with the sources and publishing years of the articles containing them.

In the second to last definition of [Table 2.1](#), Paulheim et al. [\[Pau17\]](#) define a knowledge graph as an RDF graph. However, in latter research [\[HBC⁺21\]](#) it is noted that a specific graph data model, such as RDF, concept graphs, or ontologies is not required to build a KG.

In [\[EW16\]](#), Ehrlinger et al. argue that definitions where a knowledge graph is presented as a synonym of a graph-based structure, as most depicted in [Table 2.1](#), is insufficient, as it does not enforce a minimum set of requirements a KG has to fulfill. Furthermore, it may lead to predicaments when people want to explore the topic further, having the goal of building a KG.

In addition to historical considerations, the authors explore diversity in interpretations and use, before proposing a definition.

Ehrlinger et al. recognize the work of Akerkar and Sajja [\[AS09\]](#), in which a knowledge-based system is characterized as using AI to solve problems. Furthermore, such a system is broken down into two components - a knowledge base and an inference engine. The knowledge base is a dataset that holds formal semantics and can include various types of knowledge, such as *rules, facts, axioms, definitions, statements, and primitives*. [\[DSW06\]](#).

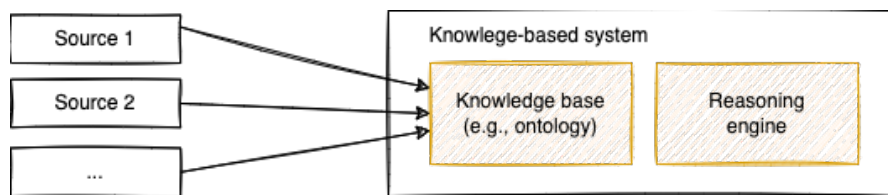


Figure 2.3: Abstract KG architecture

Ontological representations are commonly used as knowledge bases in AI applications, for example, in the context of knowledge-based system, as they allow semantic modeling of knowledge. An **ontology** is as a formal, explicit specification of a shared conceptualization that is characterized by high semantic expressiveness required for increased complexity [FW16]. Using an ontology as a foundation for knowledge allows for the verification of connections between concepts and the drawing of conclusions based on established information through reasoning (inferences). [FW16].

However, when looking at currently existing automatically-generated knowledge graphs, we can see that they also include the ability to gather, extract, and combine information from external sources, making it more than just a basic knowledge-based system. This extends the abstract concept of KG's with an integration system. In [FW16], the authors also mention that *Linked Data* is a common integration aspect in state-of-the-art implementations.

Based on the analysis above, the authors provide a visual representation of a KG architecture, as depicted in Figure 2.4.

In the remainder of the thesis we will follow the definition of a KG, described in Table 2.1 by Ehrlinger et al.

To better understand knowledge graphs, we consider an example structure, based on the format defined in the previous subsection. Even though a knowledge graph is not required to be machine and human-readable, for the sake of the example, we will show a structure that is both.

In Example (2.1.2), we show an informal KG by stating facts using the RDF syntax. Often such an example is represented as a connected graph, which consists of nodes and edges, where the arcs depict the predicates, and the nodes are made up of the subjects and objects. Figure 2.4 shows the graph consisting of the facts in the example.

Example 2: KG representation in RDF

```

<Marcel> <is a> <art enthusiast>.
<Marcel> <is a fan of> <Klimt>.
<Marcel> <was born on> <the 12th of June 1992>.
<Marcel> <admires> <The Kiss>.
<The Kiss> <was created by> <Gustav Klimt>.
  
```

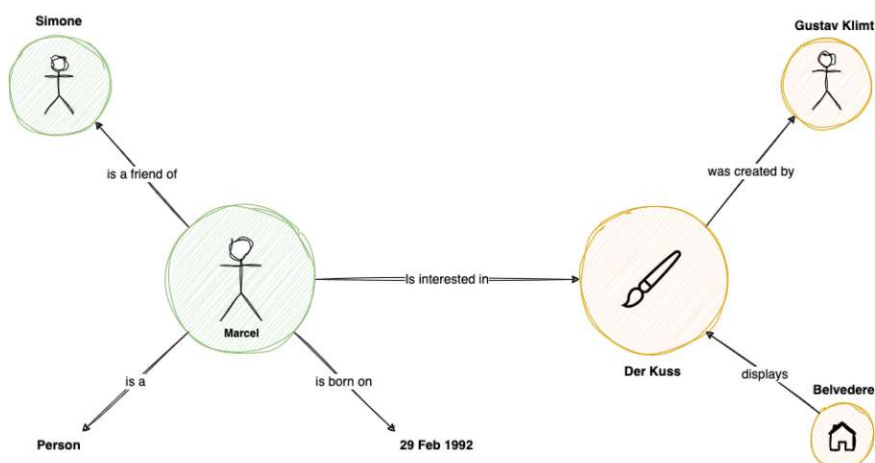


Figure 2.4: Informal graph of the sample triples

Overall, the benefits of knowledge graphs can be summed up as follows:

- **Heterogeneous data** can be collected and used to infer new insights. One trait of a knowledge graph is that it gets richer the more data is added. This is partly because the more information is added, the more inferences can be extracted.
- **Logic-based reasoning** can be performed by representing information as a network of interconnected concepts, each associated with a set of properties and relationships. These connections between concepts can be used to make inferences and deductions based on the information encoded in the knowledge graph.
- **Real-world information** can be modelled and used to organize and represent complex information in a way that is easily understandable and accessible by humans. Knowledge graphs can also be used to integrate data from multiple sources, which can help provide a more complete and accurate picture of a given subject, such as the relationship between diseases, symptoms, and treatments in the field of medicine. Often such graphs are referred to as domain-specific.

2.2 Data Provenance

This section provides an overview of the relevant concepts of data provenance to this thesis. In [Subsection 2.2.1](#) we explore a popular model for provenance information with emphasis on the structure and security considerations. Later, in [Subsection 2.2.2](#), different types of provenance systems, identified by present-day research, are explored.

Provenance - sometimes called “*lineage*” or “*pedigree*” - is most often used to describe the ancestry of an item from a particular source to a specific state of an object. For a

piece of art, provenance tracking usually refers to and establishes its chain of ownership. Alternatively, one can derive more precisely the condition of such a piece by reviewing the restorations it underwent. Referring to the record of such a derivation is the leading property of data provenance.

Definition 2.2.1 (Provenance of a piece of data [GSM+06]). The provenance of a piece of data is the process that led to that piece of data.

We see that the given definition is rather broad regarding tracking the "process that led to that piece of data", namely the how and why. This leaves room for a more specific interpretation, which is why we adopt the more concrete definition of provenance.

Definition 2.2.2 (Data Provenance [Gup09]). The term "data provenance" refers to a record trail that accounts for the origin of a piece of data (in a database, document, or repository) and explains how and why it got to the present place.

Example 3: Data Provenance

In the context of clinical testing, a significant amount of data is obtained from various sources including public databases. This data may have originally come from research papers or clinical trials, but it has likely undergone some changes before being included in the database. A provenance record is used to track the history and the source of each piece of information, including its experimental origins or the clinical trial it was obtained from.

Several community workshops have been held to understand the capabilities of different provenance systems and the expressiveness capabilities of their provenance representations. One of the first such workshops was "The first provenance challenge", defined by Luc Moreau et al. [MLA+08]. The consensus that followed led to a proposal for the Open Provenance Model, a data model for provenance. At a later edition of this workshop, the third provenance challenge, the aim was narrowed down to evaluating the efficacy of the OPM. The outcome showed that it can be difficult to write provenance queries using SQL, as they often require a lot of complicated relational joins which can be overwhelming for domain scientists who may not have the skills or time to learn and write these queries. In the fourth challenge, the last of the workshop series, using the OPM, a comprehensive example is examined, and unique capabilities have been showcased that can only be achieved through implementing an interoperable provenance solution.

Later, in September 2009, the W3C Provenance Incubator Group was created. This group aimed to "provide a state-of-the-art understanding and develop a roadmap in the area of provenance for Semantic Web technologies, development, and possible standardization", as stated in [proa]. In the following year, W3C incubator groups produced recommendations on whether a standardization effort is worth undertaking and published a final report on the subject in 2010 [prob].

As a continuation of the definition of a provenance model standard, the W3C Provenance Working Group was created in 2011. Taking into account the findings stated by the W3C Provenance Incubator Group, in 2013 by the name of PROV, a final set of recommendations was released. Further details about this recommendation are provided in the following subsection.

One approach to managing data provenance is to explicitly add annotations to data elements to capture the provenance, also known as metadata. Such metadata can be used to track the process of the creation and change of the piece of data. Furthermore, such metadata in machine-readable format can allow for effective reasoning and analysis.

The RDF* framework is an extension to RDF that enables such metadata storage to triples, assigning attributes, or creating relationships not just between two entities, but between triples and entities, or triples and triples [Sch].

2.2.1 W3C PROV Standard

The (W3C) **PROV** standard is a notable expressive specification for encoding provenance information [MBC13]. Developed based on thorough research by the Provenance Incubator Group (PROV-XG), it unifies the efforts of various data provenance communities, including the Linked Data and Semantic Web community. In this subsection, we offer a comprehensive overview on the standard's specifications.

Core Recommendations. The design of PROV emanates from eight broad core recommendations, manifesting on what a provenance framework should support:

1. The core concepts of identifying an object, attributing the object to person or entity, and representing processing steps
2. Accessing provenance-related information expressed in other standards
3. Accessing provenance
4. The provenance of provenance
5. Reproducibility
6. Versioning
7. Representing procedures
8. Representing derivation

Furthermore, the work of PROV-XG is comprised of several documents, providing details on every aspect of the standard. Several of these documents are identified as most relevant to this thesis and explored further in the next paragraphs:

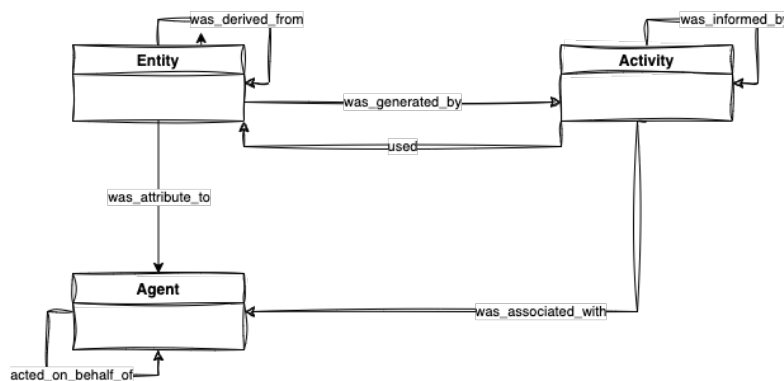


Figure 2.5: PROV Core Structures.

- **PROV-DM** provides the structure of the core data model of the PROV standard.
- **PROV-N** specifies a notation for provenance aimed at human consumption.
- **PROV-O** provides details of the PROV ontology together with details on the mapping of the PROV data model to RDF.

Additionally, in the following sections, we present extensibility and provenance security considerations regarding the W3C PROV Standard.

PROV-DM. At the core of the PROV standard is the conceptual data model recommendation, the **PROV-DM** recommendation. PROV-DM separates the basic elements of provenance information from more specialized elements that are used for specific purposes. This allows for domain and application-specific representations of provenance to be interchanged between systems [MBC13]. Even though the data model is not restricted to a specific field, it has the capability to include domain-specific information through its extensibility points.

At its core, PROV-DM is constructed using three types and seven relations, as depicted in Figure 2.5. The PROV-DM *types* are defined as follows:

1. **Entity:** In our context, an entity describes a certain thing of which we want to track the provenance. It can be physical, digital, conceptual, or any other.
2. **Activity:** An activity can be perceived as an event that occurs over a certain period of time and interacts with entities. Such interaction can be in the form of a broad range of activities, such as processing, consumption, transformation, modification, relocation, or generation of entities.
3. **Agent:** An agent is an entity that holds some level of accountability for an activity, the presence of an entity, or another agent's actions. Three common examples of agents found in literature are a software agent, an organization, and a person.

We further explore the directed edges which represent the PROV-DM *relations* and are defined as follows:

1. **Generation:** Generation is the process of creating an entity by an activity. The entity becomes available after its generation is complete.
2. **Usage:** Usage marks the start of an activity utilizing an entity. Before this process, the specific activity could not have been affected by the entity.
3. **Communication:** Communication represents the exchange of some unspecified entity by two activities, where one of the activities makes use of some entity generated by the other.
4. **Derivation:** Derivation can be linked to the processes of transforming one entity into another, the update of an entity resulting in the creation of a new one, or the construction of a new entity using another based on a pre-existing one.
5. **Attribution:** Attribution is the assignment of an entity to an agent. When an agent is associated with an entity, it means that the entity was created by an unspecified activity that is linked to the agent. This relationship is beneficial when the activity is unknown or unimportant.
6. **Association:** Association refers to the responsibility of an agent for a certain activity.
7. **Delegation:** Delegation is an assignment of responsibility and authority to an agent regarding a specific activity. The delegation can be assigned by another agent or self-assigned. However, it does not break the line of responsibility, as the agent it acts on behalf of retains some responsibility for the outcome of the delegated work.

PROV-N. This document presents a notation standard expressing provenance descriptions in a human-readable format [MM]. Often provenance is illustrated graphically, however, the graphics are not intended to represent the full details in the model. Graphical representations aim to communicate the essence of a set of provenance descriptions.

Example 4: Provenance of a Dataset

In [Figure 2.6](#), a dataset is shown, edited by some editor, while various people (agents) have contributed. The figure is displayed as a graph, where *Entities*, *Activities*, and *Agents* are shown as nodes, whereas edges show how these nodes are related by defining the following two property types: *wasGeneratedBy*, *wasAssociatedWith*.

The PROV Ontology (PROV-O). Above, we have witnessed how the model can be displayed in a human-readable relational syntax (PROV-N). While this representation is nice as a human, for reasoning over such data the PROV-XG has published a document

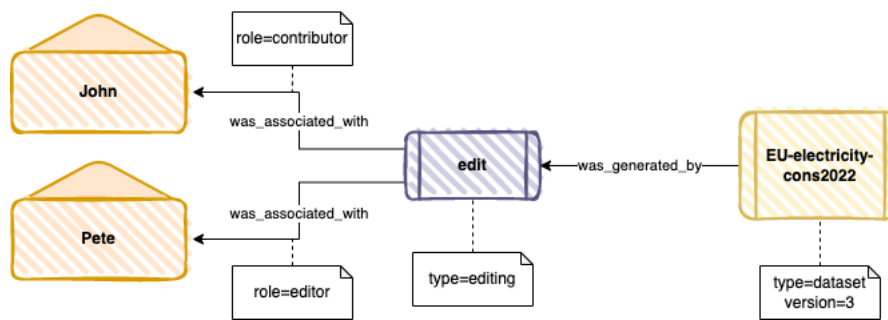


Figure 2.6: Provenance of a Dataset.

recommending how to express the model as an ontology. In essence, PROV-O is an ontology allowing the mapping of the PROV data model to RDF (details on RDF can be found in [Subsection 2.1.1](#)). A set of properties, classes, and restrictions are defined to represent, exchange, and integrate provenance information generated in different systems and contexts. This recommendation document is the essence of what forms the framework for provenance information interchange in domain-specific applications.

A frequent occurrence is that PROV-O users only need to use parts of the entire ontology. This may vary depending on the detail they would like to include in their provenance information and specific needs. To ease users in such situations, the PROV-O terms (*classes* and *properties*) are grouped into three categories: *starting point* terms, *expanded* terms, and terms for *qualifying relationships*.

Starting Point terms bring up the core for the rest of the PROV-O. Using these terms, a user can produce basic provenance descriptions that can be expanded and elaborated on using terms from other categories. As anticipated, the classes and properties in this category closely resemble the PROV data model types and relations discussed previously, with an additional property concerning time. The time properties are particular points in time, used to pinpoint the start and end time of *Activities*.

Expanded terms make additional terms available to describe the provenance among *Entities*, *Activities*, and *Agents*. An abundance of the terms in this category are subclasses or sub properties of those in the *Starting Point Category*. For example, in [Figure 2.7](#), the RDF types “prov:Person”, “prov:Organization”, “prov:SoftwareAgent” are subclasses of *Agent*.

Qualified terms are used to provide comprehensive information regarding the provenance-related influence among *Entities*, *Activities*, and *Agents* in complex systems. What differentiates this category from the previous two is the pattern used to apply the terms effectively. A defining characteristic of the pattern of this category is that the terms are used to provide additional attributes of the relations, whereas previously, the relations were applied more directly. Using this specific pattern enables users to provide comprehensive details which wouldn't be available by using only Starting Point and

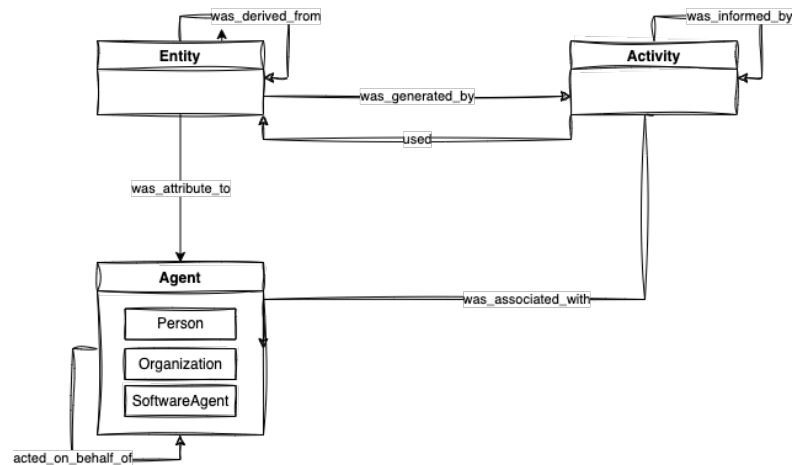


Figure 2.7: PROV Core Structures Expanded.

Expanded terms.^[1]

Extensibility

Because of the domain and technology-independent design of the PROV-DM model, it enables for extensibility to accommodate the requirements of domains specific applications [MBC13]. This allows for increased adaptability across different systems by supporting: *subtyping*, *expanding relations*, *optional identification*, *extensibility points*, and *expansion definition*.

- **Subtyping:** Enables users to easily create domain-specific versions of the core types.
- **Expanding Relations:** Extends the core relations so that they can be used to express n-ary relations.
- **Optional Identification:** Enables a way to distinguish between two different occurrences of the same relation by introducing ID's.
- **Extensibility Points:** Devised using designers in mind and allow them to adapt the model for specific applications or business domains. Three reserved attributes are defined in this concept: “prov:type”, “prov:role”, and “prov:location”.
- **Expansion Definition:** The W3C PROV also supports an expansion definition in order to allow for certain requirement changes to be met over distinct domains.

¹Note that starting point and expanded properties only contain binary relations. These quantified classes and properties allow to extend the binary relations with comprehensive information

Provenance Security

Many widespread provenance systems deal with sensitive data in *specific domains*, such as healthcare, governmental institutions, banks, etc. Several security goals have been considered across literature for this data to not get corrupted or fall into the possession of a malicious entity [DKR⁺11, Che11, Cho09]. Goals explored in this area include *availability, integrity, confidentiality, and privacy*.

Security considerations covered in the PROV documentation are mostly related to overall web security and good practices, such as personal information leakage, DNS spoofing, attacks based on file and path names, content-disposition issues, Proxies and caching, Denial of Service (DoS) attacks, and more. These considerations can be seen as a good practice, but do not provide a firm security barrier to malicious entities.

Cheney et al. [Che11] take a more detailed approach to solving security problems w.r.t provenance. In the paper, a formal framework for provenance security with the aim to overcome the bottlenecks mentioned above is proposed. Furthermore, the framework simplifies many of the specific details of individual systems, enabling the identification of similarities and general characteristics across these systems.

To the best of our knowledge, there exists no related work mentioning the usage of distributed ledgers as an approach of ensuring integrity and tamper-resistance to provenance information in a global system.

2.2.2 Types of Provenance Systems

While the PROV-O standard is widely adopted as the industrial gold standard, over times different provenance system for concrete types of provenance have been developed [GSM⁺06, WS97, BKT01, GKT07]. In general, these can be grouped into four categories, as defined in [GSM⁺06]: *fine granularity provenance systems, domain-specific provenance systems, provenance in database systems, and middleware provenance systems*.

Granularity in Provenance Systems

Fine granularity provenance systems focus on storing a high-level of granularity which means to store a high amount of provenance data. As defined by Pérez et al. [PRS18], the exact meaning of granularity highly depends on the underlying data model of the application. The biggest challenge for those systems is the required storage volume, as the storage is directly proportional to the amount of provenance information available in the system. Depending on the application, different levels of granularities exist, where “coarse-grained” is considered low-level and “fine-grained” a high-level of granularity. It has been shown that in the worst case (where basically every piece of provenance information is stored), often the storage required for provenance data is even larger than the data itself [dCCM09].

Example 5: Dataset Provenance Granularity (dataset)

In the context of dataset provenance, some may need information about the origin of a single data set or small unit of data (high level), also known as data granule, while others may need information about the origin of a group of data sets (low level).

Example 6: Database Provenance Granularity (database)

Consider a relational database system. One can store the granularity per-row, per-cell or per table and is also possible to mix concepts. For example, it would be a good option to store provenance of a data element on a cell level, data unit on a row-level and context on a table level.

When necessary, it is not uncommon in certain cases to combine multiple granular levels.

Domain-specific Provenance Systems

Often provenance information is researched in the context of a particular business domain, also known as domain-specific. The usual advantage of having a system with provenance representations tailored to the specific needs are w.r.t. the model for provenance and its implementation syntax.

An early research area for provenance was the geographic information systems (GIS), which stores as provenance data the history of changes in a map [WPM⁺08]. This is essential as it allows GIS applications to assess the map quality and make future predictions. With regard to this, two systems specifically for the purpose of provenance tracking for GIS data have been proposed [Lan90, Bos02, ZGG⁺03].

Current research often focuses on multi-domain provenance solutions [HGS⁺06], allowing for using the same provenance tracking solution for multiple different problems.

Provenance in Database Systems

In database systems, the research for provenance tracking focuses on multiple problems. Most of the research concerns the data lineage problem [CW00, GSM⁺06], which involves in determining the set of source data used to produce a given item. Apart from this, there is another track where provenance related to a database query was part of the creation process of the data [BKT01]. In this field three distinct types of provenances have been identified, namely “why” provenance (explaining why an answer is in the result of a query) and “where” (tells us precisely from where an attribute value in the output was copied) provenance [LKL17, CCT09]. Furthermore, “how” provenance is a notion proposed by Green et al. in [GKT07], where they suggest a comprehensive provenance representation that uses semirings of polynomials as an algebraic structure.

Middleware Provenance Systems

Another category of provenance information is middleware systems. Such systems aim to add provenance support to existing applications, allowing as a central service the sharing of provenance information between multiple applications and domains [GSM⁺06].

An example of a system using a middleware for SQL is proposed by Lee et al. [LKLGI17]. The middleware runs on top of a relational database and uses SQL queries, providing answers to the “why” provenance. Furthermore, the system is capable of answering why a result is missing for the query, termed by the authors as “why-not” provenance.

2.3 Blockchain

The idea of the blockchain gained popularity in 2009 after Satoshi Nakamoto published a paper on an electronic cash system named Bitcoin [Nak]. Although the concept of digital money has existed since the 1980s, it took over 20 years to develop a fully decentralized transaction system, such as Bitcoin and its blockchain technology [TS16].

In this section, we explore the concept of distributed ledgers. We discuss the key properties of distributed ledgers in Subsection 2.3.1. Next, in Subsection 2.3.2, we present the fundamental concepts that enforce these properties. Then, in Subsection 2.3.4 we explore different categories of blockchains w.r.t participation. Finally, we introduce the Ethereum blockchain, in Subsection 2.3.3, as the second biggest blockchain in market capitalization.

2.3.1 Key Properties

Generally, when talking about blockchain systems, emphasis is put on decentralization. Even though decentralization plays a big part in the architecture of such systems, they provide for multiple additional properties which, based on the application, add further benefits. These properties are summarized as follows:

- **Decentralization:** A decentralized distributed ledger reduces bottlenecks, such as privacy, transaction speed, and security issues, that are introduced by involving a third-party entity in traditional centralized ledger systems.
- **Persistency:** Due to the immutable nature of blockchains, transactions can be validated quickly. Furthermore, honest miners would not admit an invalid transaction.
- **Anonymity:** As long as there is no link between a user’s wallet address and the identity of the user, transactions remain anonymous.
- **Auditability:** The output of a transaction cannot be higher than the input, making transactions easily verifiable and traceable.

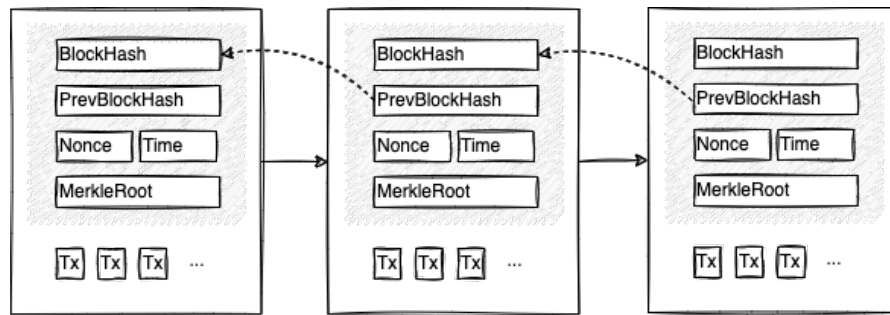


Figure 2.8: Blockchain protocol structure.

2.3.2 Fundamental Concepts

In the following paragraphs we introduce some of the key concepts that build up a blockchain. Bitcoin is chosen as an example, as it is the first blockchain that has been deployed. For this introduction we mainly follow the information provided by Tschoarsch et al. [TS16], Z. Zheng et al. [ZXD⁺17], and Antonopoulos et al. [Ant14].

Blocks. A fundamental concept in Bitcoin is that transaction records are stored in a tamper-proof linked list of *blocks*. Each block stores transaction records and meta data that includes a hash pointer to the previous list entry. Each hash depends on the block's content, enforcing the immutability of the list as visualised in Figure 2.8.

Mining. Mining is the process of creating new blocks. This creation is managed by so-called *miners*, that exchange blocks in the form of a peer-to-peer (P2P) network. Furthermore, miners ensure that the hash created for the block follows the blockchain's regulations. That is, for Bitcoin the Proof of Work (PoW) [Nak] procedure, ensuring that the hash begins with several zeros. In Bitcoin, the numbers of zeros is adapted by the number of participants (in detail the hash power) of the network to create one block about every ten minutes.

Transactions. Blockchain users handle the creation of *transactions* by utilizing asymmetric cryptography. Each key pair is linked to one or more addresses that are part of the blockchain. Essentially, the user signs the transaction with their private key after verifying they have enough balance in their address, and miners validate the transaction using the corresponding public key.

2.3.3 Smart Contracts in Ethereum

Early work on the topic of smart contracts has been done by Szabo [Sza97]. His work proposed that the future of law would rely on systems that algorithmically enforce agreements, namely *smart contracts*. Ethereum can be seen as one implementation of such a crypto-law system.

A smart contract is a program that runs on the Ethereum blockchain and is executed when certain conditions are met. Furthermore, the embedded code enables smart contracts to act autonomously. Contracts can be interacted with by calling methods, i.e. sending transactions to the smart contract account address.

Contract Accounts. In Ethereum we distinguish between two different types of accounts, externally *owned accounts* and *contract accounts*. While externally owned accounts are similar to the account types used in Bitcoin, the contract account represents a smart contract deployed to the blockchain network. This account cannot initiate a transaction and contains code which can be executed by triggering a transaction with the smart contract address as the recipient. Each transaction to this account involves a small amount of gas as operation cost, if the limit of gas is reached the transaction is aborted.

Gas Fees. Gas is a unit that measures the amount of computational effort taken to execute any transactions or smart contracts code on the Ethereum network. Every operation, including communication with smart contracts, in an Ethereum network requires some amount of gas [W+14]. Such fees can be seen as a payment for the computational resources used. Furthermore, fees are collected in the native currency, ether (ETH).

Contract Development. By the time of writing, *Solidity* and *Viper* are the two most popular languages for development of smart contracts. Solidity is an object-oriented language specifically designed to implement smart contracts. In contrast, Viper has dropped support for some object-oriented concepts with the aim to make contracts more auditable and secure. Furthermore, Solidity is greatly influenced by concepts from C++ in terms of syntax, but also notions from languages like Python and JavaScript can be noticed.

```
1  **
2  * @title Ballot
3  * @dev Implementation of a voting process containing vote delegation
4  */
5
6  contract Ballot {
7
8      struct Voter {
9          uint256 weight; // weight is accumulated by delegation
10         bool voted; // person has already voted
11         address delegate; // person delegated to
12         uint256 vote; // index of the voted proposal
13     }
14
15     address public chairperson;
16
17     mapping(address => Voter) public voters;
18 }
```

Listing 2.1: Gist of a smart contract utilizing voting, written in solidity

Development of Smart Contracts contracts for the Ethereum blockchain is a practice subject to constant changes due to improved writing standards, new vulnerability discovery, and community recommendations [PT21]. Often, it is a tedious task to keep up with the ever-evolving security practices accepted in the blockchain community. Being an open source project, participants in the Ethereum community can provide feedback on good practices and standards using the Ethereum Request for Comments (ERC) system. If accepted, these recommendations and standards are grouped into libraries, easily available for public use. Currently the system contains several reusable token standards.

Tokenization. The concept of tokenization aims to improve accessibility and transaction speed by enabling the quick and easy transfer of tokenized assets to anyone, anywhere in the world [Sch21]. Generally speaking, a token is a representation of an asset or utility. Tokenized assets can represent a variety of things, including both physical (*tangible*) assets like gold, real estate, and art, as well as non-physical (*intangible*) assets such as voting rights, ownership rights, and content licensing.

Token Standards. Token standards can be seen as templates containing a certain set of standardized functions, allowing for development of a unified token system. Tokens are split into two categories, *fungible* and *non-fungible*. Fungible tokens or assets are divisible and non-unique, while non-fungible tokens (NFT) are unique and non-divisible. Accordingly, different standards exist for the development of these tokens.

Example 7: Token standard

Currently the two most prominent token standards on Ethereum are the **ERC-20** and the **ERC-721** token standard [NFSC19].

ERC-20 is technical standard for a fungible token which allows for a way tokens to be exchanged with other tokens. Analogous to ERC-20, the ERC-721 standard allows creators to issue unique crypto assets like NFTs via smart contracts.

Smart Contract Storage Design. According to a previous study [XWS⁺17], it is common for data management in blockchain-based systems to store raw data off-chain due to the limited amount of computational power and data storage space available. This method is especially useful in distributed ledgers where the payload transaction size is limited, such as Bitcoin.

In the context of Ethereum, it is allowed to store arbitrary structured data of any size, in contrast to other blockchain systems such as Bitcoin. According to the Ethereum yellow paper [W⁺14], each transaction has a fixed upper limit use of 21,000 gas (price in GWei: 630,000²), and an additional 68 gas is incurred for every non-zero byte of data. Therefore, the total cost of storing 50 bytes of data on the Ethereum blockchain via a transaction uses approximately 24,400 gas, or \$0.0005 at the current exchange rate. ³

In addition, to storing data via transactions on the blockchain, *two additional methods* for storing arbitrary data in smart contracts are provided.

The first method is to store the data as a variable in the contract, which has a cost based on each 32 bytes of data (uses approximately 20,000 gas) and the cost of creating the contract. The total cost for storing 32 bytes of data in this way is approximately \$0.0004.

The second method is to use a log event to store the data, which has a cost based on the fixed cost of the transaction and the cost of storing the data in log topics. The total cost for storing 32 bytes of data in this way is approximately \$0.0002. The first method is more efficient for manipulating data, but less flexible due to constraints in the Solidity language. The second method has intermediate flexibility and performance, as it allows up to three parameters to be queried.

It is worth to note that, as stated in [XWS⁺17], storing data on the blockchain is a one-time cost permanent storage. This means that, in contrast to conventional data storage models, you only have to pay once to permanently store data on the blockchain.

²Prices are denoted in the unit Gwei as denomination of ETH, where each gwei is equal to 0.000000001 ETH. Current gas price retrieved from https://ycharts.com/indicators/ethereum_average_gas_price. [Date accessed: 01-02-2023]

³Current prices are retrieved from <https://coinmarketcap.com/> and are subject to constant change. [Date accessed: 01-02-2023]

	Public	Consortium	Private
Centralized	No	Partially	Yes
Immutable	Yes	No	No
Read permission	Public	Public or restricted	Public or restricted
Efficiency	Low	High	High
Consensus determination	Miner nodes	Specific nodes	Specific nodes

Table 2.2: Comparison between public, consortium and private blockchains

2.3.4 Categories

Distribution of ledgers can also be limited to a certain scope of nodes. Research has identified three broad categories of blockchains: *public*, *private* and *consortium* [TS16]. The most well-known blockchains are *public*, as anyone can participate in such and all records are publicly accessible. *Consortium* and *private*, however, limit participation in the consensus process of the blockchain to a certain set of pre-selected nodes. Specific details can be found in [Table 2.2].

2.4 Data Catalogs

In [Section 2.2], we discussed provenance information and prominent models for storing this type of data. To efficiently track the provenance of vast amounts of data in an organized way, it is crucial to utilize a suitable data management system. This is where data catalogs come into play, offering a comprehensive solution for organizing data. Moreover, several data catalogs support the Prov-O ontology, which we identified as an effective standard for representing and sharing provenance information.

A data catalog, also called a data management system, is a detailed collection of data sets available across an organization. Such collections help data professionals to easily store and locate specific data for analytical and business purposes. Among other benefits, data catalogs provide an easy process for publishing, sharing, and using large quantities of data.

The remaining parts of this section are structured as follows: Firstly, we explore the definition of a data catalog, along with the benefits of using such a system. We then proceed to investigate the latest available data catalogs and assess their relevance to this thesis by comparing them based on multiple key features.

Definition 2.4.1 (Data Catalog [ora]). “A data catalog is an organized inventory of data assets in the organization. It uses metadata to help organizations manage their data. It also helps data professionals collect, organize, access, and enrich metadata to support data discovery and governance.”

Characteristics. Modern data catalogs have many features, functions, and key characteristics, enabling a precise fit to an organization’s specific needs. Such characteristics differ throughout different data catalogs, however, they all follow the core principle of collecting metadata that identifies and describes the inventory of shareable data. Recent efforts around improving such systems include analytical automation with the help of semantic inferences and machine learning. Such automation can be found in several parts of building a data catalog, such as initial catalog building, data discovery, and data processing.

With robust metadata as the core of the data catalog, many other features and functions are supported. A comparison of such features among modern data catalogs can be found in [Table 2.3](#). The most essential features include:

- *Open-Source Software (OSS)*: computer software, released under a license where the user is granted permission to freely use, change and distribute copies of the software and corresponding source code for any purpose. A great benefit of OSS is that it can be developed with public contributions in a collaborative manner.
- *Search-based (S-B)*: ability to do specific searches of entities in data catalogs.
- *Federation (F)*: capability to create a mapping of multiple data catalogs to an isolated interface aiming to circumvent data entity repetition.
- *Data Quality (DQ)*: incorporates modern data quality assurance tools, integrations and/or methodologies.
- *Data collaboration (DC)*: enables the possibility to extract data from various internal and external origins to allow for compound data insights and extensive analytics.
- *End-to-end provenance (E2E-P)*: refers to provenance (data provenance) of all data assets utilized in the organization across all data catalogs and integrated tools (add-ons).
- *Column-level provenance (CL-P)*: specifies provenance data with regard to column level granularity.
- *Network-based (NB)*: similar concept to end-to-end-provenance, however with focus on the ownership of the data assets.
- *RDF support (RDF)*: Supports the RDF data model.

Data Catalogs for Linked Open Data. In the context of linked open data (Linked Open Data), data catalogs can be used to store and manage Linked Open Data datasets and make them accessible to users. This may involve adding metadata to the datasets to describe their contents and context, and providing tools that allow users to search for and access the data they need.

	OSS	S-B	F	DQ	DC	NB	E2E-L	CL-L	RDF
Google DC	×	✓	×	?	×	×	×	×	×
OpenMetadata	✓	✓	✓	✓	✓	✓	✓	✓	×
CKAN	✓	✓	✓	×	✓	×	×	×	✓
Magda	✓	✓	✓	×	✓	×	×	×	×
Monte Carlo	×	✓	×	✓	×	×	×	×	×

Table 2.3: Comparison between properties of modern data catalogs.

Linked Open Data segmented in data catalogs may also provide features that allow users to explore the data and see how it is linked to other datasets. This can be particularly useful for researchers and data scientists who need to find and use data from multiple sources, as it can help them understand the relationships between different datasets and how they can be combined and used together.

Additional information regarding the state-of-the-art on data catalogs can be accessed in “Data Catalogs: A Systematic Literature Review and Guidelines to Implementation.” [ESM⁺21].

2.5 Related Work

In this section, scientific background and state-of-the-art on topics relevant to this thesis will be covered. The main focus will be on a selection of papers that were recognized to have the highest relevance. To the best of our knowledge, none of them cover completely the area of interest for this thesis as defined in [Section 1.1](#).

The rest of this chapter is organized as follows. Firstly, in [Subsection 2.5.1](#), we present an overview of current solutions aiming to provide a provenance trace to Linked Data. Then, in [Subsection 2.5.2](#), we introduce the distributed ledgers to this context and explore several papers utilizing this concept. Finally, in [Subsection 2.5.3](#) we examine a taxonomy for choosing a blockchain when starting the development of a software project.

2.5.1 Provenance of Linked (Open) Data

Throughout the expansion of the Web of Data, poor-quality data propagation remained one of the most significant vulnerabilities. Data derived from replication, merging, or modification resulted in reduced data quality. Research has identified methods that use provenance information as a means of dealing with this drawback [HZ10, CMD⁺11]. Since then, the W3C has developed PROV, a family of documents describing a model, leveraging various aspects necessary for defining and managing provenance information in heterogeneous environments such as the Web. This set of documents is covered in detail in [Subsection 2.2.1](#). The standard offers multiple serializations and thus is not limited to Linked Data.

Wikidata, a platform for collaborative acquisition and maintenance of structured data [Vra12], is currently one of the most prominent examples of an open knowledge base. As such, it has provided a wide variety of research topics. One such research, conducted by Piscopo et al. [PS19], covered the quality of the retrieved information. The literature survey has shown that several quality dimensions are flawed, including accuracy and trustworthiness. To some extent this was defined to be due to the lack of provenance information and poor governance of such information.

Early work on provenance for Linked Data is investigated by Wylot et al. [WCHG17]. The writers identified a growing diversity of Linked Data on the Web and the challenges it presents for database systems. The paper focuses on how these large quantities of data impact the capability to store, track, and query provenance data.

The authors observed that the most widely used method for managing RDF data is named graphs, standardized in RDF 1.1 [rdf]. This method involves assigning a set of triples with a URI, allowing metadata, including provenance, to be linked to the graph. While named graphs are often used for provenance, they can also be used for other purposes, such as tracking access control information. Despite the widespread use of named graphs in RDF databases (also known as triplestores), the paper points out that there have been relatively few approaches specifically focused on efficient implementations of provenance within triplestores, and much of the research in this area has focused on the theoretical aspects of the problem rather than practical implementation.

In an attempt to overcome these shortcomings, they presented a new RDF store-based database system called TripleProv. TripleProv is based on a native RDF store, to which two new storage models are added. These models consider the origin of data and methods for efficiently organizing provenance data storage. With this approach, they demonstrate how databases can efficiently track the provenance information of queries. In the end, two experimental scenarios of the system are presented, evaluating the scalability of the system.

2.5.2 Provenance of Linked Open Data and Distributed Ledgers

The interlink between blockchain and Linked Open Data has gained popularity as a research topic in recent times [NSNF17, KH21]. A paper by Kirstein et al. [KH21], offers a blockchain-based approach to enabling an integrated, traceable, and timely view of Linked Open Data. The aim is to tackle a flaw within the modern architecture of Linked Open Data infrastructures, where the data providers withhold complete control of the data. This flaw is a direct result of the fact that most modern Linked Open Data infrastructures are built in a decentralized manner. It has been observed [NTLK18] that such an approach yields not only poor system performance, like high algorithmic complexity but also low data quality, weak interoperability, and little incentive for the involvement of the community. Furthermore, it is investigated how blockchain technologies can assist in addressing critical problems in the publication of Linked Open Data. Due to the fact that in the paper, the primary target is Linked Open Government Data (LOGD),

which has some specific challenges, only relevant problems related to Linked Open Data are extracted:

1. Distributed publication implies time-delayed and pull-based harvesting infrastructures. In such a situation, a blockchain can be utilized to act as a virtual access point. This virtual access point would represent a shared state between providers, preventing inaccuracies, and could also be used to ease the onboarding process of data providers.
2. A known problem in Linked Open Data is that it often includes low-quality and heterogeneous data. It has been recognized that a blockchain's consensus and immutability can enforce quality assurance over data publishers and homogeneous data standards.
3. Often, publication schemes don't provide a way to input provenance or enforce data trust. To overcome this limitation, the immutability aspect of the blockchain can be utilized, ensuring the thorough history of individual datasets by acting as a provenance layer.

All of the above-mentioned problems are tackled in the solution provided by Kirstein et al., in an implementation called Open Data Blockchain (**ODBI**), implemented as a working prototype based on Vert.x⁴ and MongoDB⁵. This custom-built blockchain presents promising results w.r.t. transparency, data quality, provenance tracking and robustness. However, how a similar approach will act over existing blockchains is not covered and remains interesting. Depending on the blockchain, this includes transaction costs, storage restrictions, and automated contract deployments.

Most papers, which aim to combine these scientific fields, adapt an approach of combining data provenance and the blockchain by having a concrete requirement in mind. In the case of the paper, written by Ricardo Neisse et al. [NSNF17], the focus is on storing provenance using the blockchain in order to enforce the General Data Protection Regulation (GDPR). These requirements are a unified set of rules with legally binding power that must be enforced in all EU member countries. Given the legal context of this paper, a significant emphasis is placed on privacy. A blockchain-based system for data accountability and provenance tracking is presented.

Furthermore, an analysis on the design choices, performance, scalability and implementation of the system is shown, taking into account the implementation of two models. These models account for expressing consent rules regarding the usage of personal data. In the first model, these rules were embedded in contracts deployed on the blockchain and

⁴Vert.x is a polyglot event-driven application framework. Further details about this framework can be found in <https://vertx.io/>

⁵MongoDB is a type of NoSQL database that can run on different platforms and its source code is available to the public. It stores data in the form of documents, which are similar to JSON and can have optional rules for their structure.

specific to each controller or processor. In the second model, each controller expressed their usage control policies in a blockchain contract with an interface that allowed users to join or leave the contract, thereby giving or withdrawing consent for each data controller or processor. These policies, which could be selected in advance or from a library of policy templates, outlined the conditions for data access, usage, and transfer to processors. The two models have different properties in terms of user privacy, data accountability, and data tracking granularity, which are discussed in the paper.

Additionally, great attention is paid on the design of the smart contracts which enforce the provenance models. The Model-based Security Toolkit (**SecKit**) [NSNFB15] is used as a modelling approach.

ProvChain [LST⁺17] is another system that utilizes the provenance trace with the help of blockchain technology. The authors present a method called “hooks” that tracks and records arising changes in the cloud storage system. By using these hooks, events are generated in accordance with the user’s actions. Once the events are created, they are stored on the blockchain as transactions. Once data is stored on the blockchain, the ProvChain system verifies the changes by using an external entity known as the auditor. The essence of the auditor verification process is generating receipts. However, this can be seen as a drawback because the verification process is centralized and thus can be biased.

This problem is tackled by Ramachandran et al. [RK18] by implementing an incentivized voting mechanism, by introducing rewards and penalties based on smart contracts. In this way, users who try to log invalid changes to the system can be penalized. Notably, the authors of **Smartprovenance** propose a solution to protecting data provenance in the context of open data, specifically by using an open-source data catalog, CKAN. Another significant difference is that Smartprovenance automates the verification process using verification scripts, enabling the rejection of invalid changes. However, all this functionality comes at a high cost, as identified by the authors. Furthermore, specific issues have been identified when it comes to concurrent disputes of a dataset.

A similar approach using the same platform is introduced by Dang et al. in [DA20]. They show a blockchain-based solution for managing provenance and characteristics in the open data context. In a like manner to the previous paper, the authors integrate their solution with CKAN. One of the main focuses in this paper is implementing two systems, namely Portal Application Programming Interface (API) and platform identity. The Portal API is a custom solution that enables the outside world (in this case, the CKAN platform) to communicate with the blockchain. The platform identity is an identity system that allows for different platforms to be identified by issuing unique certificates, in short, fingerprints.

Multiple papers have adopted Hyperledger Fabric (HF) as a blockchain solution because it provides an excellent way to manage organizations, however, this is also a drawback due to the proprietary nature of the blockchain.

In [DA20] the authors have chosen to use HF specifically because of its organization-based

	Chain	Type	Stored Data Verification
ODBI	custom	-	×
ProvChain	custom	-	×
Smartprovenance	Ethereum	public	✓ (decentralized voting + automatic scripts)
Dang et al.	HF	permissioned	×
ProvHL	HF	permissioned	×

Table 2.4: Comparison between properties of related work

nature. Furthermore, it has been deemed essential to implement a queue process to ensure the correct synchronization of data provenance.

Similarly, using HF, the authors Demichev et al. [DKP18] present **ProvHL** - a provenance hyper ledger for managing provenance metadata and data access rights in distributed storage spaces. The proposed system extends further on provenance of storage files as it enables user management restrictions depending on the provenance data. The authors pay great attention to the problem of the optimal choice of the blockchain type for such a system.

In Table 2.4 we show a comparison of the related work mentioned in this section by blockchain type and ability to verify stored data.

As we can see with many research approaches above, the direction to protecting data provenance using the blockchain is feasible. Certain solutions enhance functionality by incorporating a method to verify provenance data as well. However, most of the results presented have specific requirements to meet, resulting in a domain-specific solution. This is expected as most solutions focus mainly on the commercial sectors. This thesis aims to provide a more general solution that can be extended further to fit the specific needs of particular provenance ecosystems.

2.5.3 Blockchain Taxonomy

As a recently emerged technological solution, different blockchain systems have been developed in the past years. Some are domain-specific, solving a specific problem, while others vary in internal structure. As many options exist, it is vital for the successful implementation of our system to make an optimal choice.

Xu et al. [XWS⁺17] have developed a taxonomy of blockchain-based systems that can be used as valuable insight when choosing the architectural design of a system utilizing digital distributed ledgers. In this thesis we use this system to determine the most fitting blockchain for our solution.

Firstly, the authors take into account the architectural design regarding decentralization.

In a centralized system, all transactions are mediated by a central authority. This means that users must trust the central authority to handle transactions correctly. In contrast,

Design Decision	Option	Impact			
		Fundamental properties	Cost efficiency	Performance	# Failure points
Fully Centralized	Services with a single provider (e.g., governments, courts)	✓	✓✓✓	✓✓✓	1
	Services with alternative providers (e.g., banking, online payments, cloud services)				
Partially Centralized & Partially Decentralized	Permissioned blockchain with permissions for fine-grained operations on the transaction level (e.g., permission to create assets)	✓✓	✓✓	✓✓	*
	Permissioned blockchain with permissioned miners (write), but permission-less normal nodes (read)				
Fully Decentralized	Permission-less blockchain	✓✓✓	✓	✓	Majority (nodes, power, stake)

Table 2.5: Blockchain-related design decisions regarding (de)centralisation, with an indication of their relative impact on quality properties (✓: Less favourable, ✓✓: Neutral, ✓✓✓: More favourable).

decentralized systems like Bitcoin allow users to reach a consensus on ownership without needing to trust each other or a third party, which makes the system highly available and resistant to failure. Furthermore, partially centralized/decentralized systems exist, where there are two main options to consider - permission and verification.

- **Permission:** Blockchains that contain a central authority as a gate of participation are considered permissioned. Such authority could restrict participants from joining the blockchain, executing transactions, or even more fine-grained controls. Permissioned blockchains tend to be more suited for regulated industries bound by additional jurisdictional boundaries exist. [Table 2.5](#) presents a range of levels of centralization and decentralization, from fully centralized to fully decentralized. The downside of using a permissioned mechanism is that a potential single point of failure is introduced.
- **Verification:** In this context, blockchain verification is the issue of evaluating external (wavering) conditions that cannot be defined in the blockchain itself. A verifier role can be introduced to address this factor, who verifies against this external condition. A smart contract within a blockchain network can serve as a verifier by periodically incorporating external information into its processes. However, similar to the case of permissioned blockchains, a centralized verifier acts as a potential single point of failure. To overcome this drawback, multiple decentralized verifiers with the same task can be introduced.

Although blockchain verification, out-of-scope for this thesis, it is an interesting topic that could be explore further and is discussed in [Chapter 6](#).

Secondly, the authors explore design decisions related to storage and computation.

When using a blockchain, it is essential to consider balancing cost efficiency, performance, and flexibility when deciding what data and computation should be included on the chain

and what should be kept off the chain. One of the downsides of using the blockchain in software development is the fact that it has an upper bound on computational power and available storage space. During the research on this topic, one of the challenges was storage space w.r.t. Ethereum smart contracts, discussed in [Subsection 2.3.2](#).

In the next list, several factors which impact the choice blockchain based on the above-mentioned restrictions are shown:

- **Item data:** It is common for data storage to take place off-chain, keeping only crucial metadata and hashes on the chain [\[XWS⁺17\]](#). Choosing off-chain data storage for a blockchain system involves deciding how the blockchain will interact with the storage and whether to use a private cloud on the client's infrastructure or a public storage provided by a third party or network. The flexibility of using cloud storage can depend on the specific implementation.
- **Computation:** There are two options for performing computation on a blockchain, on-chain (e.g., through smart contracts) and off-chain. Turing-completeness is one factor that impacts a blockchain's computational expressiveness. Bitcoin is a minor in this category, as it is purposely Turing incomplete to prevent computational loops from consuming too many resources for nodes. This means that only simple scripts and conditions must be met to initiate Bitcoin transfers. Ethereum, on the other hand, is built as a Turing-complete blockchain. This allows for making modifications to data in smart contract variables, as well as conditional transactions. Using on-chain computation, rather than using the blockchain solely as a data storage layer, has the advantage of creating interoperability among systems built on the same blockchain network. Other benefits include the neutrality of the execution environment and the inability to change the program code once it has been deployed.

An evaluation of different blockchains based on the considerations stated above is shown in [Table 2.6](#).

Lastly, the technological characteristics and configuration options of different blockchain are discussed. An analysis of the most significant technological differences and what benefits they bring to certain use-cases is presented. The taxonomy of configuration options on blockchains can be found in [Table 2.7](#). We start by examining one of the most important traits, namely the scope of the blockchain.

- **Scope:** As noted in [Subsection 2.3.4](#) there exist three types of blockchain w.r.t. scope - public, private, and consortium. Among other things, this difference impacts the user's write permission. Private blockchains, which are governed and hosted by a single organization, offer the most flexibility in terms of configuration. However, using a private blockchain requires a permission management component. This can be seen as a drawback in specific cases, as it introduces a single point of failure to the system.

Design Decision		Option	Impact			
			Fundamental properties	Cost efficiency	Performance	Flexibility
Item Data	On-chain	Embedded in transaction (Bitcoin)		✓	✓	✓✓
		Embedded in transaction (Public Ethereum)	✓✓✓✓	✓✓✓✓	✓	✓✓✓
		Smart contract variable (Public Ethereum)		✓✓	✓✓✓	✓
		Smart contract log event (Public Ethereum)		✓✓✓	✓✓	✓✓
	Off-chain	Private / Third party cloud	✓		✓✓✓✓	✓✓✓✓
		Peer-to-Peer system		✓✓✓✓	✓✓✓	✓✓✓
Item Collection	On-chain	Smart contract	✓✓✓✓	✓✓✓✓	✓✓✓✓	✓
		Separate chain		✓	✓	✓✓✓✓
Computation	On-chain	Transaction constraints	✓✓✓✓	✓	✓	✓
		Smart contract				
	Off-chain	Private / Third-party cloud	✓	✓✓✓✓	✓✓✓✓	✓✓✓✓

Table 2.6: Blockchain-related design decisions regarding storage and computation, with an indication of their relative impact on quality properties (✓: Least favourable, ✓✓: Less favourable, ✓✓✓: More favourable, ✓✓✓✓: Most favourable)

- Data Structure:** Refers to the way the blocks are organized. For example, in Bitcoin, the chain is organized so that new blocks are appended to the end, preventing malicious blocks from entering the chain history. When a conflict occurs, the longest chain is selected by participants. This strategy is also known as the *longest-chain* protocol. Another chain-selection protocol is the Greedy Heaviest-Observed Sub-Tree (GHOST) protocol, where miners may reference independently-mined blocks from other miners to add weight to their own chain and increase the likelihood that it will be chosen as the main chain. Some proposals have suggested altering the structure of the blockchain from a linear list to a DAG to enable the incorporation of non-conflicting transactions from uncle blocks into the main chain. There are various design options for the internal structure of blocks as well. One such proposal is called segregated witness, which involves separating signatures (witnesses) from the rest of the data in a transaction in the Bitcoin community.
- Consensus Protocol:** These protocols help all the network nodes verify the transaction. The Bitcoin blockchain uses the Proof-of-work mechanism as a consensus protocol, which is energy and time-intensive. A compelling implementation can be found in Hyperledger Fabric, as it utilizes a modular architecture that enables implementations of various consensus protocols. Another modification of Proof-of-work found across the literature is Proof-of-retrievability. This proposal aims to redirect Bitcoin’s mining resources toward distributed archival data storage. A widely-popular consensus is the Proof-of-stake, where the next mining node is selected based on the participant’s ownership of the native digital currency of the blockchain network. When it comes to permissioned blockchain, applications of the BFT protocol can be found. It has been observed that this protocol offers improved latency and consistency, however, with the restriction of a smaller participant pool.

2. PRELIMINARIES

Design Decision	Option	Impact			
		Fundamental properties	Cost efficiency	Performance	Flexibility
Blockchain scope	Public blockchain	✓✓✓	✓	✓	✓✓
	Consortium/community blockchain	✓✓	✓✓	✓✓	✓✓✓
	Private blockchain	✓	✓✓✓	✓✓✓	✓
Data structure	Blochain	✓✓✓	✓	✓	✓
	GHOST	✓✓	✓✓	✓✓	✓
	BlockDAG	✓	✓✓✓	✓✓✓	✓✓✓
	Segregated witness	✓✓✓	✓✓	✓	✓
Consensus protocol	Proof-of-work(PoW)	✓✓✓	✓	✓	✓
	Proof-of-retrievability(PoR)	✓✓✓	✓	✓	✓
	Proof-of-stake(PoS)	✓✓	✓✓	✓✓	✓✓✓
	Byzantine Fault Tolerance (BFT)	✓	✓✓✓	✓✓✓	✓

Table 2.7: Blockchain-related design decisions regarding blockchain configuration (✓: Least favourable, ✓✓: Less favourable, ✓✓✓: More favourable, ✓✓✓✓: Most favourable)

In summary, the successful implementation of a blockchain system depends on choosing the most optimal solution. The taxonomy developed by [XWS+17](#) allowed us to easily consider the advantages and disadvantages of different systems and eventually make the right choice of blockchain to use in our implementation. It is worth to note that not all decisions about the choice of blockchain are solely based on this taxonomy. Factors outside of the technological scope were also considered e.g. developer experience with the different systems. We discuss the details of our system of choice in [Subsection 4.1.1](#).

CHAPTER 3

Design

In this chapter we present the design for solving the problem stated in [Section 1.1](#). The outlined design aims to establish a system that functions as a shared state for Linked Open Data and provides a reliable source of truth for provenance information. This process involves articulating comprehensive specifications and formulating an architectural framework for the system, which will guide subsequent implementation efforts. Furthermore, one sub-research question, namely **RQ3.2**, is identified and explored, as a result of the research conducted. The rest of this chapter is structured as follows:

In [Section 3.1](#), we specify requirements that need to be met in a PoC implementation based on the research questions. Lastly, in [Section 3.2](#), we present an overview of the system architecture together with the underlying components.

3.1 Requirements

The aim of this section is to lay foundations and to describe considerations that should be taken into account during the implementation. An essential part of our system implementation will be composed of two middleware agents, the requirements of which will be presented separately. In addition, the functionality requirements of the Smart Contract for storing provenance are defined.

As often done in software development, the requirements are based on *use cases*, details of which can be found in the next subsection.

3.1.1 Use cases

A wide case of scenarios exist when implementing a software project with such a broad scope. In this subsection, we aim to provide several priority scenarios, predisposing to the usage of our system. The documented scenarios are fictional, however, they are designed

Use Case X: {Use Case Application}**Actor:** {The main person or object performing the actions.}**Stakeholder:** {Anyone who is particularly interested in how the system operates and behaves. Often they are not people in direct usage of the system but indirectly benefit from it.}**Scenario:** {Details about the situation the actor is in when needing to perform this use case.}**Preconditions:** {Statements that describe the necessary conditions that must be met before the use case can be executed.}**Basic flow:** {Description of the basic flow, also known as the main success scenario, is a use case that runs smoothly and without any exceptions or errors.}

to meet real life usage examples. This approach would allow us to identify impediments not only in the technological implementation, but also in the user experience process of our system.

The formulation of a use case primarily takes into consideration the best practices obtained from a book by Gomaa et al. [Gom11]. Above, we present a use case (UC) template, accompanied by an explanation of the various components comprising the case structure.

Based on the above template, we give several examples of real-life scenarios of how certain actors can make use of the corresponding software.

Use Case 1: Governmental Data Storage**Actor:** Data analyst.**Stakeholder:** State institution, Open public.**Scenario:** The Austrian government wants to publish a dataset containing information on the COVID daily trends. The provenance is based on multiple datasets, comprised in different formats. By storing the provenance they aim to prevent erroneous data / duplications from entering the data and be able to keep track of publishers. Being a state institution, data security is a priority.**Preconditions:** Dataset is published in the open data platform. Smart Contract is updated and stores a hash of the dataset together with provenance information.**Basic flow:** Governmental Data Storage.**UC 1 1****Use Case 2: Governmental Data Retrieval****Actor:** Person with interest of obtaining open governmental data.**Stakeholder:** Open public.**Scenario:** Someone is developing an open-source provenance validation engine of Linked Open Data and wants to test the system on governmental data.

Preconditions: Access to the system is provided.

Basic flow: The latest dataset version is retrieved from the data catalog. Additionally, dataset provenance is successfully retrieved from the blockchain.

UC 2 2

Use Case 3: Non-profit Organization Data Storage

Actor: Campaign manager.

Stakeholder: Open public.

Scenario: The campaign manager wants to securely store transaction data regarding the organization's fiscal year. It is vital, that the origin of every single transaction is justified. Furthermore, the organization policy necessitates transparency and traceability of financial resources, so he would like to make the information open to the public.

Preconditions: Access to the system is provided and multiple dataset to be stored is available.

Basic flow: The dataset is successfully published on the data catalog. Smart Contract is updated storing a hash of the dataset, dataset provenance, and transaction provenance information. Both dataset and provenance information are publicly accessible.

Use Case 4: Existing Dataset Duplication

Actor: Open-source contributor.

Stakeholder: Developer community.

Scenario: A developer would like to test the robustness of his latest contribution to an open-source project, before submitting a pull-request. The project he is contributing to is used to verify data provenance. One requirement he has is that the testing provenance data doesn't change with time, so he can obtain constant results across time for all his tests.

Preconditions: Access to the system is provided and the provenance information is available.

Basic flow: The provenance contract is used to successfully retrieve a specific version of provenance data.

These use cases act as a foundation for defining the system requirements needed for a PoC implementation. Therefore, in the next subsections functional and non-functional requirements are presented.

3.1.2 Functional Requirements

Functional requirements describe how the system should behave in terms of functionality [Dav93]. For presentational purposes, we have divided the functional requirements based on the objectives of the main components of our system. Every requirement is followed by a concise explanation and a reference to relevant use cases. The following requirements have been established:

Data extraction agent middleware:

- *Extraction of heterogenous data to structured data:* The agent must be able to format heterogenous data into a certain structure, better suited for machine processing.
- *Publishing to Data Catalog:* The agent must be able to publish the acquired data into a data management platform.

The aforementioned requirements are derived from [UC 1](#) and addresses the necessity for extracting provenance of data provided in different formats. Furthermore, the need for the data to be published to a data management system is addressed.

Provenance handling agent middleware:

- **Capture:** The agent must be able to capture incoming data from the data catalog.
- **Verify data integrity:** The agent must be able to verify the integrity of the data provided.
- **Excessive Data Storage:** The agent must be able to save data on-chain if the size doesn't exceed a particular threshold.
- **Publish:** The agent must be able to publish the captured triples onto the blockchain by utilizing a smart contract.

The above requirements are extracted from [UC 1](#) and [UC 3](#). The initial requirement addresses the need for communication between the data catalog and the provenance handling agent. The second requirement focuses on security concerns, primarily discussed in [UC 1](#). Although the third requirement is not explicitly mentioned in a use case, it is inferred based on research outlined in [Subsection 2.5.3](#). Lastly, the final requirement is found in all use cases and the need for communication between the agent and the blockchain.

Smart Contract:

- **Storage:** The contract must be able to store RDF triples of LOD provenance information into a PROV model onto the blockchain. In addition on-chain storage of dataset metadata must be possible.
- **Reference:** The contract must be able to keep a reference (hash code) of the dataset that the provenance information refers to.
- **Access:** The contract must be able to provide access to the provenance data stored for a specific dataset.

Requirements for the smart contract are extracted based on all use cases. They refer to the fundamental goal of our solution to store and provide provenance data in a secure manner.

3.1.3 Non-functional Requirements

Non-functional requirements describe how the system works w.r.t. properties and constraints. [Dav93]. As such, in this subsection specifications that describe the system's operation capabilities and constraints are shown. We present requirements for our system in a broader scope and we do not differentiate between explicit system components.

The requirements are defined as follows:

Reliability. We aim to implement a highly reliable system with, with a low risk of data loss or corruption. In our context, the reliability evaluation should focus on how well the system leverages blockchain technology to maintain data integrity in addition to assessing the design and implementation of smart contracts, their ability to validate and enforce data integrity rules, and their robustness against potential security threats.

Security. Security is a priority requirement for our solution as we aim to ensure data trust for provenance information. The system should ensure the security and privacy of sensitive information and prevent unauthorized access, modification, or deletion of data.

Capacity. System capacity is one of the major challenges developers are faced with when implementing solutions utilizing the blockchain. The system should have the capacity to store and process large volumes of data.

Extensibility. Extensibility is another type of requirement we will comply with. The system should be designed with extensibility in mind, so that new features and functionality can be added as needed. Furthermore, the solution should be modular and easily maintainable to allow for future updates and enhancements.

Based on the functional and non-functional requirements defined above we are able to define an architecture of a system implementation in the following section.

3.2 Architecture

In this section, we go over an abstract blueprint of a system for efficient storage and management of provenance information for cataloged open data in a tamper-resistant manner. The main goal is to discover an appropriate resolution for recording, coordinating, and gathering provenance information within the blockchain, while also enabling authorized access to utilize this information. We will discuss the rationale behind the design choices made for each component, exploring alternative options and highlighting the reasons for selecting a particular approach based on the requirements and established best practices. Additionally, establishing an optimal architectural structure is crucial when evaluating the technology stack in the subsequent chapter.

The proposed architecture consists of several individual components, the fundamental functionality behind them, and an explanation of how they interact with each other.

In [Figure 3.1](#) a sketch of the proposed solution is displayed. In the next subsections we explore in detail every single component. Next, we provide a brief overview of the proposed solution.

The architecture is comprised of a *data source*, a *data extraction agent*, a *provenance handling agent*, a *data management platform* (data catalog), a *graphical user interface (GUI)*, and a *smart contract* deployed on the blockchain. The system trigger is shown in the form of an individual (Actor), supplying data to the platform. Once the data is provided, the data extraction agent takes care of making sure the data complies to the RDF format and publishes the dataset to the data catalog.

Once the data reaches the data catalog, it gets extracted by the provenance handling agent on specified time intervals. Subsequently, the agent manages the provenance information of this data and appends it to the blockchain.

3.2.1 Data Sources

The data source component in [Figure 3.1](#) has the basic role of a remote storage system for heterogenous data, in our case - data sets.

A data set is a group of interconnected data that can be accessed individually, combined, or managed as a single entity. Often, it is based on a collection of related, discrete pieces of data. Data sets include many different files, also known as resources. In addition to these resources, metadata can be provided, containing details about the structure. In our context, such metadata can be provenance information, the physical URL to access the files, ownership, publishing status etc.

A primary objective of this thesis is to track provenance information of datasets. At a low granularity level, this entails documenting information regarding the dataset's creation and all future actions performed on it. Conversely, at a high granularity level, all the metadata contained within the dataset is utilized and stored on the blockchain.

3.2.2 Data Extraction Agent

The goal of this component is to extract data from multiple remote data sources and deliver those resources to the data catalog.

Various approaches can be considered for this component, such as ETL (Extract, Transform, Load) tools or custom scripts. However, we opted for a workflow-based approach with Directed Acyclic Graphs, also known as DAGs, which represent a collection of all the tasks we want to execute. This allows for a more granular control over the extraction process, better error handling, and easier extensibility. This approach also aligns with established best practices in data pipeline management.

The main task is to retrieve the remote file, extract all datasets and create or update actual cataloged datasets based on that. Additional tasks may include deletion handling, error handling and more. Furthermore, a sequence and dependencies in which tasks

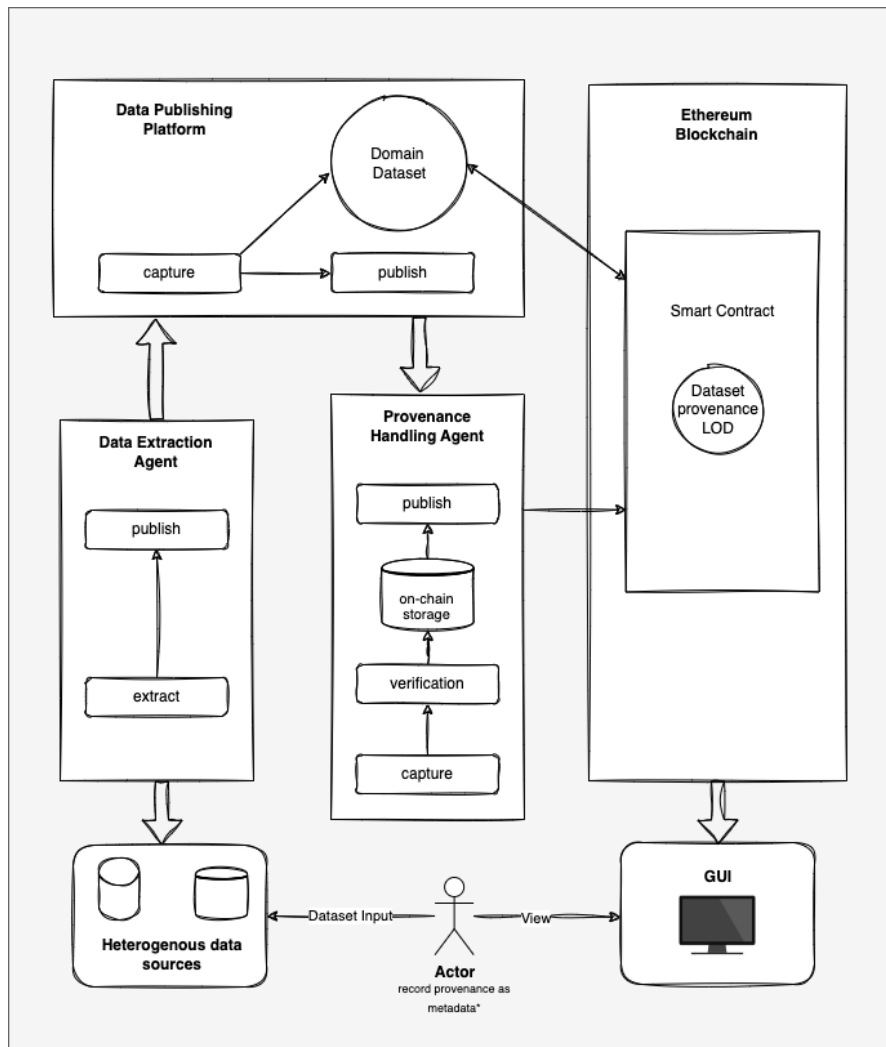


Figure 3.1: System Overview.

should be executed can be defined, so that certain tasks cannot be executed before the previous workflow has terminated successfully.

3.2.3 Data Catalog

The data catalog component is the main communication point to the open public. It can be used to manage, search and retrieve datasets, the provenance of which can be later validated against the corresponding smart contract. As stated in [Section 2.4](#), modern data catalogs can be classified by multiple differences in characteristics. As our goal is to accomplish a global solution to LOD provenance storage, the aim is to develop a solution which can be easily transferred to each and every data catalog system independent of individual characteristics.

To achieve this, we designed our architecture to be agnostic of specific data catalog implementations, focusing instead on creating a flexible solution that can be adapted to different systems with minimal changes.

3.2.4 Provenance Handling Agent

The provenance handling agent is the component that makes the connection between the data catalog and the blockchain. We considered several options for the data capture task, such as storing all data on the blockchain or employing off-chain storage for some portions of the data. Ultimately, we decided to use a segmented approach that stores only core data on the blockchain, addressing concerns related to storage costs and scalability while still ensuring data integrity.

The component is comprised of several tasks, namely *capture*, *verification*, *publish*. These tasks are discussed in detail in the next paragraphs, addressing them sequentially in the order they are executed.

Data capturing. The data segmentation task is used to manage on and off-chain storage. As mentioned in [Subsection 2.5.3](#), on-chain storage involves storing data directly on the blockchain, where it is secured by cryptographic algorithms and distributed across the network. However, we recall that storing such data can be expensive ([Subsection 2.3.2](#)). The goal of this task is to segment the information so that only *core data* is stored on the blockchain.

Within the scope of this thesis, the term “core data” refers to provenance information. At a low granularity level, this involves the provenance information related to a dataset, achieved by utilizing the well-established Data Catalog Vocabulary (DCAT) specification [\[w3c\]](#) as data model. DCAT provides a standardized model for describing and sharing datasets within data catalogs across various organizations and domains.

At a high granularity level, core data encompasses not only provenance information of the dataset but also that of individual data entities, in that way offering a more comprehensive view of origins and transformations of the data.

Depicted in [Figure 3.2](#) is a visual representation of the dataset storage concept in our solution. As shown, a hash value is generated from all of the retrieved data. Subsequently, the core data, in conjunction with the generated hash value, is transmitted to the smart contract that resides on the blockchain.

Data provenance. The primary goal of this task is to communicate the provenance information to the smart contract. Furthermore, the mapping to every one of the functions and storage points available on the SC is handled. Additional information about concrete functions and storage points are detailed in the following chapter.

To simplify the view of provenance in this thesis, we consider that provenance data is organized into *stories* about an object (either a dataset or an entity) with each modification being regarded as a distinct chapter within the story.

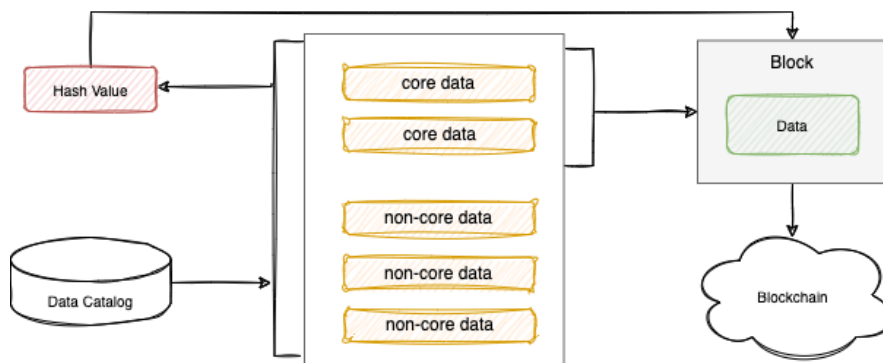


Figure 3.2: On-chain storage overview.

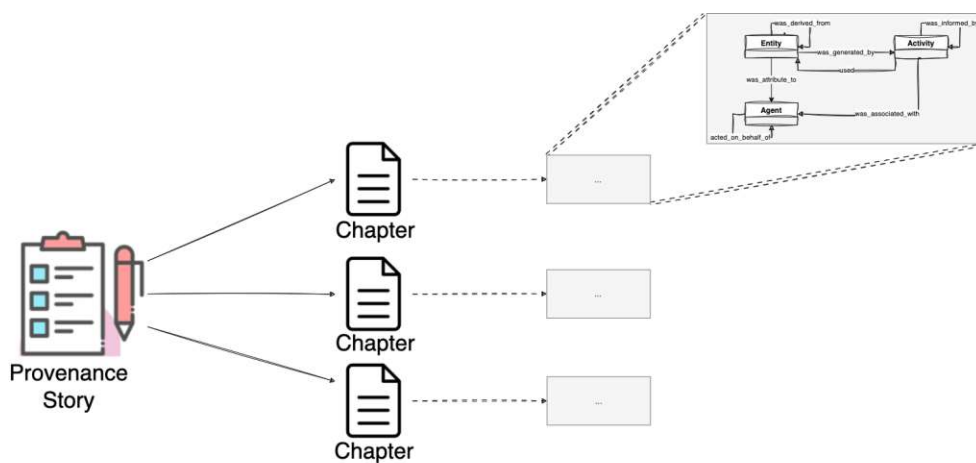


Figure 3.3: Structure of provenance as a story.

A visual representation of this concept is depicted in [Figure 3.3](#). As seen, the *provenance chapter* corresponds to the W3C PROV core structures such as entities, activities and actions. On every update a new *provenance chapter*, representing a new version, is constructed and appended to the provenance story.

Data verification. In this context, the verification task refers to confirming that the new entry extends the previous version of the same dataset. When a dataset is updated, its contents may be modified or manipulated in various ways, which can potentially compromise the integrity or accuracy of the data. By storing a hash of the previous version of the dataset, the agent is able to verify that the data has not been tampered with or modified in an unauthorized manner during the update process.

Implementation of this verification procedure is done by a simple method which ensures that the previous version hash corresponds to the one we are updating in the blockchain storage. For this to work, the “PrevHash” value is validated against the “Hash” value stored in the blockchain. If the hashes do not match, then it indicates that the data has

been tampered with and the update should be rejected.

As only the data catalog has access to the “PrevHash” value to be provided to the story, this approach ensures the integrity of data passed to the agent and also prevents faulty data from entering the blockchain.

3.2.5 Blockchain

The distributed ledger plays a vital role in our system, as it accounts for storing the provenance information and the ability to retrieve it at a later stage. As can be seen in [Figure 3.3](#), a smart contract is deployed to the blockchain. This SC that contains a model of the PROV schema (story chapters), a reference to the corresponding dataset and a partial model of the DCAT schema used for on-chain dataset storage.

Furthermore, when selecting a smart contract technology for our system, it is important to consider compatibility with the underlying blockchain platform, scalability to accommodate growth in users and transactions, security features to maintain data integrity and trustworthiness, flexibility to support various data models, and interoperability to enable seamless interaction with other systems and platforms. These factors play a crucial role in the overall architecture design and performance of the solution.

3.2.6 GUI

A graphical user interface provides a visual representation for interacting with complex systems in a user-friendly manner. In our context, a GUI is used to display information and communicate with the smart contract, allowing users to view information about the contract, input data, and trigger actions. By providing a simple and intuitive communication layer for interacting with smart contracts, the GUI provides an intuitive approach for analytical purposes.

3.3 Summary

In summary, the provided architecture in this section acts as a building ground for the technological choices made in the following chapter. The design is based on extensive research, including a focus on state-of-the-art data catalogs and their technical capabilities.

The proposed architecture consists of several individual components, including a data source, a data extraction agent, a provenance handling agent, a data management platform, a graphical user interface, and a smart contract deployed on the blockchain. This system is designed to provide efficient storage and management of provenance information for cataloged open data in a tamper-resistant manner.

Further functionality regarding extending these components, e.g. extending the provenance handling agent by a task for automatically deploying smart contracts for individual data sets, can be found in [Chapter 6](#).

Implementation

To demonstrate and evaluate the effectiveness of the proposed solution, we provide a PoC implementation. In this chapter, details regarding the implementation process, chosen technology stack, and outcome are presented. To gain a deeper understanding of the implementation, class diagrams, code snippets, and clear explanations of the code's functionality are employed. Decisions made in this chapter are partially based on the outcome of the research conducted in [Section 2.5](#).

In the first section, [Section 4.1](#), a comprehensive documentation of the technology stack of choice is given. Technological choices are further enforced by arguments about the optimality of this decision. Later, in [Section 4.2](#), we provide a thorough description of the smart contract functionality and storage model. Lastly, in [Section 4.3](#) we review the implementation process of the two middleware agents, based on our solution architecture.

4.1 Technology Stack

A primary goal of the technology stack is to be able to fulfill the demands based on the architecture structure together with the functional and non-functional requirements, as defined in [Section 3.1](#). Having flexibility for future extensions in mind, popular modern technological approaches are prioritized. In addition, in some cases miniscule precedence is given to languages with which the developer is familiar with. The reason for this is to save resources in terms of time of implementation.

4.1.1 Blockchain

The choice of blockchain was one of the most impactful decisions in our work. As such, detailed research was conducted on the matter, resulting in the decision to follow a high-level taxonomy ([Subsection 2.5.3](#)) in making this choice.

In our context, an essential requirement is that the blockchain has the capability to facilitate smart contracts, as the application's architecture highly depends on this functionality. For providing greater flexibility and versatility, it is expected that the smart contract feature will be able to support Turing-complete languages.

Additionally, the blockchain must be permissionless, which is crucial for ensuring the integrity of data. In a permissioned blockchain, not everyone has access to the network, which means that not everyone can verify the accuracy of the data [XWS⁺17]. As data integrity is a primary goal of our system, it's crucial to utilize a blockchain, where anyone can access and participate in the network in order to validate the data.

Another decisive criteria for a choice of blockchain is public accessibility. An open-source blockchain allows for greater transparency and accountability by enabling users to review and validate the code themselves. It fosters collaboration and innovation as developers can contribute to the codebase, fix bugs, and add new features. Moreover, an open-source blockchain reduces the risk of vendor lock-in and ensures that the system remains accessible and adaptable over time.

Lastly, the blockchain maturity is considered. Using a system that has persisted through challenges, such as extensive development, testing, and deployment procedures, is evidence to guarantee its stability, security, and dependability. In addition to functionalities being refined through years of use and experience, by using such a blockchain, we ensure that the demands of real-world applications can be met. Furthermore, a mature blockchain should have a well-established ecosystem of developers, and stakeholders who have contributed to its advancement and adoption.

Based on the above considerations and endorsed by the taxonomy evidence of (de)-centralization, storage, computation, and configuration, the candidate which stood out and would allow for optimal software development of our system was the **Ethereum** network.

In addition to performance and cost efficiency as driving factors in this decision, previous knowledge of working with this blockchain was also a factor. Experience in implementing solutions on the blockchain meant that we can save time on implementation and concentrate on analyzing the solution. As mentioned previously, transactions are not free on the Ethereum blockchain, partly enforcing Turing-completeness. As such, a detailed analysis is able to be conducted to evaluate how gas costs alternate in different scenarios based on our proof of concept implementation.

Smart Contract

With regard to smart contracts, the first thing we consider is the choice of **execution clients**. The execution client in ethereum listens to new transactions broadcasted in the network, and holds the latest state and database of all current Ethereum data. Additionally, these clients allow for execution of smart contracts and contain an instance of the Ethereum Virtual Machine.

Multiple execution clients are available for Ethereum, each based on a distinct programming languages. Geth, also known as Go-ethereum is a popular choice written in Go. A client written in Rust is available - OpenEthereum (previously Parity Ethereum). An implementation written in C++ exists, namely Aleth, however it is no longer maintained. Lastly, pyethapp (pyethereum & pydevp2p) is a python implementation of an ethereum client.

From the above clients, one stands out in terms of popularity and resilience and that is **Geth**. Geth was one of the original Ethereum implementations, which has grown in time by popularity, making it the most tested client. Due to these factors Geth was appended to our technology stack.

Next we consider **programming languages** for writing smart contracts. The choice to use Ethereum as a blockchain puts bounds on the available programming languages for implementing this solution. Upon considering several factors, the language chosen for implementing the smart contracts in this project is **Solidity**.

This choice is largely based on the vast community support and abundance of developer tools available for this language. Notwithstanding the tremendous popularity, according to a 2018 survey [NKS⁺18], many solidity contracts had a severe vulnerability during development. To decrease the likelihood of such security breaches, in our contract, we use the latest stable version of the language together with the latest development standards.

To streamline the development process, we selected **Truffle** as a comprehensive toolkit for smart contract development. Truffle is an active community end-to-end-development framework which provides automation for essential default tasks. The framework comes with tools for building, testing, debugging, and deploying Smart Contracts with Solidity. One tool in the truffle suite we make use of is **Ganache**, being a command line tool for personal blockchain deployment and testing.

Every truffle project comes with a predetermined folder structure with the following items:

- **contracts/**: Directory for Solidity Smart Contracts.
- **migrations/**: Directory for scriptable deployment files.
- **test/**: Directory for test files for testing your application and contracts.
- **truffle-config.js** : A Javascript file containing required configuration options.
- **build/**: Directory for the built contracts. The build directory becomes available once we generate our contracts and gets updated if changes on the contracts are made.

We have successfully defined an optimal base for the smart contract functionality, next we explore implementation regarding the blockchain communication layer.

The Vue client in our project acts as a GUI for interacting with the blockchain, more concretely - the smart contract. The client uses **Vue.js**¹, a frontend framework as a backbone. The client can be accessed by using any modern web browser. The choice for using this framework was solely based on previous experience of working with Vue. However, implementing a similar solution in other frameworks or even in native Javascript is indeed possible.

Furthermore, in this project we make use of **Vuex**, which is a state management system specifically designed for use with the Vue.js framework. This state management system allows us to implement shared states (data) across all components. In our solution this data includes the provenance contract address, used in different components for communication purposes. Data which is utilized on component-level, e.g. provenance chapter details, is not added to the shared state.

Additionally, we are utilizing the **Metamask**² browser extension in order to connect to the Ethereum blockchain. In addition to enabling key management and transaction signing directly through the browser, Metamask also provides the necessary tools for connecting to the blockchain for applications running in the browser, as is the frontend of our solution.

Having defined the technologies for implementing the blockchain, the SC and the blockchain communication layer, next we consider the tech stack for the two middleware agents.

4.1.2 Middleware agents

As described in [Section 3.2](#), in our architecture two middleware agents exist, each performing specific tasks to meet the functionality needs. Even though the tasks of each agent differ functionality-wise, there is some overlapping in the underlying technologies used. Therefore, in this subsection we won't differentiate between the different agents. We lay out a technological foundation, the utilization of which we will later describe each agent individually.

The first thing we consider for development of data handling middleware agents is the choice of programming language.

In the context of developing middleware clients, a significant amount of functionality needs to be supported. Additionally, it's necessary for the middleware to be both portable and scalable. To fulfill these needs, a more advanced programming language is being sought after. Based on these requirements, **Python** is chosen as programming language for both middleware agents. Python allows for smooth portability and efficient programming capabilities. Lastly, previous knowledge of working with the python language serves to validate our decision.

¹Vue.js is an open-source front end JavaScript framework used for building user interfaces and single-page applications. Further details - <https://vuejs.org/>

²MetaMask is an extension for accessing Ethereum enabled distributed applications in your browser. Further details - <https://metamask.io/>

Once defined the programming language, there are additional technological decisions to be made. Due to the fact that each agent performs multiple different workflows, the necessity of a platform for managing and scheduling these workflows is evident. Furthermore, by being able to schedule tasks as parts of different workflows, our system will be able to prevent incidents related to synchronization issues.

Apache Airflow³ is a platform for programmatically authoring, scheduling and monitoring workflows. This platform is a great fit for our solution as one of the key use cases of Airflow is to extract data from various sources and publish it to multiple destinations. This is achieved by defining a Directed Acyclic Graph (also known as DAGs). DAGs specify the tasks to be executed, their dependencies, and the logic for their execution. Using Airflow DAGs, data can be extracted from sources such as databases, APIs, and file systems, transformed using various tools such as Pandas, PySpark, or SQL, and published to destinations such as data lakes, databases, or message queues. The system also has a Python operator⁴, used to execute Python callables which is an excellent match for the operational needs defined.

Example 8: Python task in Airflow

For example, the *PythonOperator* can be used to execute Python code that extracts data from APIs or databases, transforms the data into RDF format and publishes the data onto a data catalog.

One last thing to consider when choosing the middleware technology stack is data models to use for encoding information. As previously mentioned in [Subsection 2.2.1](#), we have extensively researched a standard for saving the provenance information, we reached the conclusion that **PROV** is the best choice for our use case. However, a provenance model is not the only one we need for reaching a global solution for storing provenance information.

To enforce availability of provenance information on a dataset level between multiple data catalogs, it is necessary to use a supported model for a dataset. For this we have chosen to use the **Data Catalog Vocabulary** (DCAT), as it provides a standard way of describing metadata about datasets that can be published as linked open data. Furthermore, DCAT specifies a framework for describing the relationships between datasets and for creating hierarchies of datasets based on their metadata properties. Such properties, as is the case in our context, may also include provenance data.

In this subsection we have successfully defined the necessary technological solutions for implementing two middleware agents, next we explore such solutions w.r.t the last component in our architecture, the data catalog.

³<https://airflow.apache.org/>

⁴<https://airflow.apache.org/docs/apache-airflow/stable/howto/operator/python.html>

4.1.3 Data Catalog

Earlier, in [Section 2.4](#), we explored several modern data catalogues and compared them by multiple characteristics. There are very few concerns when it comes to choosing a data catalog for our solution, as we aim to support most modern data catalogs. Therefore, priority is given to the solution which would be the fastest for implementation as a proof of concept.

For this we chose the **CKAN** data catalog as one of the key features of CKAN is its native support for RDF. CKAN's RDF support allows metadata about datasets to be described using RDF vocabularies, such as DCAT, and for datasets themselves to be published as RDF. The use of DCAT and W3C PROV, both RDF vocabularies, streamlines our development process by allowing us to maintain a consistent format throughout the PoC implementation. This simplifies the overall development workflow. In addition, due to the native support for the models utilized by our solution, further analytics of the performance of our solution are possible.

It is important to note that although most reviewed data catalogs don't have native support for the RDF data model, they can store Linked Data in other widely-supported formats. One such format, tailored for use with the JavaScript Object Notation (JSON) is JSON-LD⁵ (JSON for Linking Data), which is a lightweight syntax for expressing RDF data in JSON. This format enables easy integration of Linked Data with JSON-based applications and services, in our case different data catalogs.

There are multiple different ways for installing CKAN for local development, however the CKAN development team recommends using the Docker image⁶ for development purposes.

Docker⁷ is a platform for developing, shipping, and running applications using containerization technology. Containerization allows you to package an application and all its dependencies into a single container that can run consistently across different computing environments.

As CKAN is utilized as a out-of-the-box solution in our system and there is no further development needed for using it in our system, thus we do not document an implementation process.

4.2 Smart Contract

In this section, the different aspects of the smart contract implementation process are presented.

Initially, we explore the underlying contract design together with specifics about the deployment process. Next, we explore different data models that are utilized in our

⁵<https://json-ld.org/>

⁶<https://github.com/ckan/ckan-docker>

⁷<https://www.docker.com/>

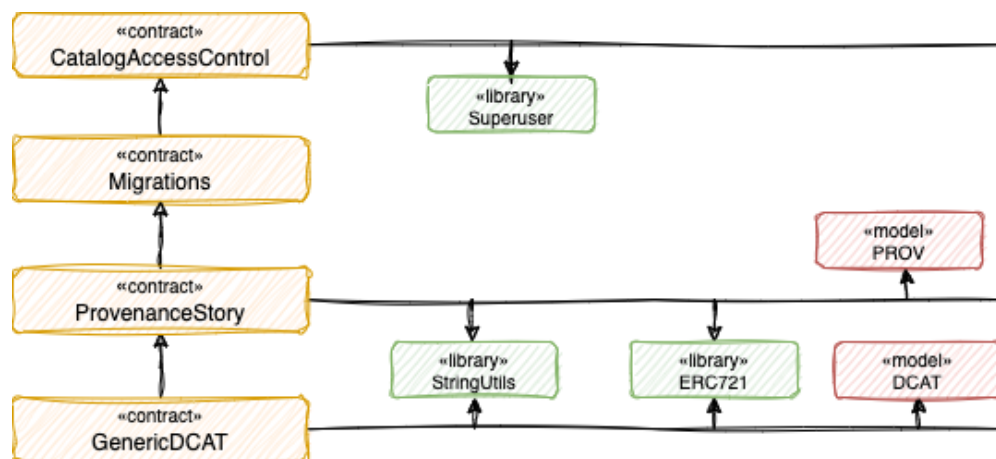


Figure 4.1: Contract Architecture.

solution. Lastly, details about the frontend implementation, namely the Vue client, are provided.

4.2.1 Design Structure

In this subsection the foundation of the provenance smart contract is presented. We show the underlying contract structure and how necessary libraries are linked in our solution.

The contract architecture is shown in [Figure 4.1](#), revealing the fundamental design of the contract. Multiple smart contracts are highlighted in yellow to showcase how they add functionality to the final solution through inheritance. The green color illustrates the libraries that are utilized, while the red color represents the two models used for storing data. The architecture is based on best practices when building an Ethereum SC, provided from the ETH community and documentation.

First we go over the libraries chosen, namely the *ERC-721* token standard, *StringUtils* and *Superuser*. As mentioned in [Subsection 2.3.3](#), the *ERC-721* token standard provides our SC with a template containing a certain set of standardized functions for non-fungible assets. Furthermore, by using this template we ensure a standardized approach to saving the provenance data. The *StringUtils* and *Superuser* libraries provide us with standard functionality for basic string utilities and for access control management in order to restrict some of the SC functionalities. In addition, library functions are not accessible to the open public and cannot be called by transactions. This further ensures security when implementing functionality where libraries are utilized.

Next we discuss the different contracts that build up our solution:

GenericDCAT. This contract represents a basic storage container for the dataset metadata. In extension to additional information, this contract stores one of the most important variables, namely the dataset hash value. This value is later checked against

when updating provenance information and verifies the updater’s legitimacy. Furthermore, the contract utilizes the DCAT model for on-chain storage of dataset provenance information.

ProvenanceStory. This contract contains functionality to maintain provenance information in the shape of provenance stories. As we saw earlier, in [Figure 3.3](#), a provenance story is organized in chapters utilizing the PROV model. The most important functions for *creating* and *updating* provenance information regarding a dataset are provided here. Furthermore, these stories take the form of an ERC-721 token, ensuring data consistency.

Migrations. Because of the fact that our solution has to be build to last, one concern is to provide a way to extend the contract functionality and overcome potential vulnerabilities. The migration contract is useful when you want to update the logic of an existing contract without losing the data stored in the previous contract.

Example 9: SC vulnerability migration

A vulnerability is discovered in the ProvenanceStory contract’s functionality. The migration contract is used to create a new version of the ProvenanceStory contract and to move the data from the old contract to the new one.

The above example provides a scenario where a migration contract is utilized to overcome a vulnerability in the SC, however there are multiple additional uses of a migration contract. Such include upgrading contract functionality, optimizing performance and gas usage, and contract duplication (if the dataset is duplicated and provenance diverges).

CatalogAccessControl. The catalog access control contract takes care of accessibility concerns. For instance, as the data catalog needs access to only the “has” value of the dataset, we want to restrict usage of further functionality. In addition, the superuser (owner) is defined here and permission for actions are provided (e.g. to migrate the contract to a newer version).

We further provide information on how these smart contracts are made accessible on the blockchain.

Deployment is the process where a contract is made available to other users on the blockchain. Essentially, deployment of a contract is done by sending an Ethereum transaction containing the compiled byte code of the smart contract, without specifying a recipient. As stated earlier, any transaction in the Ethereum chain costs a certain fee, where additional fees occur based on the transaction size. Therefore, the cost of deploying a contract is based on the complexity of the contract.

In our PoC solution, the compilation of the byte code and the deployment is performed by using the Truffle framework. This process can be configured by using the so-called migration files, stored in the migration folder of the project. Once deployed, truffle provides ten unique accounts (and their private keys) that can be used to interact with the blockchain. We make use of these ten accounts by inputting them into Metamask when using the Vue client.

Now that we have defined the different contracts that build up our solution and the deployment process, we provide information on the data models utilized by our system.

4.2.2 Data models

In this subsection we provide a detailed description on the underlying data models used in our PoC project. As mentioned earlier, the smart contract storage structure is based on two data models, namely W3C PROV and DCAT. As our aim is to provide a proof of concept solution, the implementation is designed to meet only the essential requirements necessary to demonstrate a model representation of the data used.

PROV

Provenance data storage has a leading role in our system solution. As such, greater effort is concentrated on the development of a SC standard, utilizing the PROV model. Due to the extensive scope of the standard, in this subsection we present only the most important functionality.

In [Listing 4.1](#) a simplified gist from the *ProvenanceStory* contract is shown. The internal representation of the contract resembles a story model, comprised out of a mapping to different provenance chapters.

Each chapter consists of a simplified representation of objects as defined in the *W3C PROV* core structures, namely entities, activities and actions. In our context, entities are represented as datasets where the ID is used as the identifier. Furthermore, entity attributes are used, for storage of metadata information such as the dataset hash value. Activities resemble objects that contain information for the occurred event, in our case creation or update of a dataset. The agent takes the form of the person performing the activity. In our solution this is a CKAN user ID, together with an attribute containing the user's organization ID.

Now that we have described the provenance chapter structure, we further discuss the two functions depicted in the listing.

The *createProvenance()* function is used to create a chapter and append it to the chapters mapping. The new chapter is created only if the inputs are valid and follow the correct RDF format. Every time a new chapter is appended to the mapping, the *provenanceIndex* is updated.

Updating provenance information of a chapter is done by using the *updateProvenance()* function. This function acts as a safe guard in the case that incorrect provenance information has entered our data. Moreover, it contains rather complex logic, therefore we provide further context in [Listing 4.2](#).

Updating can be initiated by providing a valid provenance identification (ID) of the story to be updated. Furthermore, the previous hash value of the dataset needs to be provided in order to ensure the integrity of the callee. If the provenance ID doesn't exist or the data hash is not valid, the execution stops and all changes are reverted.

```

1      contract ProvenanceStory {
2
3          struct ProvenanceChapter {
4              uint256 id;
5              uint256 index;
6              uint256[] inputChapterIDs;
7              string context;
8          }
9
10         mapping (uint256 => ProvenanceChapter) private provenanceChapters;
11         uint256[] private provenanceIndex;
12
13         function getProvenance(uint256 _provId) public view returns
14         (uint256 id, ...)
15         {
16             ...
17         }
18
19         function createProvenance(uint256 _provId, ...)
20         internal returns (uint256 index)
21         {
22             ...
23         }
24
25         function updateProvenance(uint256 _provId, ...)
26         internal returns (bool success) {
27             ...
28         }
29     }

```

Listing 4.1: ProvenanceStory contract gist.

Once all checks are passed, the provenance story gets updated accordingly. Furthermore, in Line 17 - 24 an event is emitted. Events refer to the EVM's logging facilities [\[str\]](#). These logging facilities enable the execution of callbacks in systems, triggered by these events. When triggered, events append arguments to be stored to the transaction's log - a special data structure in the blockchain. Logs are associated with the SC which emitted the events. It is also possible to store variables in events by passing them as parameters. There exist a special type of parameters, as depicted in Line 2, marked by the attribute *index*. This type of parameters are not stored themselves, but a hash value of these parameters is stored. Furthermore, these parameters allow for searching capabilities of events, useful when conducting analytics and when providing a visual representation of the data.

In the *UpdateProvenance* event, the indexed parameter is the story ID. In our context, this can be used for updating information on the frontend as well as for the data catalog to be able to retrieve the dataset hash value at a certain provenance story chapter. In Line 2-7 additional parameters are values, that can be used for verifying the chapter

```

1     event UpdateProvenanceEvent (
2         uint256 indexed provStoryId,
3         uint256 index,
4         string _datasetHashValue,
5         uint256 tokenId,
6         uint256[] inputProvStoryIds,
7         string context);
8
9     function _updateProvenance(uint _provStoryId, uint _tokenId, uint[]
10        memory _inputProvStories, string memory _context, string
11        _datasetHashValue) internal returns (bool success) {
12        require(isProvStory(_provStoryId), "Error: ID doesn't exist");
13        require(provStoryHasHash(_provStoryId, _datasetHashValue), "Error
14            : Hash value doesn't match");
15
16        provenanceChapters[_provStoryId].tokenId = _tokenId;
17        provenanceChapters[_provStoryId].inputProvStoryIds =
18            _inputProvStoryIds;
19        provenanceChapters[_provStoryId].context = _context;
20
21        emit UpdateProvenanceEvent (
22            _provStoryId,
23            provenanceChapters[_provId].index,
24            _datasetHashValue,
25            _tokenId,
26            _inputProvStoryIds,
27            _context
28        );
29        return true;
30    }

```

Listing 4.2: ProvenanceStory: Update function gist.

structure and expose the variable IDs.

Having presented an abstract overview of how the W3C PROV standard is implemented as a model in our solution, next we go over the implementation of the second data model in our system.

DCAT

The data catalog vocabulary provides a set of classes and properties to enable the publication and discovery of data resources. These classes and properties allow for creating a model of the underlying objects, in our context a dataset. By following a separation of concerns [\[DWP.JV02\]](#) design principle.

We have abstracted the logic for DCAT storage into a separate contract, depicted in [Listing 4.3](#). This smart contract provides a simple functionality for storing and retrieving DCAT entities on the Ethereum blockchain. Next we provide details on the underlying contract functionality.

```

1
2  contract DCATEntityStorage {
3      struct DCATEntity {
4          string id;
5          string description;
6          string[] keywords;
7          string[] distributions;
8          uint256[] chapterTokenIds;
9      }
10
11     mapping (uint256 => DCATEntity) private dcatEntities;
12     uint256 private dcatEntityCounter;
13
14     function addDCATEntity(string memory id, ...) public {
15         ...
16     }
17
18     function getDCATEntity(uint256 id) public view returns
19     (string memory, ...) {
20         ...
21     }
22
23     function addProvenanceChapter(uint256 dcatEntityIndex, ...) public {
24         require(dcatEntityIndex <= dcatEntityCounter && dcatEntityIndex >
25             0, "Invalid DCAT entity index");
26         uint256 chapterTokenId = _createChapter(activity, agent, entity,
27             timestamp);
28         dcatEntities[dcatEntityIndex].chapterTokenIds.push(chapterTokenId
29     );
30     }
31 }

```

Listing 4.3: DCAT contract gist.

The *DCATEntityStorage* contract has a mapping (*dcatEntities*) that stores DCAT entities. The keys of the mapping are the SHA-256 hash of the entity identifier, and the values are entity data represented as a strings.

The contract has two functions. The *addDCATEntity()* function takes two arguments: the entity identifier as a *bytes32* value and the entity data as a *string* value. It then performs a check, if the entity data is not empty and if an entity with the same ID does not already exist in the registry. If both conditions are met, it adds the entity data to the *DCATEntities* mapping using the entity ID as a key value.

The *getDCATEntity()* function takes the entity ID as an argument and returns the corresponding entity data as a string. It first checks if an entity with the given ID exists in the storage, and if so, it returns its data.

As provenance tracking is a primary objective of our project, in-depth details on the *addProvenanceChapter()* function are provided.

The function is designed to add a new provenance chapter to a specific DCAT entity within the `DCATEntityStorage` contract. This function accepts four string parameters representing provenance information (activity, agent, and entity) and a timestamp (as an unsigned integer). The main functionality of this function can be summarized as follows:

1. Validate the `dcatEntityIndex` parameter to ensure it is within the range of existing DCAT entities. If the provided index is invalid, the function will revert with an error message.
2. Call the `__createChapter()` function from the inherited `ProvenanceStory` contract, passing the provenance information (activity, agent, and entity) and timestamp. The `__createChapter` function creates a new `Chapter` struct, stores it in the chapters mapping, mints a new ERC-721 token with the new chapter ID, and returns the new chapter's token ID.
3. Retrieve the specified DCAT entity from the `dcatEntities` mapping using the `dcatEntityIndex` parameter, and add the newly created chapter's token ID to the `chapterTokenIds` array of the DCAT entity.

4.2.3 Vue client

The main goal of this client is to be able to retrieve and interact with data stored on the blockchain for presentational and analytical purposes. Multiple conclusions described in are drawn using this frontend.

Based on best practices, we design the client as a single-page application [vue], comprised of the following components:

- *DCAT*: This component provides an overview of a dataset stored on the `DCA-TEntityStorage` contract. The component is used to retrieve and display all public details of the dataset, including *description*, *keywords*, *distributions*, and *provenance chapters*.
- *Story*: This component serves as an overview of the provenance story, it also contains chapters, related to it. A visual representation of this component can be seen in [Figure 4.2](#). Furthermore, this component provides access to the functionality for appending new chapters to the story.
- *Chapter*: The chapter component is used to provide details on different chapters.

The interface of the Vue frontend utilizing the blockchain and containing the above components is designed as follows:

The DCAT component is the main view that displays an overview of a dataset. It can be directly accessed on the homepage of the application. The component presents all public

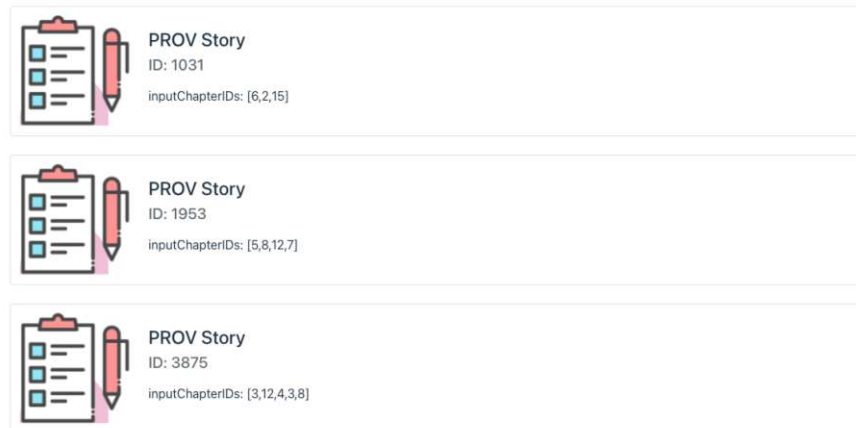


Figure 4.2: Frontend display of Stories.

details of the *DCATEntityStorage* contract, including *description*, *keywords*, *distributions*, and *provenance chapters*. The DCAT component is designed to include tabs or accordions to display the different chapters of the dataset in a user-friendly way. Users can switch between the different tabs to view the information they are interested in.

The Story component serves as an overview of a provenance story. It can be accessed from the DCAT component by clicking on the provenance chapter. The Story component displays the main details of the provenance story, including the *index*, *context*, and *related chapters*. The component also provides access to the functionality for appending new chapters to the story. Users can click on the “Add Chapter” button to open the Chapter component and create a new chapter for the story.

The Chapter component is used to provide details on different chapters. It can be accessed from the Story component by clicking on a chapter. The Chapter component displays the details of the chapter, including information on the *subject*, *predicate*, and *object*. Furthermore, users can view the transactions associated with the chapter by clicking on a link to the transaction on the blockchain explorer. The Chapter component also provides create, read, update, and delete (CRUD) functionality for performing actions on this component.

4.3 Middleware Agents

Our architecture utilizes two middleware agents with the aim of retrieving, processing, and delivering data to different parts of our system.

This section provides an overview of the implementation of these middleware agents and is split as follows. In [Section 4.3](#) we outline the process of how heterogeneous data is extracted and delivered to our data catalog by the *data extraction agent*. Then in

[Section 4.3](#) we show the implementation of the *provenance handling agent* and revise the underlying functionality.

As mentioned in [Subsection 4.1.2](#) we use Apache Airflow for the implementation of the middleware agents. Furthermore, we follow Airflow’s DAG structure as a collection of tasks arranged in a logical sequence, where each task represents a specific action that needs to be executed. In addition, task dependencies and the Python functions to be executed by each task are provided.

Once we have defined the DAG, the execution of the tasks needs to be scheduled. Scheduling of the tasks is done by the Airflow scheduler itself, which is a separate process that needs to be started in order to run the DAG at specified intervals.

To initiate scheduling for the DAG in the listing, we need to save the code as a Python file in an Airflow DAG folder, and then start the Airflow scheduler by using the *airflow scheduler* command in the terminal or command prompt. Once the scheduler is running, it will automatically execute the tasks in the DAG according to their dependencies and the specified schedule interval.

In addition, the latest version of Airflow provides a GUI for scheduling tasks where no interaction with a terminal or command prompt is needed.

Data extraction agent middleware

The data extraction agent is responsible for automatically extracting data from different resources and passing them to the data catalog component of our system. We can see a gist of our implementation in [Listing 4.4](#), which shows an Airflow instance containing two DAGs and their implementation.

Initially, in rows 1-4, multiple libraries are loaded. The most important of which for our solution are the Python Data Analysis Library (pandas), boto3, and requests. PANDAS is the library responsible for reading and writing to a *.csv* file containing our dataset, while requests establishes a way for communication with the data catalog. Lastly, for the PoC, all datasets are stored in a cloud storage, more specifically an Amazon S3 bucket⁸, where boto3⁹ is the client responsible for providing access to the bucket.

The DAG we implement defines two PythonOperators: *create_new_dataset* and *feed_to_ckan*. The *create_new_dataset* function creates a new boto instance and retrieves the datasets from the storage bucket. The data is then passed to a Pandas DataFrame where the dataset fields are extracted, and saved to a local *.csv* file.

The *feed_to_ckan* function loads the new dataset from the CSV file, prepares the data for the CKAN API, and posts it to the data catalog using the CKAN API.

⁸<https://aws.amazon.com/s3/>

⁹<https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>

4. IMPLEMENTATION

```
1 ...
2 import pandas as pd
3 import requests
4 import boto3
5
6 def extract_s3_data():
7     # Create an S3 client
8     ...
9
10    # Retrieve the list of objects in the S3 bucket
11    ...
12
13    # Iterate over the objects and extract the data
14    ...
15
16 def feed_to_ckan():
17     # Load the new dataset from the CSV file
18     ...
19     # Get the variable from XCom
20     ...
21
22     # Prepare the data for the CKAN API
23     ...
24     # Post the new dataset to the CKAN data catalog using the CKAN API
25     ...
26
27 default_args = {
28     'owner': 'author',
29     ...
30 }
31
32 dag = DAG('create-and-feed-to-ckan', ...)
33
34 extract_s3_data = PythonOperator(
35     task_id='create_new_dataset',
36     ...
37 )
38
39 feed_to_ckan_task = PythonOperator(
40     task_id='feed_to_ckan',
41     ...
42 )
43
44 extract_s3_data >> feed_to_ckan_task
```

Listing 4.4: Data extraction agent middleware.

Furthermore, the DAG can be scheduled to run at a defined period, in our case once a day as defined by `schedule_interval=timedelta(days=1)`. Lastly, in Line 44, the two tasks are connected with a simple dependency which defines the execution sequence.

We have seen our first middleware implementation, containing basic functionality for handing data. Next we explore the second middleware in our solution, responsible for handling provenance information

Provenance handling agent middleware

This middleware is responsible for retrieving information from the data catalog, verifying the legitimacy of the request, segmenting data for on-chain storage, and publishing this data together with the corresponding provenance information to the SC. All of the described functionalities are split into separate tasks, performed by the agent. Furthermore, the tasks are chained in order, so that if one task fails execution of consequent tasks is suspended.

Similarly to the previous middleware agent, Airflow is utilized as a system for the implementation of these tasks. In [Listing 4.5](#) a listing is depicted containing an overview of the implementation.

In this DAG we utilize several libraries for communication and management of data. Essential segments of these libraries are imported at the beginning of the snippet. In the listing, the most important libraries for implementing our solution are shown, namely *ckanapi*, *harvesters*, *rdflib*. Next, we describe the functionality that each library provides:

- **ckanapi:** The *ckanapi* library is a Python package that provides a convenient way to interact with a CKAN data portal using the CKAN API. It allows for programmatic access to CKAN resources such as datasets, organizations, and users, as well as perform various operations such as creating, updating, and deleting resources. The library provides a simple and consistent interface for accessing CKAN resources, and handles tasks such as authentication, error handling, and rate limiting automatically.
- **harvesters:** The *kanext.dcat.harvesters* library is a Python package that provides a set of tools for harvesting and importing metadata from external sources into a CKAN data catalog. The library is an extension of the *kanext-dcat* plugin for CKAN, which provides support for the Data Catalog Vocabulary (DCAT) specification.
- **rdflib:** The *rdflib* library is a Python package that provides a comprehensive set of tools for working with Resource Description Framework (RDF) data. It allows users to create, manipulate, and query RDF graphs and triples, and provides support for a variety of RDF serialization formats such as RDF/XML, Turtle, and N-Triples. The library includes a number of powerful features such as SPARQL querying,

```
1 ...
2 from ckanapi import RemoteCKAN
3 from ckanext.dcat.harvesters import DCATHarvester
4 from rdflib import Graph, RDF, Namespace
5
6 def log_dataset_update(ds, **kwargs):
7     # get user who triggered the DAG
8     ...
9     # get hash of updated dataset
10    ...
11
12
13 # Define the function that retrieves RDF data from a CKAN data catalog using
14 # the ckanext-dcat library
15 def get_rdf_data():
16     # Connect to the CKAN data catalog
17     ...
18     # Get the DCAT metadata for a dataset
19     ...
20     # Harvest the RDF data from the DCAT metadata
21     ...
22     # Convert the RDF data into an rdflib Graph object
23     ...
24     # Extract the metadata and provenance information using SPARQL queries
25     ...
26     # Convert the results into a dictionary
27     ...
28     return metadata
29
30 # Define the function that passes provenance to a blockchain smart contract
31 def pass_to_blockchain():
32     # Connect to the Web3 provider
33     ...
34     # Set up the contract ABI and address
35     ...
36     # Send a transaction to a contract instance
37     ...
38     # Wait for the transaction to be mined and return the new hash to the
39     # data catalog
40     ...
```

Listing 4.5: Provenance handling agent middleware.

inference, and serialization, and provides for a flexible and extensible framework for working with RDF data.

Now that we have shown what libraries are used to support us in the implementation, we further provide details on the implementation of the different tasks:

- **capture:** In this example, the *log_dataset_action()* function logs information about the action performed, user who triggered it and the hash of the updated dataset. To generate the dataset hash value the function uses the *hashlib* library. The function is triggered every time a dataset is altered.
- **verification:** The *verify_legitimacy()* function provides simple functionality for verification of updated dataset legitimacy. This function provides a means to compare the current version of the dataset with its previous version.
- **on-chain storage:** The *on_chain_storage()* function uses the *ckanapi* library to connect to a CKAN data catalog and the *ckanext-dcat* library to retrieve the DCAT metadata for a specific dataset and harvest the RDF data from it. The RDF data is then converted into an *rdflib* Graph object, and metadata and provenance information can be extracted using a query language such as SPARQL¹⁰.
- **publish:** Lastly, the *pass_to_blockchain()* function uses the *Web3* library to connect to a local Ethereum node, in our case a ganache instance, and pass the provenance data to the smart contract function, *store_provenance()*, for storage on the blockchain.

In this section, we have described the implementation of the two middleware agents. Furthermore, we have provided code snippets displaying the underlying libraries and functions that build up our DAGs. Next, we evaluate the results from experiments conducted on the implemented solution.

¹⁰SPARQL (SPARQL Protocol and RDF Query Language) is a query language and protocol for querying and manipulating RDF (Resource Description Framework) data. Additional information - <https://www.w3.org/TR/rdf-sparql-query/>

Evaluation

In this chapter, we analyze different performance scenarios of the system implementation. An assessment of the proposed solution is conducted, drawing upon a benchmark analysis of the implemented contracts. The evaluation is segmented into two main parts - requirements and quantitative analysis. This approach enables a thorough and objective evaluation of the solution's effectiveness and performance in the given context.

The evaluation chapter is structured as follows: [Section 5.1](#) introduces the analysis of requirements, providing a detailed examination of how the non-functional requirements are satisfied by our solution. Furthermore, in this section, the execution of a use case is presented and a comparison is made with current solutions. Later, in [Section 5.2](#), results with regard to overhead and efficiency costs are discussed. Besides providing information on deployment expenses, we offer various cost models for storage, taking into account both on-chain and off-chain factors.

5.1 Requirements Analysis

Prior to the design and implementation, in [Section 3.1](#), we outlined use cases, functional and non-functional requirements that our system should fulfill. The goal of this section is to evaluate the extent to which use cases and non-functional requirements have been fulfilled. As functional requirements are concerned with the system implementation and covered by the PoC solution we presented in [Chapter 4](#), they are not included in the evaluation.

5.1.1 Use cases

Earlier, in [Section 3.1](#) we defined multiple use cases that our solution implementation must be able to support. The UCs are based on various scenarios, with some demonstrating a more extensive application of our solution's capabilities. The following analysis is

	UC 3	UC 4	Granularity level	Multi-Domain	Transparency
ODBI	×	✓	high	no	yes
ProvChain	×	✓	low	no	yes
Smartprovenance	×	✓	low	no	yes
Dang et al.	×	✓	high	yes	no
ProvHL	×	×	low	no	no
Our Solution	✓	✓	high	yes	yes

Table 5.1: Comparison between our solution and related work.

centered on two primary use cases, which show the most comprehensive utilization of our solution’s features. This allows for a focused evaluation of the proposed solution’s effectiveness within the context of its most demanding applications.

UC Choice. The selection of use cases for evaluation and comparison across different solutions is done by determining the ones that have the highest utilization of our system functionality. For this, we use the “weighing” methodology defined by Gustav Karner, as detailed in [SW01], which allows us to assess the complexity (highest system utilization) of use cases. The basis for evaluating UC complexity depends on the number of transactions involved in a use case. A transaction refers to an indivisible set of activities that are executed either in their entirety or not at all. In the current context, activities are considered as system functionality.

Upon applying this methodology on our use cases, the most complex cases were found to be UC 3 and UC 4.

A primary difference is that these use cases operate with provenance information on a high-level of granularity, increasing the number of transactions involved. Moving forward, we proceed to evaluate the extent to which our system can effectively address the requirements and challenges presented by these particular scenarios.

Execution Comparison. We proceed to evaluate the results from executing UC 3 and UC 4 across the state-of-the-art as identified during the research done on the related work, found in Section 2.5.

In Table 5.1, we present a comprehensive comparison between various solutions addressing these particular UC scenarios. In addition to comparing the feasibility of the different solutions to execute these use cases, we add three additional properties - granularity level, multi-domain support, and transparency. The objective of this approach is to offer a deeper understanding of how individual use case transactions perform across various solutions. We proceed to individually examine the results related to the use cases.

As defined earlier in this subsection, use case complexity is evaluated by the number of activities to be performed, i.e. transactions.

In **UC 3**, there are three primary transactions: (1) data storage, (2) high granularity provenance storage, and (3) transparency and traceability. (1) is effortlessly addressed by each solution, as it represents fundamental prerequisites for a functional approach. In the case of (2), however, the compared solutions exhibit varying abilities to address this particular challenge. Most solutions do not accommodate this transaction due to their lack of support for storing provenance data at a high granular level. The solution by Dang et al. is the only one from the related work, which provides high granularity support. Therefore, we assess the third (3) and last transaction - transparency and traceability. While traceability is inherently achieved due to the blockchain's traceable nature, transparency encompasses a more extensive range of considerations.

One such consideration is transparency concerning the blockchain. Two of the compared solutions utilize a proprietary blockchain, namely HF. Hyperledger Fabric, is a permissioned blockchain, typically used for private, consortium-based networks where access and participation are controlled. Utilizing a private network reduces the level of transparency, which is a critical aspect when considering solutions for global data provenance traceability.

Further arguments on the advantages and disadvantages of using Ethereum or HF have been made by, Dinh et al. in **[DLZ⁺18]**. The authors assert that Hyperledger surpasses Ethereum in terms of performance. Nevertheless, it experiences failure when expanded beyond 16 nodes. Moreover, Ethereum exhibits enhanced robustness when confronted with node failures.

Having successfully compared the first use case, we proceed with the evaluation of the second scenario.

In **UC 4**, there are two primary transactions - provenance data retrieval and versioning. As all solutions utilize a specific blockchain, we can presume that the versioning requirement is met, given the unchangeable historical records inherent to ledgers. Regarding the retrieval of provenance data, ProvHL stands as the sole solution incapable of supporting this use case. This limitation stems from the fact that this solution is designed exclusively for internal automated programmatic access to the provenance data stored on the blockchain.

The comparison revealed that most solutions lacked high granularity support, with Dang et al.'s solution being the exception. Additionally, some solutions exhibited lower transparency levels compared to others, leading to a failure in addressing one of the use cases. We proceed with the evaluation of non-functional requirements.

5.1.2 Non-functional requirements

In this subsection, we will examine the extent to which our proposed solution fulfills the non-functional requirements previously identified in **Subsection 3.1.3**, more concretely - reliability, security, capacity, and extensibility. The goal is to provide a clear understanding of the solution's effectiveness in meeting the requirements criteria.

Requirement	Data Sources	DEA	PHA	Data Catalog	Blockchain	GUI
Reliability	✓	✓	✓	✓✓	✓✓✓✓	✓
Security	✓	✓	✓✓✓	✓✓	✓✓✓✓	✓
Capacity	✓	✓✓✓	✓✓✓	✓	✓✓✓	✓
Extensibility	✓	✓✓✓	✓✓✓	✓✓	✓✓✓	✓

Table 5.2: Impact of solution components on non-functional requirements. (✓: Minimal impact, ✓✓: Low impact, ✓✓✓: Moderate impact, ✓✓✓✓: High impact)

Given that our system solution consists of numerous components (as defined in [Figure 3.1](#)), it is crucial that each requirement is assessed in relation to the components it influences, ensuring a thorough understanding of its effects on the overall system performance and functionality.

In [Table 5.2](#), we provide an overview of these components, evaluated by the extent of their influence on the non-functional requirements. The outcomes depicted are derived from elements identified during the process of creating the solution.

Impact levels range from minimal impact to high impact, where minimal impact indicates a negligible or slight effect on the requirement, while high impact signifies a substantial influence on the requirement’s fulfillment or performance. Based on this outcome, in the next paragraphs we explore each individual requirement. The analysis will focus solely on the components that have moderate to high impact on the respective requirement.

Reliability. The primary goal we strive to achieve in ensuring our system’s reliability is to safeguard against data loss or corruption.

By utilizing *blockchain* technology as a distributed ledger for storing provenance information, we allow for enhanced data integrity. Due to the transparent and immutable nature of blockchains, any tampering attempts can be easily detected and prevented. Furthermore, by choosing the Ethereum blockchain for our solution, we employ the PoS consensus mechanism. This ensures that all nodes agree on the contents of the blockchain, providing a secure and reliable method for maintaining data consistency and preventing potential attacks.

Moreover, *smart contracts* play a crucial role in automating provenance management processes, ensuring the accuracy and consistency of data. The contract architecture is based on best practices for building Ethereum smart contracts, provided by the ETH community and documentation. The use of libraries, such as ERC-721, StringUtils, and Superuser, offers standard functionality for non-fungible assets, basic string utilities, and access control management. In this way accessibility concerns are tackled, ensuring that only authorized users can access certain functionalities. This ensures a secure and reliable implementation of the system. Additionally, the inclusion of migration contracts enables the system to be updated, extended, or optimized when necessary, which is crucial for long-term reliability. Migration contracts can be used to address vulnerabilities, upgrade

functionality, optimize performance and gas usage, or duplicate contracts when dataset provenance diverges.

Security. The main aim when addressing this requirement is to ensure the security and privacy of sensitive information and prevent unauthorized access, modification, or deletion of data. Our system utilizes various approaches to tackle these issues, which are present in the following components:

- **Provenance Handling Agent.** The agent is responsible for managing the provenance data within the system. By using a dedicated agent, we ensure that only authorized users can create, or modify provenance information. This access control mechanism helps prevent unauthorized access to sensitive data, enhancing the security of the system. The agent is responsible for managing and distributing the provenance data within the system. Before allowing updates to the provenance data, the agent compares the previously generated hashes against the ones stored on the blockchain. This comparison ensures that the update is coming from a valid source (data catalog) and helps maintain the integrity of the provenance information. By verifying the source of updates, the agent prevents unauthorized changes and tampering, further enhancing the security of the system.
- **Blockchain.** The Provenance Handling Agent interacts with smart contracts on the Ethereum blockchain to manage provenance data securely. Smart contracts enforce the rules and conditions governing data access and modification, ensuring that the data remains secure and consistent. By working in conjunction with smart contracts, the agent helps maintain the security and integrity of the provenance data.

Capacity. Addressing system capacity is a significant challenge for developers when creating solutions that leverage blockchain technology. Our aim in satisfying this requirement is to provide a solution to store and process extensive amounts of data efficiently. In this paragraph, we do not distinguish between distinct system components, since the approach employed to address this requirement is distributed across several components.

The *Provenance Handling Agent* is responsible for extracting and processing the necessary metadata and provenance information from the data catalog. By streamlining the extraction process and focusing on the essential information, the agent reduces the amount of data that needs to be stored and processed on the *blockchain*. By utilizing the on and off-chain storage methods (see [Chapter 3](#)), we are able to effectively manage the data flow through our system.

The on-chain storage, in this case, the Ethereum blockchain, is used to store crucial metadata and provenance information securely. However, storing large amounts of data on the blockchain can be costly and slow, which is why our solution also utilizes off-chain storage. By adopting the off-chain storage method, our system is capable of managing

large volumes of data more efficiently. This approach enables the system to handle large volumes of data without incurring high costs or slowing down the blockchain.

Extensibility. The assessment of extensibility should concentrate on the system's capability to preserve data integrity as it manages a growing volume of data and accommodates a variety of data catalogs.

A primary concern during the development of the architecture, we made it as easy as possible for future extension of the code base to take place. This is done by utilizing a modular approach, allowing developers to easily add, modify or replace components as needed. This ensures that the system can evolve over time to accommodate new requirements, technologies, or standards.

Furthermore, by using widely-accepted data models such as *W3C PROV* and *DCAT*, the system ensures interoperability with other systems and simplifies the development process. By using the W3C PROV model, the system can consistently and effectively manage provenance data while maintaining compatibility with other provenance-aware systems and tools. Moreover, by using the DCAT model, the system can effectively manage and exchange metadata about datasets, facilitating interoperability with other data catalogs and systems. In this way we address one of the primary research objectives, namely data catalog interoperability.

5.2 Cost Efficiency

In this section, we provide a comprehensive analysis of the costs associated with preserving provenance data on the blockchain. To achieve this, we initially evaluate the deployment cost of our smart contract. Then we evaluate the costs associated with storing diverse amounts of provenance data. We explore differences in storing solely DCAT data on the blockchain (and off-chain provenance) as opposed to storing a combination of DCAT and provenance data on-chain.

5.2.1 Deployment cost

The primary expenses to set up a working version of our solution is the smart contract deployment cost. The monetary cost of the deployment depends on the market conditions. In this subsection we will cover the costs in terms of gas.

There are a total of six components that determine the amount of gas required to deploy the contract:

- A fixed cost of 21,000 gas for triggering an Ethereum transaction.
- A fixed cost of 32,000 gas for the creation of a new contract.
- Each storage variable set incurs a cost of 22,100.

- Transaction data incurs a cost of 4 gas for each zero byte and 16 gas for each non-zero byte.
- An expense associated with executing each bytecode during the initialization process.
- A fixed cost of 200 gas per byte of deployed bytecode.

Based on this cost structure, the deployment of our contract currently required 5.3 million in gas expenses.

5.2.2 Storage cost

Earlier, in [Subsection 2.3.3](#), we explored multiple methods for storing data in Ethereum. In our approach, we employ two of these techniques, namely events and storage within smart contracts. We proceed to evaluate the efficiency of these two methods in our solution.

SC storage. As mentioned in [Subsection 4.2.2](#), we utilize two storage models in our solution, namely DCAT and W3C PROV. As the DCAT model contains low granularity details on a dataset level, in our solution it is always stored on-chain. On the other hand, the W3C PROV can become significantly large in size depending on the granularity level, thus we investigated various approaches to store data, specifically focusing on both on-chain and off-chain storage solutions. Based on this, in our tests we explore multiple different scenarios of smart contract storage: DCAT and off-chain provenance, DCAT and low granularity on-chain provenance, DCAT and high granularity on-chain provenance.

In the case of *DCAT and off-chain provenance*, we store only a hash value of the provenance on-chain. As we can see in [Figure 5.2](#), this results in a constant storage cost. The main cost factor associated to this scenario is storing the DCAT dataset metadata. As dataset metadata size does not exceed standard best practices, in our tests we set an upper bound on the size at 4Kb (≈ 2000 words). As visible, the biggest transaction has a cost of 2.5MGas. It is important to note that the provided data is within the current maximum allowed gas per block in the Ethereum chain, which is at the time of writing set at 30M Gas.

Next, in [Figure 5.2](#), we provide an overview of storage performance of our contract when storing *DCAT and on-chain provenance* w.r.t different granularities.

As we can see, storing data in high granularity results in a relatively high cost. Besides the increased volume of data, the smart contract implementation significantly contributes to this aspect. As covered previously in [Section 4.2](#), for storing provenance data in the contract, our solution utilizes the ERC-721 token standard. The smart contract uses state variables to keep track of the token ownership and other relevant data, which is stored directly on the blockchain. Despite the advantages provided by this token standard in addressing multiple issues, it is crucial to acknowledge the numerous added expenses that

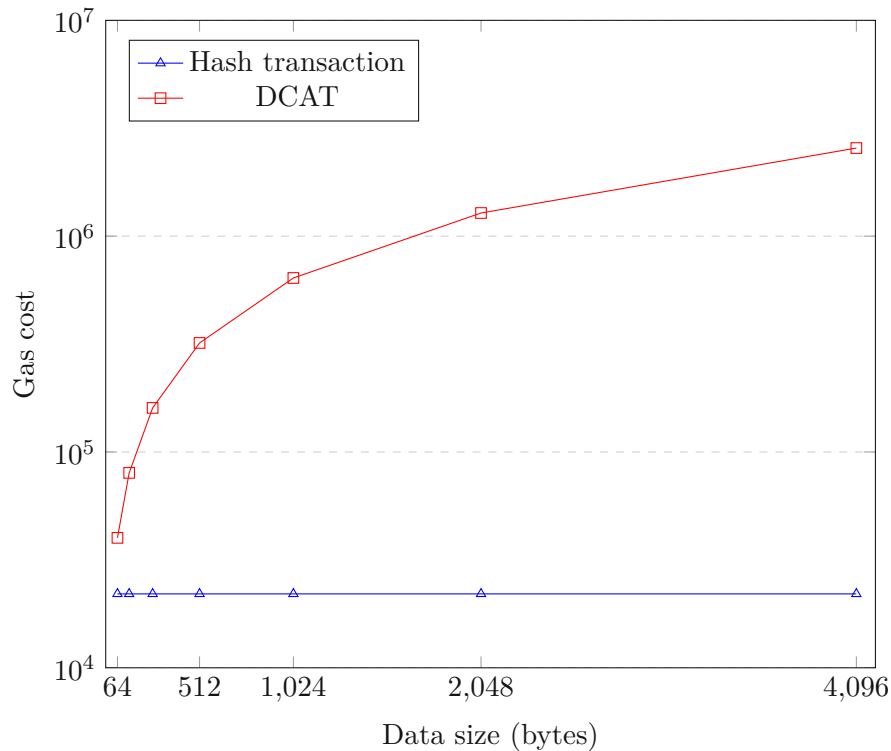


Figure 5.1: Estimated gas costs for storing DCAT and off-chain PROV data

occur by its utilization. The main one being that the ERC-721 standard includes several events, such as *Transfer* and *Approval*, that are emitted during token-related operations. Emitting these events results in additional gas. Further details on gas consumption in event logs are provided in the next paragraph.

Lastly, it is worth mentioning that at the time of writing, the block gas limit is around 30 million gas. By using our solution, this limit would be reached when storing approximately 23,418.75 bytes of data. We can make further use of this data to set an optimal cap for switching to off-chain data storage in our solution.

Event-logs. As referenced in [Subsection 2.3.2](#), when triggered, events add arguments for storage to a transaction's log – a unique data structure within the blockchain. These logs are connected to the smart contract that produced the events. Furthermore, three distinct types of logs are supported, allowing for different purposes: indexed, non-indexed, and anonymous indexed.

In contrast to non-indexed logs, indexed logs allow for efficient searching and filtering of log entries based on specific topics. On the other hand, anonymous indexed logs, provide a blend of the two, allowing for topic-based filtering without explicitly associating the logs with a particular event or smart contract. [Table 5.3](#) provides an overview of the events tested, together with the different topics and log data. The actual event declarations are

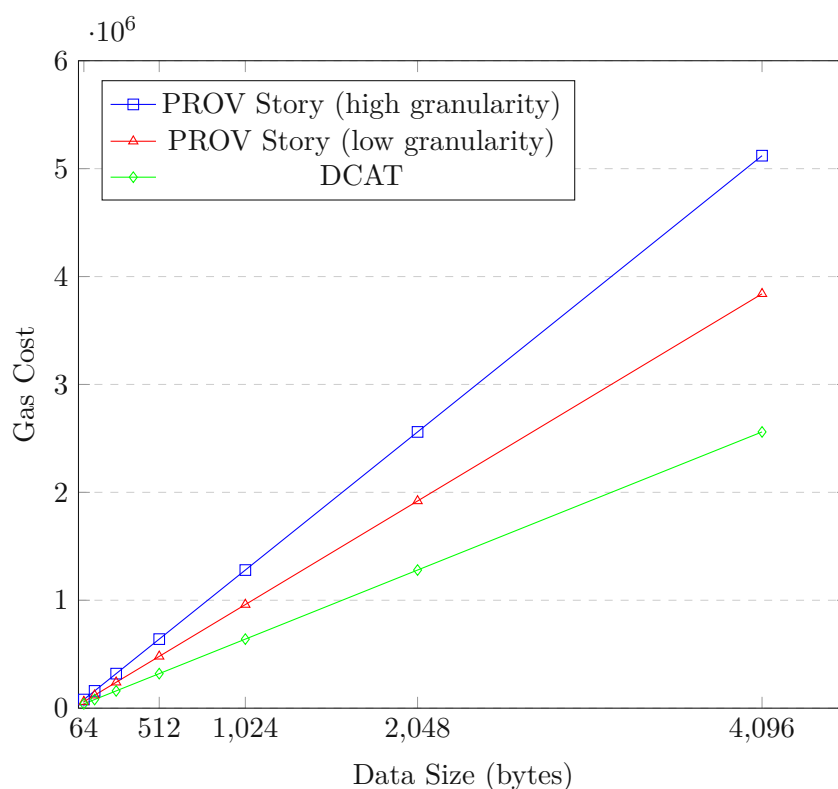


Figure 5.2: Estimated gas costs for storing DCAT and on-chain PROV data.

	Indexed	Non-indexed	Anonymous indexed
Declaration	(1)	(2)	(3)
Topic [0]	keccak(event signature)	keccak(event signature)	-
Topic [1]	abi_encode(provStoryId)	-	abi_encode(provStoryId)
Topic [2]	abi_encode(inputProvStoryIds)	-	abi_encode(inputProvStoryIds)
Topic [3]	abi_encode(index)	-	abi_encode(index)
Topic [4]	abi_encode(_datasetHashValue)	-	abi_encode(_datasetHashValue)
Topic [5]	abi_encode(tokenId)	-	abi_encode(tokenId)
Topic [6]	abi_encode(context)	-	abi_encode(context)
Log data	abi_encode(data)	abi_encode(data)	abi_encode(data)

Table 5.3: Structure of event-logs.

shown in [Listing 5.1](#).

All three types of logs have different cost factors involved, therefore we provide a comparison. The gas costs for indexed and non-indexed parameters is calculated as follows:

- Event signature: 375 gas (as it's a 32-byte topic)
- Data storage: 8 gas per byte

```

1   (1) event UpdateProvenanceEvent (
2       uint256 indexed provStoryId,
3       uint256[] inputProvStoryIds,
4       ...
5   )
6
7   (2) event UpdateProvenanceEvent (
8       uint256 provStoryId,
9       uint256[] inputProvStoryIds,
10      ...
11  )
12  (3) event UpdateProvenanceEvent (
13      uint256 indexed provStoryId,
14      uint256[] inputProvStoryIds,
15      ...
16  ) anonymous

```

Listing 5.1: Event declarations.

- Indexed topics: 375 gas per topic

The anonymous event does not store the event signature as a topic, allowing for a reduced gas cost. Based on the explored costs, the number of topics created by triggering the event in each scenario, along with the quantity of information stored in the log data field, are crucial elements contributing to the variation in cost between the three types of logs.

The costs resulting from triggering each event display minimal variation, as can be seen in [Figure 5.3](#). From the chart, it is apparent that the gas cost increases with the data size for all three types of event logs. Although anonymous events are the most cost-effective choice, they should be used sparingly. The reason for this is that they do not record their signature as a topic, thus making it difficult to uniquely identify them and retrieve them, which is often a main reason for implementing events.

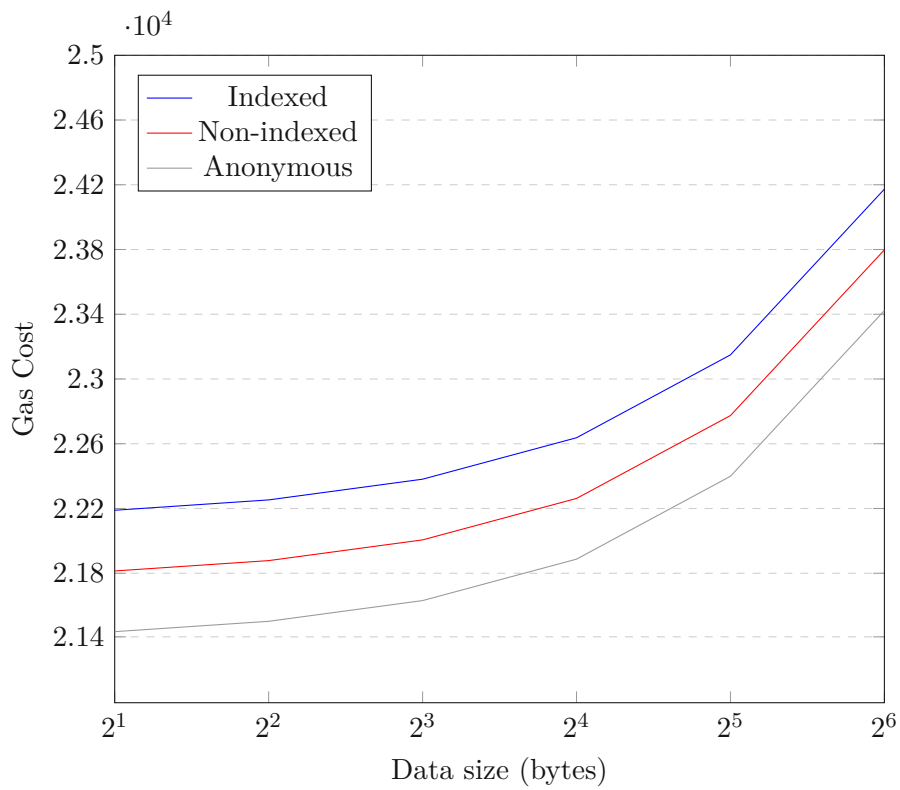


Figure 5.3: Storage cost of event-logs.



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.

Conclusion and Future Work

Combining provenance tracking and blockchain technologies has shown a significant potential for advancing various industries [LST⁺17]. It has been observed that the integration of these technologies can provide multiple benefits [RK18]. However, besides the opportunities that the combination these solutions provide, there are also multiple challenges that have to be addressed. A primary challenge is enhancing the efficiency of the blockchain in terms of computational and financial aspects, while ensuring that its advantages remain uncompromised.

In this thesis we propose comprehensive solution for managing provenance information across various domains and data catalogs. By utilizing the blockchain to store provenance in a decentralized manner, we successfully have increased transparency, security and trust in the data.

Through a scenario-based assessment, we demonstrated that our proposed solution is capable of addressing a wide range of use cases, some of which are not supported by similar modern solutions. A particular differentiation is the support of high-level data granularity. In addition, this direction highlighted the significance of maintaining an affordable system by utilizing off-chain data storage, an element commonly ignored by other solutions. By adopting a modular and stand-alone architecture, our solution encourages integration across various data catalogs. Domain independence is further supported by utilizing a widely adopted provenance data standard, not limited to any particular field. Lastly, we examined non-functional properties, and showed that our solution can be easily expanded to accommodate complex provenance scenarios, including those related to security properties.

6.1 Research Questions

The research questions of the thesis are discussed in the following, taking into account the insights and results obtained from the results and evaluation presented in this and previous chapters.

RQ 1. How can we extend existing solutions in collaborative Linked Open Data to support sufficient provenance information as means of preserving provenance information in a globally unique way?

To answer this question, we have developed a versatile solution capable of *multi-domain* support that leverages blockchain-based storage and tamper-resistant management of provenance information. In [Section 3.2](#) we describe in detail the architecture of our proposed solution.

Our approach emphasizes extensibility and adaptability, allowing for seamless integration with various data catalogs through an API-based architecture. During development, we were precautious not to rely on any specific data catalog functionality, ensuring that our solution can be readily employed across different systems.

Automation is another key aspect of our solution, which was addressed by system for the automatic scheduling and monitoring of workflows. By using this system we managed to minimize the administrative overhead associated with managing our solution. However, there is one area where automation remains a challenge: the deployment of smart contracts.

Smart contracts play a crucial role in our system, as they provide for a transparent and immutable on-chain storage solution. By using state-of-the-art standards, we were able to address the challenge of maintaining data consistency. Although the automatic generation of smart contracts proved more challenging than anticipated, and thus could not be addressed within the scope of this thesis, it remains a promising direction for future work.

Finally, we tackled the problem of contract extensibility and the integration of additional domain knowledge by designing our blockchain solution with migration functionality. This feature allowed us to achieve incorporation of new information and domain knowledge as needed, further demonstrating the adaptability of our approach.

RQ2. Which authority and consensus mechanisms are available to ensure the integrity of such a system? How can we preserve ethical considerations w.r.t energy consumption in such a solution?

In addressing the question of authority and consensus mechanisms for ensuring the integrity of our system, several options were considered. With the help of a taxonomy, details of which can be found in [Subsection 2.5.3](#), we were able to identify benefits and drawbacks of using different consensus protocols. Considering these factors together with the non-functional requirements, the best match for our solution implementation was a PoS mechanism.

To preserve ethical considerations regarding energy consumption, it is crucial to select a consensus mechanism that minimizes environmental impact without compromising the integrity and security of the system. By utilizing a PoS consensus mechanism we were able to preserve essential characteristics while also offering an excellent balance between cost-effectiveness, performance, and adaptability.

RQ3. What standards and tools are suitable to implement a universal solution, expandable to multiple domains with considerations of usability, scalability and flexibility?

In [Section 4.1](#) we have provided details based on our research, justifying the standards and tools employed in our solution.

Our approach combines on-chain and off-chain provenance storage to achieve a versatile and adaptable system. By utilizing a diverse technology stack, we ensure that the solution can cater to various domains and maintain its core properties of globality, scalability, and flexibility. These properties are further justified by the consensus mechanism utilized by our solution. In addition, we have performed a comprehensive analysis of the expenses associated with both on-chain and off-chain data storage, showcasing our solution's versatility in accommodating diverse cost constraints.

6.2 Future Work

The goal of the following section is to open a discussion for possible directions for future improvements and extensions of our system. Although the system we have is an early prototype we have been able to show that efficient storage and management of provenance information for cataloged open data in a tamper-resistant manner is feasible.

At present, open data is predominantly offered by public entities and seldom by private enterprises, but it has significant potential for economic and social benefits. Several open specifications and software tools are available to facilitate the publication of open data, yet there are numerous obstacles to overcome, particularly with regard to the quality and availability of data.

Data Verification. In our solution, we have implemented a verification task that involves confirming that the new entry extends the previous version of the same dataset. This allows us to maintain the integrity or accuracy of the data on a very basic level. Nonetheless, there exists potential for enhancement to attain a more sophisticated level of verification.

In [Subsection 2.5.2](#), we have seen a successful implementation of a verification process for cloud data provenance. The approach employs an auditor responsible for verifying the provenance data. This is achieved by extracting transaction details from the blockchain network through blockchain receipts, which encompass both block and transaction-related information.

Employing a similar direction in our solution can further increase the trust factor in the data. Furthermore, as we have seen our solution is quite expensive. Using a reward-based verification system can potentially provide for increased usability of our solution.

Automatic contract deployment. In our current solution, we employ a pre-configured smart contract. Nevertheless, a promising direction for future work could involve an automatic contract deployment on a per-dataset basis. Such an approach would improve the setup process, allowing for a seamless utilization of our solution. Furthermore, incorporating a custom ontology model for the smart contract could lead to a more comprehensive understanding of the provenance information. This enhancement would enable the system to draw inferences and derive more insightful conclusions, thus providing added value and expanding the potential use cases for our solution.

Scalability. Scalability remains a critical concern for the continued development and improvement of our solution. One approach under consideration for addressing scalability challenges is the implementation of sharding [NSNF17], which involves dividing the blockchain into separate chains, each responsible for managing contracts for a subset of controllers and processors. By distributing the workload across these distinct private chains, the system can achieve a more efficient performance. These individual chains would then synchronize with the public chain at regular intervals, such as every 1,000 blocks, to maintain public verifiability. This strategy has the potential to significantly enhance the system's scalability while still preserving the core principles of transparency and trustworthiness that are central to blockchain technology.

List of Figures

2.1 The four principles of Linked Data BHB09	10
2.2 An RDF triple statement	11
2.3 Abstract KG architecture	13
2.4 Informal graph of the sample triples	14
2.5 PROV Core Structures.	17
2.6 Provenance of a Dataset.	19
2.7 PROV Core Structures Expanded.	20
2.8 Blockchain protocol structure.	24
3.1 System Overview.	45
3.2 On-chain storage overview.	47
3.3 Structure of provenance as a story.	47
4.1 Contract Architecture.	55
4.2 Frontend display of Stories.	62
5.1 Estimated gas costs for storing DCAT and off-chain PROV data	76
5.2 Estimated gas costs for storing DCAT and on-chain PROV data.	77
5.3 Storage cost of event-logs.	79

List of Tables

2.1	Selected definitions of knowledge graph.	12
2.2	Comparison between public. consortium and private blockchains	28
2.3	Comparison between properties of modern data catalogs.	30
2.4	Comparison between properties of related work	34
2.5	Blockchain-related design decisions regarding (de)centralisation, with an indication of their relative impact on quality properties (✓: Least favourable, ✓✓: Neutral, ✓✓✓: More favourable).	35
2.6	Blockchain-related design decisions regarding storage and computation, with an indication of their relative impact on quality properties (✓: Least favourable, ✓✓: Less favourable, ✓✓✓: More favourable, ✓✓✓✓: Most favourable)	37
2.7	Blockchain-related design decisions regarding blockchain configuration (✓: Least favourable, ✓✓: Less favourable, ✓✓✓: More favourable, ✓✓✓✓: Most favourable)	38
5.1	Comparison between our solution and related work.	70
5.2	Impact of solution components on non-functional requirements. (✓: Minimal impact, ✓✓: Low impact, ✓✓✓: Moderate impact, ✓✓✓✓: High impact)	72
5.3	Structure of event-logs.	77



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.

Listings

4.1 ProvenanceStory contract gist.	58
4.2 ProvenanceStory: Update function gist.	59
4.3 DCAT contract gist.	60
4.4 Data extraction agent middleware.	64
4.5 Provenance handling agent middleware.	66
5.1 Event declarations.	78



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

- [AN19] Ifeyinwa Angela Ajah and Henry Friday Nweke. Big data and business analytics: Trends, platforms, success factors and applications. *Big Data Cogn. Comput.*, 3(2):32, 2019.
- [Ant14] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, Inc., 2014.
- [AS09] Rajendra Akerkar and Priti Sajja. *Knowledge-Based Systems*. Jones and Bartlett Publishers, Inc., USA, 1st edition, 2009.
- [BCL⁺94] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The world-wide web. *Commun. ACM*, 37(8):76–82, 1994.
- [BHB09] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.
- [BKT01] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*, volume 1973 of *Lecture Notes in Computer Science*, pages 316–330. Springer, 2001.
- [BL] Tim Berners-Lee. Linked data. <https://www.w3.org/DesignIssues/LinkedData.html>. Last Accessed: 2023-04-30.
- [BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *ScientificAmerican.com*, 05 2001.
- [Bos02] Rajendra Bose. A conceptual framework for composing and managing scientific data lineage. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management, July 24-26, 2002, Edinburgh, Scotland, UK*, pages 15–19. IEEE Computer Society, 2002.

- [Cag19] Kurt Cagle. The coming merger of blockchain and knowledge graphs. <https://medium.com/@kurtcagle/685e052c614c>, Nov 2019. Last Accessed: 2023-04-30.
- [CCT09] James Cheney, Laura Chiticariu, and Wang Chiew Tan. Provenance in databases: Why, how, and where. *Found. Trends Databases*, 1(4):379–474, 2009.
- [Che11] James Cheney. A formal framework for provenance security. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*, pages 281–293. IEEE Computer Society, 2011.
- [Cho09] Stephen Chong. Towards semantics for provenance security. In *First Workshop on on Theory and Practice of Provenance, TAPP'09, USA, 2009*. USENIX Association.
- [CMD⁺11] Sam Coppens, Erik Mannens, Davy Van Deursen, Patrick Hochstenbach, Bart Janssens, and Rik Van de Walle. Publishing provenance information on the web using the memento datetime content negotiation. In Christian Bizer, Tom Heath, Tim Berners-Lee, and Michael Hausenblas, editors, *WWW2011 Workshop on Linked Data on the Web, Hyderabad, India, March 29, 2011*, volume 813 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.
- [CW00] Yingwei Cui and Jennifer Widom. Practical lineage tracing in data warehouses. In David B. Lomet and Gerhard Weikum, editors, *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA, February 28 - March 3, 2000*, pages 367–378. IEEE Computer Society, 2000.
- [CZA⁺18] Yannis Charalabidis, Anneke Zuiderwijk, Charalampos Alexopoulos, Marijn Janssen, Thomas Lampoltshammer, and Enrico Ferro. *The Open Data Landscape: Concepts, Methods, Tools and Experiences*, pages 1–9. 09 2018.
- [DA20] Tran Khanh Dang and Thu Duong Anh. A pragmatic blockchain based solution for managing provenance and characteristics in the open data context. In Tran Khanh Dang, Josef Küng, Makoto Takizawa, and Tai M. Chung, editors, *Future Data and Security Engineering*, pages 221–242, Cham, 2020. Springer International Publishing.
- [Dav93] Alan M. Davis. *Software requirements - objects, functions, and states*. Prentice Hall international editions. Prentice Hall, 1993.
- [dCCM09] Sérgio Manuel Serra da Cruz, Maria Luiza Machado Campos, and Marta Mattoso. Towards a taxonomy of provenance in scientific workflow management systems. In *2009 IEEE Congress on Services, Part I, SERVICES*

I 2009, Los Angeles, CA, USA, July 6-10, 2009, pages 259–266. IEEE Computer Society, 2009.

- [DKP18] Andrey Demichev, Alexander Kryukov, and Nikolai Prikhodko. The approach to managing provenance metadata and data access rights in distributed storage using the hyperledger blockchain platform. In *2018 Ivanikov Ispras Open Conference (ISPRAS)*, pages 131–136, 2018.
- [DKR⁺11] Susan B. Davidson, Sanjeev Khanna, Sudeepa Roy, Julia Stoyanovich, Val Tannen, and Yi Chen. On provenance and privacy. In *Proceedings of the 14th International Conference on Database Theory, ICDT '11*, page 3–10, New York, NY, USA, 2011. Association for Computing Machinery.
- [DLZ⁺18] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Ji Wang. Untangling blockchain: A data processing view of blockchain systems. *IEEE Transactions on Knowledge and Data Engineering*, 30(7):1366–1385, 2018.
- [DSW06] John Davies, Rudi Studer, and Paul Warren. *Semantic Web Technologies: Trends and Research in Ontology-Based Systems*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2006.
- [DWPJV02] Bart De Win, Frank Piessens, Wouter Joosen, and Tine Verhanneman. On the importance of the separation-of-concerns principle in secure software engineering. In *Workshop on the Application of Engineering Principles to System Security Design*, pages 1–10. Citeseer, 2002.
- [ESM⁺21] Lisa Ehrlinger, Johannes Schrott, Martin Melichar, Nicolas Kirchmayr, and Wolfram Wöß. Data catalogs: A systematic literature review and guidelines to implementation. In Gabriele Kotsis, A. Min Tjoa, Ismail Khalil, Bernhard Moser, Atif Mashkoor, Johannes Sametinger, Anna Fensel, Jorge Martinez-Gil, Lukas Fischer, Gerald Czech, Florian Sobieczky, and Sohail Khan, editors, *Database and Expert Systems Applications - DEXA 2021 Workshops*, pages 148–158, Cham, 2021. Springer International Publishing.
- [EW16] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. In Michael Martin, Martí Cuquet, and Erwin Folmer, editors, *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16) co-located with the 12th International Conference on Semantic Systems (SEMANTiCS 2016), Leipzig, Germany, September 12-15, 2016*, volume 1695 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.
- [FBMR18] Michael Färber, Frederic Bartscherer, Carsten Menne, and Achim Rettinger. Linked data quality of dbpedia, freebase, opencyc, wikidata, and YAGO. *Semantic Web*, 9(1):77–129, 2018.

- [FW16] Christina Feilmayr and Wolfram Wöß. An analysis of ontologies and their success factors for application to business. *Data Knowl. Eng.*, 101:1–23, 2016.
- [GKT07] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In Leonid Libkin, editor, *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 31–40. ACM, 2007.
- [Gom11] Hassan Gomaa. *Software modeling and design: UML, use cases, patterns, and software architectures*. Cambridge University Press, 2011.
- [GPVW17] José Manuel Gómez-Pérez, Jeff Z. Pan, Guido Vetere, and Honghan Wu. Enterprise knowledge graph: An introduction. In Jeff Z. Pan, Guido Vetere, José Manuel Gómez-Pérez, and Honghan Wu, editors, *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, pages 1–14. Springer, 2017.
- [GSM⁺06] Paul Groth, Jian Sheng, Simon Miles, Steve Munroe, Victor Tan, Sofia Meacham, and Luc Moreau. An architecture for provenance systems. 02 2006.
- [Gup09] Amarnath Gupta. *Data Provenance*, pages 608–608. Springer US, Boston, MA, 2009.
- [HBC⁺21] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D’amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4), jul 2021.
- [HC20] Soumaya Ben Hassine and Delphine Clément. Open data quality dimensions and metrics: State of the art and applied use cases. In Witold Abramowicz and Gary Klein, editors, *Business Information Systems Workshops - BIS 2020 International Workshops, Colorado Springs, CO, USA, June 8-10, 2020, Revised Selected Papers*, volume 394 of *Lecture Notes in Business Information Processing*, pages 311–323. Springer, 2020.
- [HGS⁺06] Christian Halaschek-Wiener, Jennifer Golbeck, Andrew Schain, Michael Grove, Bijan Parsia, and James A. Hendler. Annotation and provenance tracking in semantic web photo libraries. In Luc Moreau and Ian T. Foster, editors, *Provenance and Annotation of Data, International Provenance and Annotation Workshop, IPAW 2006, Chicago, IL, USA, May 3-5, 2006, Revised Selected Papers*, volume 4145 of *Lecture Notes in Computer Science*, pages 82–89. Springer, 2006.

- [HHU⁺11] Aidan Hogan, Andreas Harth, Jürgen Umbrich, Sheila Kinsella, Axel Polleres, and Stefan Decker. Searching and browsing linked data with SWSE: the semantic web search engine. *J. Web Semant.*, 9(4):365–401, 2011.
- [HZ10] Olaf Hartig and Jun Zhao. Publishing and consuming provenance metadata on the web of linked data. In Deborah L. McGuinness, James Michaelis, and Luc Moreau, editors, *Provenance and Annotation of Data and Processes - Third International Provenance and Annotation Workshop, IPAW 2010, Troy, NY, USA, June 15-16, 2010. Revised Selected Papers*, volume 6378 of *Lecture Notes in Computer Science*, pages 78–90. Springer, 2010.
- [KH21] Fabian Kirstein and Manfred Hauswirth. Blockchain for trustworthy publication and integration of linked open data. In *Proceedings of the 11th on Knowledge Capture Conference, K-CAP '21*, page 269–272, New York, NY, USA, 2021. Association for Computing Machinery.
- [Lan90] David P. Lanter. Lineage in gis: The problem and a solution. 1990.
- [LKLG17] Seokki Lee, Sven Köhler, Bertram Ludäscher, and Boris Glavic. A sql-middleware unifying why and why-not provenance for first-order queries. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*, pages 485–496. IEEE Computer Society, 2017.
- [LST⁺17] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid '17*, page 468–477. IEEE Press, 2017.
- [MBC13] Paolo Missier, Khalid Belhajjame, and James Cheney. The w3c prov family of specifications for modelling provenance metadata. In *Proceedings of the 16th International Conference on Extending Database Technology, EDBT '13*, page 773–776, New York, NY, USA, 2013. Association for Computing Machinery.
- [MLA⁺08] Luc Moreau, Bertram Ludascher, Ilkay Altintas, Roger S. Barga, Shawn Bowers, Steven Callahan, George Chin, Ben Clifford, Shirley Cohen, Sarah Cohen-Boulakia, Susan Davidson, Ewa Deelman, Luciano Digiampietri, Ian Foster, Juliana Freire, James Frew, Joe Futrelle, Tara Gibson, Yolanda Gil, Carole Goble, Jennifer Golbeck, Paul Groth, David A. Holland, Sheng Jiang, Jihie Kim, David Koop, Ales Krenek, Timothy McPhillips, Gaurang Mehta, Simon Miles, Dominic Metzger, Steve Munroe, Jim Myers, Beth Plale, Norbert Podhorszki, Varun Ratnakar, Emanuele Santos, Carlos Scheidegger, Karen Schuchardt, Margo Seltzer, Yogesh L. Simmhan, Claudio Silva, Peter

Slaughter, Eric Stephan, Robert Stevens, Daniele Turi, Huy Vo, Mike Wilde, Jun Zhao, and Yong Zhao. Special issue: The first provenance challenge. 20(5):409–418, apr 2008.

- [MM] Luc Moreau and Paolo Missier. Prov-n: The provenance notation. <https://www.w3.org/TR/2013/REC-prov-n-20130430/>. Last Accessed: 2023-04-30.
- [MM74] E. Marchi and O. Miguel. On the structure of the teaching-learning interactive process. *International Journal of Game Theory*, 3(2):83–99, Jun 1974.
- [Nak] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>. Last Accessed: 2023-04-30.
- [net21] Network graph. <https://datawalk.com/glossary/network-graph>, Dec 2021. Last Accessed: 2023-04-30.
- [NFSC19] Robert Norvill, Beltran Fiz, Radu State, and Andrea Cullen. Standardising smart contracts: Automatically inferring erc standards. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 192–195, 2019.
- [NKS⁺18] Ivica Nikolić, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. Finding the greedy, prodigal, and suicidal contracts at scale. In *Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC '18*, page 653–663, New York, NY, USA, 2018. Association for Computing Machinery.
- [NSNF17] Ricardo Neisse, Gary Steri, and Igor Nai-Fovino. A blockchain-based approach for data accountability and provenance tracking. In *Proceedings of the 12th International Conference on Availability, Reliability and Security, ARES '17*, New York, NY, USA, 2017. Association for Computing Machinery.
- [NSNFB15] Ricardo Neisse, Gary Steri, Igor Nai Fovino, and Gianmarco Baldini. Seckit: A model-based security toolkit for the internet of things. *Computers & Security*, 58, 10 2015.
- [NTLK18] Sebastian Neumaier, Lörinc Thurnay, Thomas J. Lampoltshammer, and Tomá Knap. Search, filter, fork, and link open data: The adequate platform: Data- and community-driven quality improvements. In *Companion Proceedings of the The Web Conference 2018, WWW '18*, page 1523–1526, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

- [ora] What is a data catalog? <https://www.oracle.com/big-data/data-catalog/what-is-a-data-catalog/>. Last Accessed: 2023-04-30.
- [Pau17] Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, 8(3):489–508, 2017.
- [PMGC13] Jay Pujara, Hui Miao, Lise Getoor, and William Cohen. Knowledge graph identification. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web – ISWC 2013*, pages 542–557, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [proa] Provenance incubator group charter. <https://www.w3.org/2005/Incubator/prov/charter>. Last Accessed: 2023-04-30.
- [prob] Provenance xg final report. <https://www.w3.org/2005/Incubator/prov/XGR-prov-20101214/>. Last Accessed: 2023-04-30.
- [PRS18] Beatriz Pérez, Julio Rubio, and Carlos Sáenz-Adán. A systematic review of provenance systems. *Knowl. Inf. Syst.*, 57(3):495–543, 2018.
- [PS19] Alessandro Piscopo and Elena Simperl. What we talk about when we talk about wikidata quality: A literature survey. In *Proceedings of the 15th International Symposium on Open Collaboration, OpenSym '19*, New York, NY, USA, 2019. Association for Computing Machinery.
- [PT21] Giuseppe Antonio Pierro and Roberto Tonelli. Analysis of source code duplication in ethereum smart contracts. In *28th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2021, Honolulu, HI, USA, March 9-12, 2021*, pages 701–707. IEEE, 2021.
- [QK04] D. A. Quan and R. Karger. How to make a semantic web browser. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, pages 255–265. ACM, 2004.
- [Ran18] Suhas Ranganath. Leveraging catalog knowledge graphs for query attribute identification in e-commerce sites. *CoRR*, abs/1807.04923, 2018.
- [rdf] Rdf 1.1 concepts and abstract syntax. <https://www.w3.org/TR/rdf11-concepts/>. Last Accessed: 2023-04-30.
- [RK18] Aravind Ramachandran and Murat Kantarcioglu. Smartprovenance: A distributed, blockchain based dataprovenance system. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY '18*, page 35–42, New York, NY, USA, 2018. Association for Computing Machinery.

- [SBP14] Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. Adoption of the linked data best practices in different topical domains. In *International semantic web conference*, pages 245–260. Springer, 2014.
- [Sch] Bess Schrader. Rdf*: What is it and why do i need it? <https://www.w3.org/TR/prov-overview/>. Last Accessed: 2023-04-30.
- [Sch21] Fabian Schär. Decentralized finance: On blockchain-and smart contract-based financial markets. *FRB of St. Louis Review*, 2021.
- [sem14] From taxonomies over ontologies to knowledge graphs. <https://semantic-web.com/from-taxonomies-over-ontologies-to-knowledge-graphs/>, Mar 2014. Last Accessed: 2023-04-30.
- [str] Structure of a contract - events. <https://docs.soliditylang.org/en/v0.8.19/structure-of-a-contract.html#events>. Last Accessed: 2023-04-30.
- [SW01] Geri Schneider and Jason P Winters. *Applying use cases: a practical guide*. Pearson Education, 2001.
- [Sza97] Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997.
- [TS16] Florian Tschorsch and Björn Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Commun. Surv. Tutorials*, 18(3):2084–2123, 2016.
- [Vra12] Denny Vrandečić. Wikidata: A new platform for collaborative data collection. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, page 1063–1064, New York, NY, USA, 2012. Association for Computing Machinery.
- [vue] Vue spa. <https://vuejs.org/guide/extras/ways-of-using-vue.html#single-page-application-spa>. Last Accessed: 2023-04-30.
- [W⁺14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.
- [w3c] Data catalog vocabulary (dcat) - version 2. <https://www.w3.org/TR/vocab-dcat/>. Last Accessed: 2023-04-30.
- [WCHG17] Marcin Wylot, Philippe Cudré-Mauroux, Manfred Hauswirth, and Paul Groth. Storing, tracking, and querying provenance in linked data. *IEEE Trans. Knowl. Data Eng.*, 29(8):1751–1764, 2017.

- [WPM⁺08] Shaowen Wang, Anand Padmanabhan, James D. Myers, Wenwu Tang, and Yong Liu. Towards provenance-aware geographic information systems. In Walid G. Aref, Mohamed F. Mokbel, and Markus Schneider, editors, *16th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2008, November 5-7, 2008, Irvine, California, USA, Proceedings*, page 70. ACM, 2008.
- [WS97] Allison Woodruff and Michael Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In W. A. Gray and Per-Åke Larson, editors, *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997, Birmingham, UK*, pages 91–102. IEEE Computer Society, 1997.
- [XWS⁺17] Xiwei Xu, Ingo Weber, Mark Staples, Liming Zhu, Jan Bosch, Len Bass, Cesare Pautasso, and Paul Rimba. A taxonomy of blockchain-based systems for architecture design. In *2017 IEEE International Conference on Software Architecture, ICSA 2017, Gothenburg, Sweden, April 3-7, 2017*, pages 243–252. IEEE Computer Society, 2017.
- [Yu14] Liyang Yu. *A Developer’s Guide to the Semantic Web, Second Edition*. Springer, 2014.
- [ZGG⁺03] Jun Zhao, Carole Goble, Mark Greenwood, Chris Wroe, and Robert Stevens. Annotating, linking and browsing provenance logs for e-science. 11 2003.
- [ZKS⁺17] Faheem Zafar, Abid Khan, Sabah Suhail, Idrees Ahmed, Khizar Hameed, Hayat Mohammad Khan, Farhana Jabeen, and Adeel Anjum. Trustworthy data: A survey, taxonomy and future trends of secure provenance schemes. *J. Netw. Comput. Appl.*, 94:50–68, 2017.
- [ZXD⁺17] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In George Karypis and Jia Zhang, editors, *2017 IEEE International Congress on Big Data, BigData Congress 2017, Honolulu, HI, USA, June 25-30, 2017*, pages 557–564. IEEE Computer Society, 2017.