UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Informáticos

Convergence and representation of blockchain and smart contracts using the semantic web

DOCTORAL THESIS

Submitted for the degree of Doctor by:

Juan Cano de Benito
MsC in Artificial Intelligence

Madrid, 2024

UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingenieros Informáticos

Doctoral Degree in Artificial Intelligence

Convergence and representation of blockchain and smart contracts using the semantic web

DOCTORAL THESIS

Submitted for the degree of Doctor by:

Juan Cano de Benito
MsC in Artificial Intelligence

Under the supervision of:
Dr. Javier Bajo Perez
Dr. Raúl García Castro

Madrid, 2024

Title: Convergence and representation of blockchain and smart contracts using the semantic web

Author: Juan Cano de Benito

Doctoral Programme: Doctoral Degree in Artificial Intelligence

Thesis Supervision:

Dr. Javier Bajo Perez, Full Professor (Universidad Politécnica de Madrid)

Dr. Raúl García Castro, Associate Professor (Universidad Politécnica de Madrid)

External Reviewers:

Thesis Defense Committee:

Thesis Defense Date:

"Es preciso soñar, pero con la condición de creer en nuestros sueños. De examinar con atención la vida real, de confrontar nuestra observación con nuestros sueños, y de realizar escrupulosamente nuestra fantasía" Vladimir Ilyich Ulyanov

Acknowledgements

Una tesis doctoral no es trabajo baladí. Los trabajos de investigación requieren de gran trabajo que no solo es realizado por una persona, si no que es un trabajo en equipo. Por ello en estos párrafos intentaré resumir el gran trabajo que han realizado aquellos que me han ayudado en mi camino, sin hacer el honor que merecen, ya que no encuentro las palabras para expresar este agradecimiento.

En primer lugar, esta tesis no habría sido posible sin Javier Bajo Pérez ni sin Raúl García Castro, por enseñarme el maravilloso mundo de la investigación y por ofrecerme la maravillosa oportunidad de estar en un gran laboratorio que me ha ayudado a crecer profesional y académicamente. En este primer lugar debe ser mencionado mi compañero Andrea Cimmino. Andrea, gracias por tu paciencia, apoyo, optimismo y consejos a lo largo de este camino que han ayudado a allanar el camino de rocas. Nunca sabré como agradecer todo lo que habéis hecho por mi.

En segundo lugar, al Departamento de Inteligencia Artificial, por acogerme hace años como becario, ya que este camino hubiera sido radicalmente distinto sin esa oportunidad, por lo que gracias a todo el profesorado del Departamento de Inteligencia Artificial. Me gustaría poner aquí todos y cada uno de vuestros nombres y explicar lo mucho que significó ese periodo para mi, pero entonces solamente estos agradecimientos serían ya en sí mismos una tesis. También al Grupo de Ingeniería Ontológica (OEG), un gran grupo donde hay una gran cantidad de buenas personas y donde el buen ambiente impera en cualquier momento.

Third, thanks to John Domingue. Thanks for giving me the opportunity to work and collaborate with you and your team, especially with Aisling Third and Michelle Bachler. Thank you to Andrea, Susan, and Tony for opening the doors of your home to me and facilitating my research stay in Milton Keynes. And thanks to my Spanish colleagues working at the OU, Cecilia, Paula, and Ángel.

En el campo personal, gracias a toda mi familia, en especial a mi padre, Santiago, a mi madre, Almudena, a mi hermana, Rocío y a mi abuela, Amada. Y gracias a mi otra familia. A Rocío, por aguantarme todos estos años y los que te quedan. A Mar y Chema, por recorrer juntos el camino de la tesis e investigación. A Cristina por su optimismo pero sin dejar de lado el realismo, a Adrián por su compañerismo y espíritu vikingo y a Alex, cuya amistad se forjó durante la infancia y sigue perdurando. A Daniel y a Fer, por soportarme estos años.

I als meus amics de València: Adrián, Juliol, Fran, Vicente, Ana, Imanol, Rosa, Alvaro i la resta, que demostren que de les xicotetes casualitats de la vida poden sorgir grans moments.

Gracias a toda la gente mencionada, por la cual cada minuto es un regalo.

Abstract

The Internet and the World Wide Web were designed with the intention of being decentralised and interoperable, and therefore more democratic, where users have power over their data. Despite efforts to maintain this decentralisation, in recent years, the Internet has become a highly centralised network, thus ignoring the principles of decentralisation and democratisation on which it was designed. In response to this increasing centralisation of the Internet, different technologies have emerged that advocate the principles on which the Internet was built, such as the semantic web and blockchain.

This thesis addresses the synergy between blockchain technology and the semantic web, two technologies that advocate the decentralisation of the Internet and give control of data back to users, allowing them to manage their own information in a secure way while both technologies have unique characteristics and can feed back on each other, either by offering the interoperability characteristic of the semantic web or the security of data provided by blockchain technologies. Therefore, this thesis explores and lays the groundwork for how the combination of these technologies can address the problem of decentralisation while offering solutions for transparent and interoperable data management.

First, a state-of-the-art analysis of both technologies is performed, defining the main characteristics of the blockchain and the semantic web. The potential of blockchain to provide the decentralisation of the web and the immutability of records is analysed, as well as the different existing blockchain technologies, while the main standards and characteristics of the semantic web, such as interoperability and data understanding through ontologies and knowledge graphs, are analysed. Second, an analysis of the benefits that arise from the synergy between both technologies is made, and different prototypes of scenarios that integrate blockchain with the semantic web are proposed. Third, an analysis of the most promising scenarios is performed, and experiments are designed to evaluate the effectiveness and performance of the proposed scenarios. Fourth, ontology models are developed to cover Ethereum blockchain technology and smart contracts. Finally, a knowledge graph with these ontologies is built to demonstrate how the integration of these technologies can facilitate data management and improve blockchain analytics.

In conclusion, this study lays the foundation for achieving a decentralised, standards-based, user-driven, data-driven Internet. Despite the benefits reflected in this thesis in achieving decentralisation of the Internet, the results obtained indicate that the integration of blockchain technology with the semantic web is feasible, but has certain limitations that need to be addressed, such as the cost of storing semantic data on the blockchain.

Resumen

El internet y la World Wide Web fueron diseñados con la intención de ser descentralizados e interoperables y, por lo tanto, más democráticos, donde los usuarios tuvieran el poder sobre sus datos. A pesar de los esfuerzos por mantener esta descentralización, en los últimos años Internet se ha vuelto una red altamente centralizada, ignorando así los principios de descentralización y democratización sobre los que fue diseñado. En respuesta a esta creciente centralización de Internet, han surgido diferentes tecnologías que abogan por los principios sobre los que se construyó Internet, como la web semántica y la blockchain.

La presente tesis aborda la sinergia entre la tecnología blockchain y la web semántica, dos tecnologías que abogan por la descentralización de Internet y por devolver el control de los datos a los usuarios, permitiendo gestionar su propia información de manera segura, al mismo tiempo que ambas tecnologías tienen características únicas y pueden retroalimentarse entre sí, ya sea por ejemplo ofreciendo la interoperabilidad característica de la web semántica o la seguridad de los datos que proporcionan las tecnologías blockchain. Por lo tanto, esta tesis explora y sienta las bases de cómo la combinación de estas tecnologías pueden abordar el problema de la descentralización, ofreciendo además soluciones para la gestión transparente e interoperable de los datos.

En primer lugar, se realiza un análisis del estado del arte de ambas tecnologías, definiendo las principales características de blockchain y la web semántica. Se analiza el potencial del blockchain para proporcionar la descentralización de la web y la inmutabilidad de los registros, así como se estudian las distintas tecnologías blockchain existentes, mientras que se analizan los principales estándares y características de la web semántica, como la interoperabilidad y la comprensión de los datos a través de ontologías y grafos de conocimiento.

En segundo lugar, se realiza un análisis de los beneficios que surgen producto de la sinergia al combinar ambas tecnologías y se proponen diferentes prototipos de escenarios que integran blockchain con la web semántica. En tercer lugar, se realiza un análisis de los escenarios más prometedores y se realizan experimentos para evaluar la eficacia y rendimiento de los escenarios propuestos. En cuarto lugar, se desarrollan modelos ontológicos para cubrir la tecnología blockchain Ethereum y los contratos inteligentes. Por último, con estas ontologías se construyó un grafo de conocimiento para demostrar cómo la integración de estas tecnologías puede facilitar la gestión de datos y mejorar el análisis de la blockchain.

En conclusión, este estudio sienta una base para alcanzar un Internet descentralizado, cuyos datos sean gestionados por los propios usuarios y esté basado en estándares. A pesar de los beneficios que se reflejan en esta tesis para alcanzar la descentralización de Internet, los resultados obtenidos indican que la integración de la tecnología blockchain con la web semántica es viable, pero tiene ciertas limitaciones que deben ser abordadas, como el coste de almacenar datos semánticos en blockchain.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> Science knows no country, because knowledge belongs to humanity, and is the torch that illuminates the world.
>
> *Louis Pasteur*

Since the early days of the Internet, with ARPANET [128], it was intended to promote decentralised and, thus, more democratic social and economic structures [109]. In 1968, Stewart Brand, the creator of the Whole Earth Catalogue (WEC) [21], held the view that technological progress, social harmony, and the preservation of nature were not incompatible. Therefore, the main objective of the WEC was to provide access to technical knowledge to the greatest number of individuals to enable them to achieve decentralised production of material goods and to transcend capitalist structures, leading to a decentralised technological world in which the Internet would be fully decentralised [145]. Since these early days of the Internet, decentralised databases have been proposed, but it is worth noting the work of David Chaum [29], whose decentralised database supports digital currency. This approach can be considered the precursor of blockchain.

Tim Berners-Lee invented the World Wide Web (WWW), making the first proposal in March 1989 [15], with the main philosophy of maintaining it open, non-proprietary and free [79]. With the emergence of the World Wide Web, authors like McGeady [98] refer to the World Wide Web as a free and open medium, which eliminates the separation of roles between communicator and receiver, leading to a shift towards decentralised management models and decentralised working models. However, authors like Postman [130] emphasised that the challenge since the invention of the WWW is to harness information, rather than simply disseminate it, to generate knowledge and insight, and therefore the information should be protected.

In 1991, the first distributed ledger technology (DLT) was proposed [60]. DLT can be described as a platform that uses interconnected devices to store ledgers, thus guaranteeing the accuracy and security of the data. This was a first step towards allowing users to choose what and how to exchange their data without the need for an intermediary in a decentralised way. In

1994, Tim Berners-Lee founded the World Wide Web Consortium (W3C)[1], and in 2001, Tim Berners-Lee [14] described an expected evolution of the existing Web to a semantic Web. According to the W3C, the semantic web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. The semantic web philosophy emphasises decentralisation, eliminates data silos, empowers data ownership, and is open and secure. In this semantic web proposal, RDF (Resource Description Framework) is the standard model for data interchange on the Web for describing structured information [158].

With the growing power of corporations in the context of the Internet and the World Wide Web, characterised by a historically unprecedented bundling of private sector power over infrastructures [44], Tim O'Reilly [118] questioned who owns the user data across the Internet. In this context, the term prosumer arises as a consumer who also produces and would imply the decentralisation not only of the processes of media production and dissemination, but also of the processes of sociopolitical organisation [136].

After the 2008 financial crisis, the trust inherent in money and companies also began to erode. This led to the emergence of blockchain technology in 2008 by Santoshi Nakamoto with Bitcoin [108]. Although blockchain is a type of DLT, in the past decade, it has become a disruptive technology used in industry, government, and research sectors, becoming a technology that has the potential to revolutionise multiple industries, from finance and healthcare to supply chain management, and more [7]. The reason is due to some key characteristics of this technology, such as decentralisation, persistence, anonymity, and auditability [175]. Another reason is that, unlike traditional DLTs, it has different consensus algorithms that address the Byzantine Generals Problem [101]. According to some authors, blockchain technology could be the catalyst for a new post-capitalist period, as intermediary institutions and companies are perceived as increasingly obsolete [37, 38]. Figure 1.1 depicts the main efforts made for the decentralisation of the Internet throughout history in the context of this thesis.

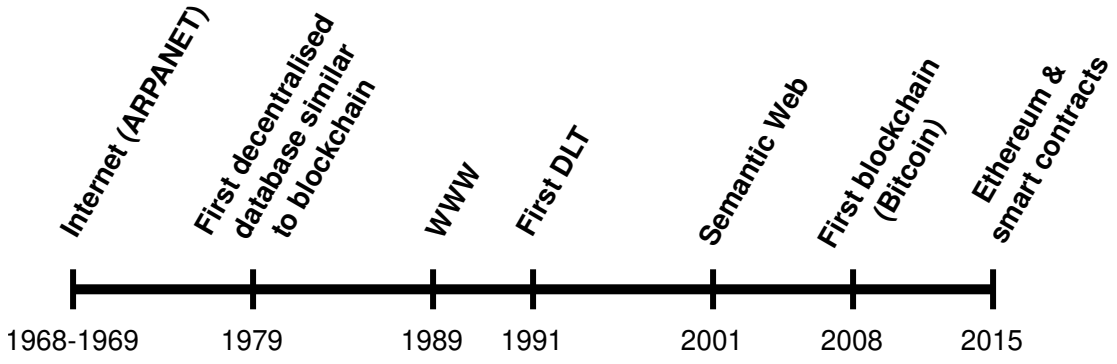

Figure 1.1: History of decentralisation of the Internet

Despite the efforts for decentralisation, the Internet has become highly centralised due to its role as a site of capital accumulation. In contrast to the early days of the Internet, when vital Internet infrastructure embraced decentralisation as a technical and organisational feature,

[1] https://www.w3.org/about/history/

Big tech companies have embraced a more centralised structure that is friendlier to corporate forms and profits. Most regions are served by only a few or even one broadband Internet Service Provider (ISP)[2], and in the years 2021 and 2022, approximately half of the Internet data traffic was generated by the six largest big techs[3] as depicted by table 1.1. In itself, decentralisation has no value. Rather, it is valuable because of how it affects the landscape of power. By embedding decision-making power in the leaves of the network, in the citizenry that constitutes the Internet, decentralisation provides the basis (at least in theory) for greater democratic control.

Brand	2021	2022
Google	21.0%	13.9%
Netflix	9.4%	13.7%
Meta	15.1%	6.5%
Microsoft	3.3%	5.1%
Apple	4.2%	4.6%
Amazon	3.4%	4.2%
Total	56.4%	48.0%

Table 1.1: Internet traffic generation in 2021 and 2022

To summarise, the Internet was designed using the most advanced decentralisation technologies available at the time. Its design was never robust enough to permanently resist the forces of capital accumulation, and it is clear that these forces have now completely demolished the bulwarks of decentralisation built at the inception of the Internet. New technological innovations will be needed to build a decentralised Internet, and a new organisational scheme will have to be developed to redistribute power on the Internet. The main motivation of this thesis is to provide the basis for a possible new Internet scheme, using technologies for decentralisation such as blockchain and the semantic web.

The key characteristics of blockchain are significant. First, the decentralised nature of blockchain means that it is not controlled by any single entity or organisation, making it highly resistant to hacking and other forms of tampering. Additionally, the use of cryptographic hashes ensures that the data on the blockchain is immutable and transparent, providing a high level of security and transparency. Moreover, blockchain technology can reduce costs and increase efficiency by eliminating the need for intermediaries and reducing transaction times. With the ever-growing adoption of this technology, new proposals and technologies have been devised, such as Ethereum, which introduced the concept of smart contracts [170] (computer programs that automate the actions required in an agreement or contract) and has become a relevant characteristic of blockchain; Peercoin, which introduced a new consensus mechanism [89] (the entire stack of protocols, incentives and ideas that allow a network of nodes to agree on the state of a blockchain), or IOTA, which is focused on IoT devices [129]. The common method for storing data in a blockchain is through the use of transactions; these transactions can store data by using smart contracts or directly in the transaction. For example, in Ethereum, a transaction has a field called "data" where additional information

[2] https://global-internet-map-2022.telegeography.com

[3] Real Instituto el Cano: Considerations on digital strategic autonomy (Policy Paper)

or metadata can be included in order to store information, although it may increase the transaction size and associated fees. The information contained in the "data" field will be immutably recorded on the blockchain.

Due to the characteristics of blockchain and its innovations, one of the interests that researchers have shown lately is to combine the semantic web with blockchain technologies [140, 119]. The reason for this interest relies on the symbiotic relationship that enhances both technologies and the potential that can be reached by combining them [48, 26]. The semantic web is a framework for organising and linking data on the Internet, enabling machines to better understand and interpret information in a more human-like way [148]. It is based on a set of standards and technologies that allow for the creation of structured and interlinked data, known as "linked data". When combined with blockchain technology, the semantic web can benefit from the transparency and decentralisation that blockchain provides, while blockchain can benefit from the improved data quality and interoperability that the semantic web enables.

1.1 Motivation

Over the last few years, companies have taken full control of the Internet [44]. Blockchains have the potential to play a key role in decentralising the Internet, which means avoiding excessive concentration of power and influence in the hands of a few. The combination of the semantic web and blockchain starts with understanding the philosophy of the semantic web, which is centred around decentralisation and interoperability [61]. The semantic web aims to create a common framework for sharing and reusing data across different applications and platforms. It allows machines to understand and process the information on the Web, making it more intelligent and useful. However, the semantic web faces issues of trust, data integrity, and security.

Blockchain, as a decentralised and secure technology, can address these issues and provide a robust foundation for the semantic web. The semantic web and blockchain promote the idea of decentralisation, where data is not controlled by a single authority. By combining these technologies, it becomes possible to create an environment where individuals have greater control over their data, and data can be easily shared and accessed without relying on centralised platforms. Furthermore, the immutability and transparency provided by the blockchain can enhance the security and trustworthiness of the semantic web. With blockchain, every data transaction is recorded on a distributed ledger, making it difficult to tamper with. This level of security, combined with the semantic web's standardised data representation, enables users to trust the information they access, improving overall user confidence in online transactions and data sharing. In addition, by merging the standardised way of structuring and sharing data of the semantic web with the secure and tamper-proof data storage of blockchain, we can create a more interconnected and interoperable data network. This allows different platforms and applications to interact with one another, fostering collaboration and innovation.

Moreover, most blockchains have so-called smart contracts. Both the semantic web and smart contracts seek to improve automation on the Web. The semantic web allows machines to understand and process information more intelligently, while smart contracts automate the

execution of agreements and transactions based on predefined conditions. The semantic web allows data to be shared and accessed freely and more decentralised. At the same time, smart contracts operate on decentralised blockchain networks, ensuring agreements' security, transparency and integrity.

Moreover, the semantic web and blockchain have the potential to create a wide range of applications, as mentioned before. Integrating these technologies makes it possible to develop solutions that improve transparency, interoperability and security in multiple areas. For example, in supply chain management [117, 138], the combination of the semantic web and blockchain can facilitate the traceability and transparency of products throughout the supply chain. This allows stakeholders to track goods from production to consumption, ensuring the authenticity and quality of products. In the area of digital identity [49], integrating the semantic web and blockchain can enable the creation of decentralised and secure digital identity systems. These systems could control the personal data of the users, ensuring privacy and allowing for authentication and identity verification without intermediaries. Furthermore, in the field of intellectual property and copyright management [54, 87], the integration of the semantic web and blockchain can facilitate the management and protection of intellectual property. This allows creators to register, track and monetise their works more efficiently and securely. Finally, the combination of the semantic web and blockchain can empower the development of smarter and more secure Internet of Things (IoT) applications [8, 140, 141]. This enables autonomous and decentralised interaction and collaboration between devices and systems.

As more companies adopt blockchain as the underlying infrastructure for their services, new opportunities for a more decentralised Internet will open up. Blockchain technology has the potential to offer numerous societal benefits and solve several problems, as depicted in figure 1.2, such as:

- **Access to financial services**: 65% of adults currently do not have access to bank accounts [122]; in this situation, blockchain could democratise access to financial services. For example, microcredits and peer-to-peer (P2P) lending could be facilitated through blockchain platforms, allowing people who would normally not have access to traditional banking services, such as the unbanked or underbanked, to access credit. In addition, the blockchain could facilitate international remittances, which would benefit people who work abroad and send money home. By cutting out the intermediaries, transactions can be done faster and cheaper.
- **Access to education**: More than 72 million children of primary education age are not in school, and 759 million adults are illiterate and do not have the awareness necessary to improve their living conditions and those of their children [104]. Blockchain can be used to store and share academic credentials securely and verifiably. This can facilitate access to education, especially online, by making it easier for institutions and employers to verify student credentials. In addition, blockchain technology could be used to create a system of transferable academic credits, making it easier for students to move between different institutions and educational programs.
- **Access to health care**: Healthcare is one of the basic pillars of humanity and a recent

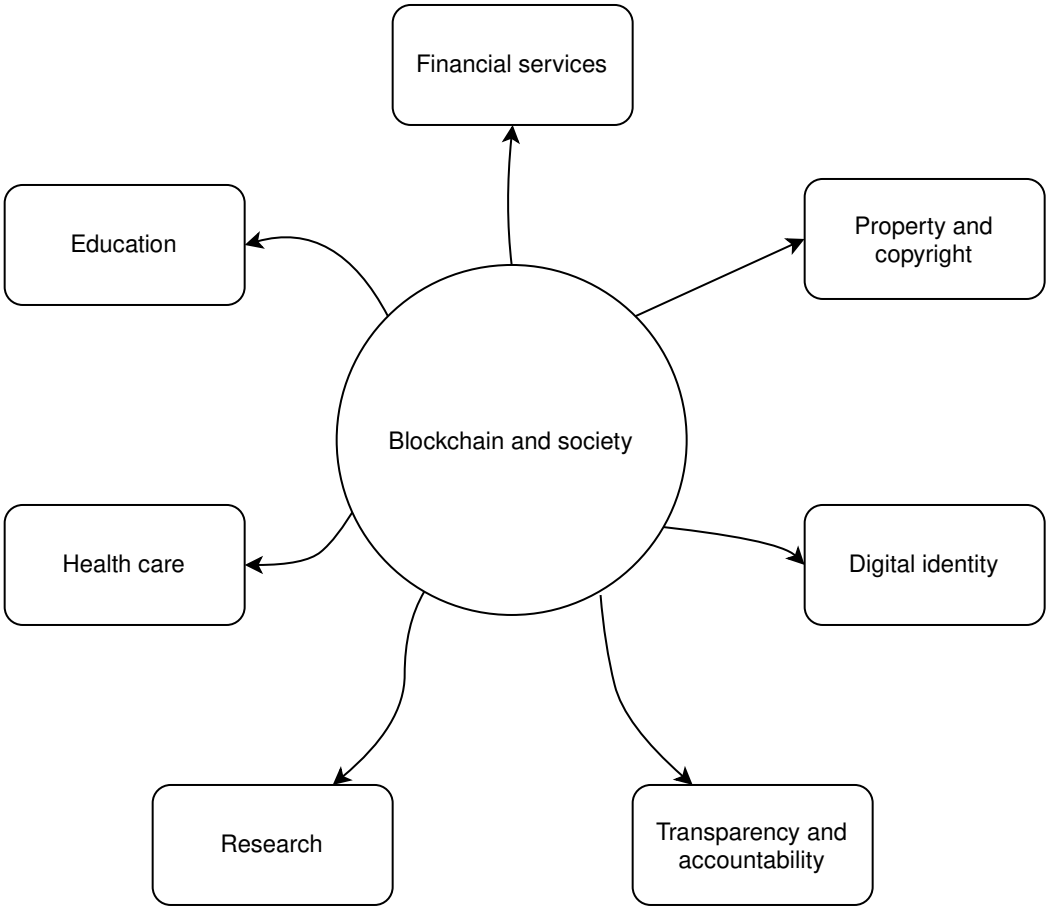

Figure 1.2: Benefits of blockchain in society

global monitoring report by the World Health Organization (WHO) reveals that half of the world still lacks coverage for the most essential health services [120]. Blockchain technology could facilitate telemedicine by providing a secure platform for sharing patient data and conducting online medical consultations. This could make medical care more accessible to people living in rural or isolated areas.

- **Research**: Research data could be securely stored on a blockchain and shared for research purposes. For example, in the medical case, patient histories, such as test results and medical records, could be stored anonymously and securely on a blockchain and shared for research purposes with the patient's consent. This could facilitate collaboration between researchers and accelerate the development of new treatments and drugs.
- **Transparency and accountability**: Blockchain can help improve transparency and accountability in different areas, from aid funding to governance. By recording transactions on a blockchain, it is possible to track exactly how funds are spent, which can help prevent corruption and ensure that resources reach those who need them most.
- **Digital identity**: Blockchain can provide a solution for those without official identity documents. By using blockchain-based digital identities, individuals can fully control

their personal data and decide who can access it. This can facilitate access to various services, from healthcare to voting.

- **Property and copyright**: Blockchain technology can help secure property and copyright rights by recording ownership of assets and working securely and transparently on the blockchain. This can protect creators and asset owners and ensure that they are properly remunerated for their work.

Moreover, blockchain, in addition to democratising access to different sectors, can offer numerous benefits in various sectors of the economy and governance, as depicted in figure 1.3, such as:

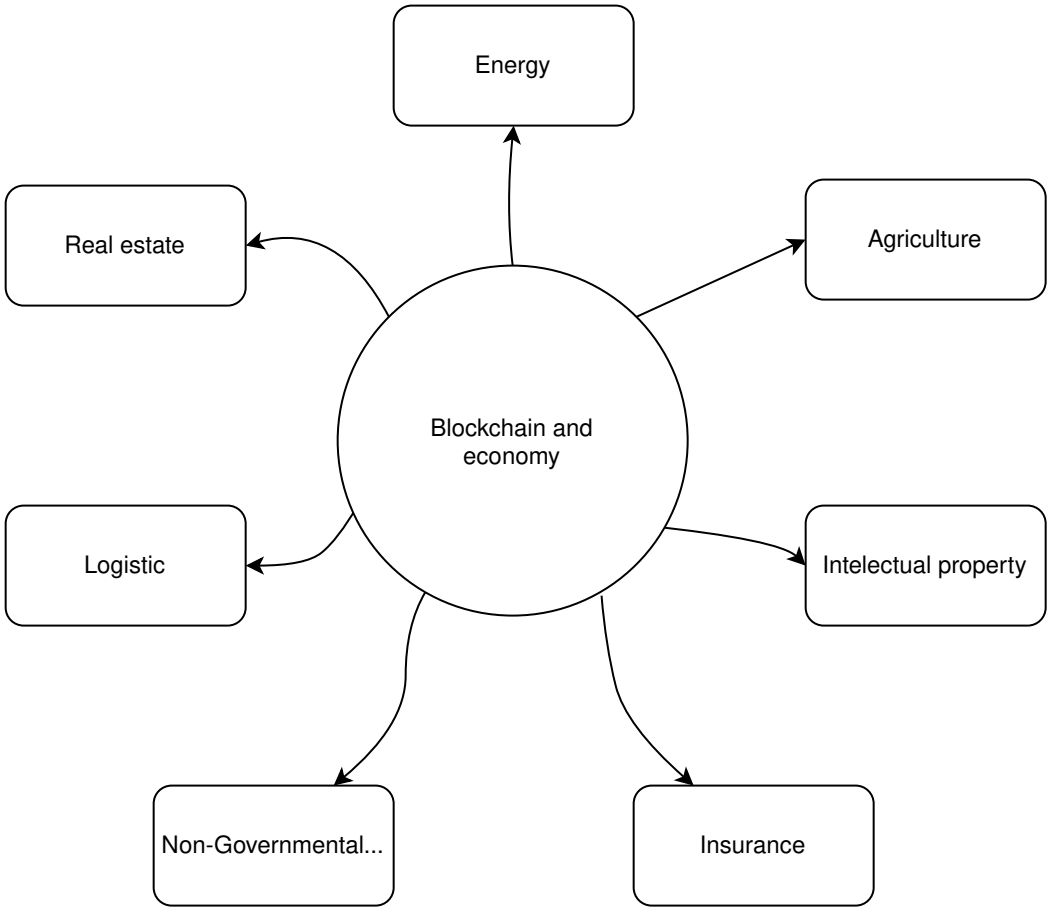

Figure 1.3: Benefits of blockchain in private or public economy sectors

- **Energy**. Blockchain can have an impact on the energy industry, especially when it comes to renewable energy. The technology can facilitate the creation of peer-to-peer (P2P) energy markets, allowing energy producers to sell surpluses directly to consumers, promoting sustainability [123]. This can help decentralise the energy grid and give consumers greater control over their energy supply. In addition, the blockchain can improve the transparency and efficiency of the energy supply chain, making it easier to trace the energy source and ensure that environmental standards are met.
- **Real estate**. Adopting blockchain technology in real estate can change how property

transactions are conducted. By recording property transactions in an immutable public ledger, the need for costly intermediaries, such as title agencies, is significantly reduced [55]. In addition, blockchain offers a level of transparency that can reduce the prevalence of fraud in the real estate industry. Finally, blockchain can simplify the process of buying and selling properties by eliminating the need for physical paperwork, as all documents can be securely stored and made accessible on the blockchain.

- **Logistic**. The blockchain can provide the supply chain and logistics with a transparent and immutable tracking system. Products can be tracked at every stage of the supply chain, from production to delivery to the consumer. This can help reduce fraud and loss of goods and ensure that products meet quality and safety standards [160]. In addition, blockchain can improve supply chain efficiency by automating processes with smart contracts, thereby reducing costs and human error.
- **Agriculture**. Blockchain technology can increase efficiency and transparency in the food supply chain. For example, producers can record their production on a blockchain from the time of harvest until it reaches the consumer. This allows effective traceability and helps to guarantee product quality and authenticity [142]. Additionally, farmers can use blockchain to access global markets and receive fair and transparent payment for their products.
- **Entertainment**. In the entertainment industry, blockchain can be used to manage intellectual property and copyright. Artists and content creators can register their works on the blockchain, guaranteeing their ownership and enabling fair distribution of royalties [166]. Blockchain can facilitate the creation of smart contracts for the sale and distribution of content, providing transparency and efficiency.
- **Insurance**. Blockchain technology can transform the insurance industry by making processes more efficient and transparent. For example, smart contracts on blockchain can automate insurance claims processes, reducing the time and cost of processing claims [114]. In addition, blockchain can be used to track the history of an insured asset, which can improve accuracy in risk assessment and help prevent fraud. A concrete example would be in auto insurance: the blockchain could contain the entire history of the vehicle (accidents, repairs, maintenance, etc.), allowing insurance companies to more accurately determine insurance premiums based on historical and current vehicle data.
- **Non-Governmental Organisations**. Non-governmental organisations and philanthropic entities can also benefit from blockchain technology. For example, it can help track donations from the donor to the final entity, ensuring that funds are used for their intended purpose and increasing transparency for donors. In addition, by reducing the need for intermediaries, more funds can go directly to the causes and people in need of help. For example, the United Nations humanitarian aid organisation World Food Programme has used blockchain to distribute cash aid to refugees, providing a secure and efficient solution that reduces transaction costs and ensures that aid reaches those who need it most [86].

In summary, if we add semantic web technologies to the blockchain, the combination of both

technologies can have a significant impact on different aspects of society and the economy, as depicted in figure 1.4, such as:

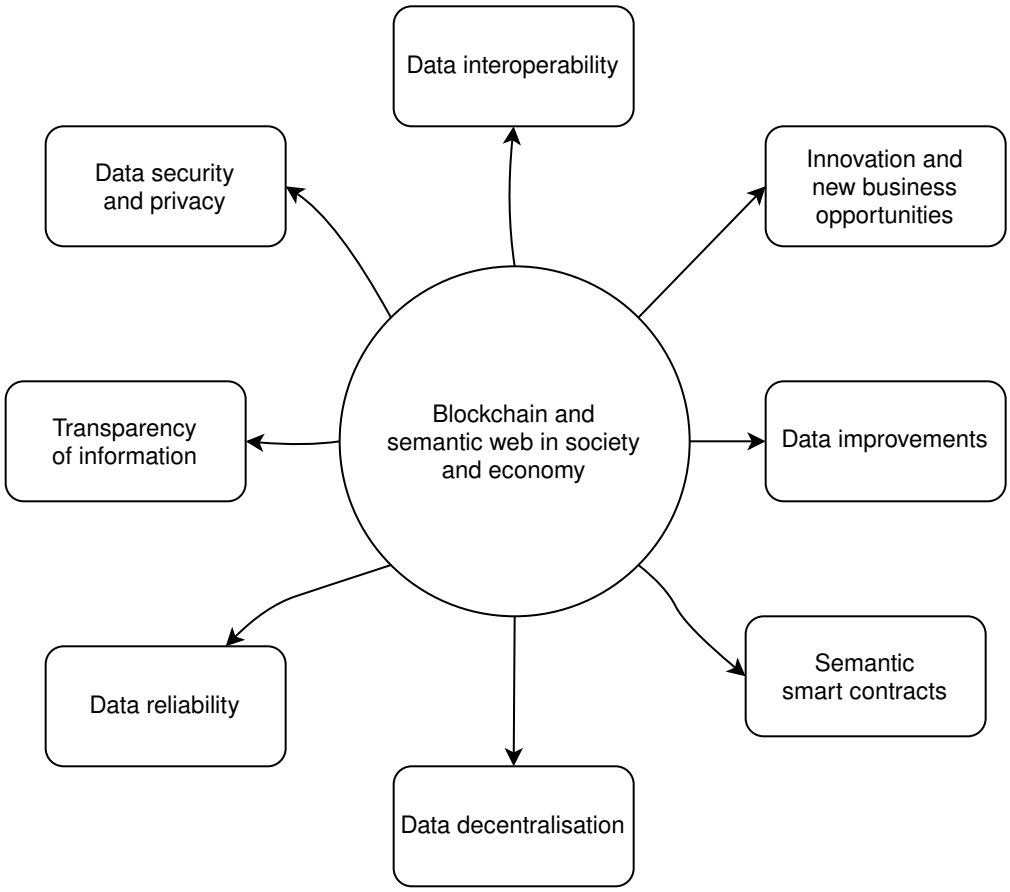

Figure 1.4: Benefits of combining blockchain and the semantic web in society and economy

- **Data interoperability**. The semantic web enables data interoperability by structuring information in a way that can be shared and understood across different platforms. Combining this with blockchain could create a system where data from different industries and sectors can be shared and used securely and efficiently.
- **Data security and privacy**. The combination of blockchain and semantic web could improve the security and privacy of online data. Blockchain could provide a secure method for storing and sharing data, while the semantic web could allow users to control exactly how and when their data is used.
- **Transparency of information**. The semantic web aims to make online data understandable and usable by machines. When combined with blockchain, which records transactions in an immutable and transparent way, it could create a system where online information is easily accessible.
- **Data reliability**. Blockchain ensures data immutability, while the semantic web aims to enhance data consistency and relevance, thereby increasing their reliability.
- **Data automation**. Blockchain allows for the development of smart contracts that

self-execute when certain conditions are met. When combined with the semantic web, these contracts can interact with a wider variety of data and perform more complex operations.

- **Data enrichment**. The ability to relate data in a blockchain to an ontology can significantly enhance the data stored in the blockchain. Data can be understood and processed in a broader context, allowing for more accurate analysis and more informed decision-making. For example, in the healthcare sector, patient data stored on the blockchain could be linked to a medical ontology to enable a more detailed analysis of health conditions and treatments.
- **Data decentralisation**. Blockchain and the semantic web promote the idea of decentralisation, where data is not controlled by a single authority.
- **Innovation and new business opportunities**. The combination of blockchain and semantic web could open up new business opportunities and innovation. It could enable the development of new services and applications that take advantage of the security, transparency and efficiency of these technologies.

1.2 Thesis phases

Current literature focuses mainly on specific applications that rely on blockchain and the semantic web, with the exception of the article written by English and colleagues, who analysed the benefits of combining these technologies. This article [48] provides an overview of what the semantic web can do for the blockchain and vice-versa; nevertheless, their work focuses on covering a large number of topics, and not only the benefits, and thus, they lack an in-depth analysis of the benefits and the scenarios in which both technologies can be combined. This thesis covers this analysis, providing an in-depth study of the benefits that both technologies can provide to each other and a categorisation of the scenarios in which the two technologies can be combined. This is the starting point of the thesis, studying how to merge the semantic web and the blockchain, establishing the basis of how to combine these two technologies and experimentally evaluating this combination. From the analyses performed in these experiments, some first steps are given for the creation of a semantic blockchain. For these reasons, we organise the contributions of this thesis into three different phases, as depicted in figure 1.5.

1.2.1 Synergies between blockchain and the semantic web

The first phase focuses on the interplay and synergies between blockchain and semantic web technologies. Due to the fact that blockchain is a decentralised technology, and the semantic web is deliberately designed to be decentralised, it can help to reduce the reliance on centralised authorities, enabling more open and transparent data sharing and management, fitting in with the idea of the democratic network philosophy [113], but this is not exclusive and more benefits can be found between these technologies.

Analysing and understanding the alignment and benefits that emerge from the intersection

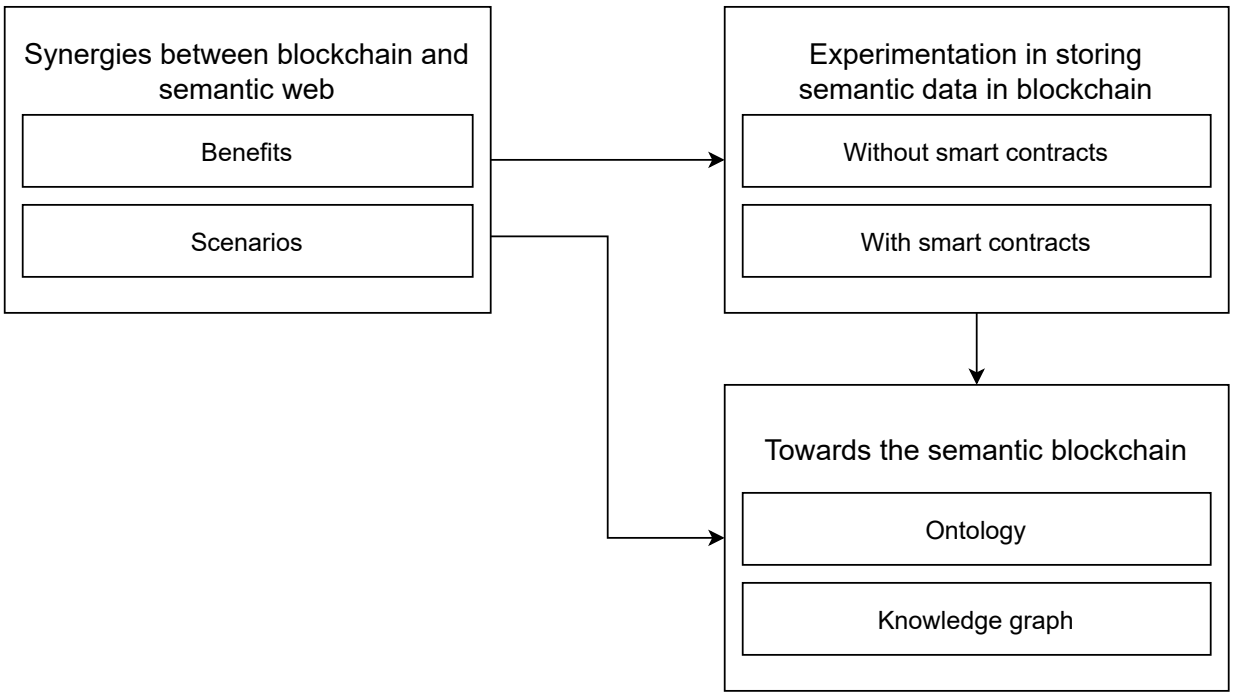

Figure 1.5: Phases of the thesis

of these two technologies is central to this first phase. Therefore, the thesis starts with an analysis of the benefits that the blockchain can provide to the semantic web and vice-versa. Additionally, a set of scenarios is proposed to encapsulate the possible ways these technologies can be intertwined. Each scenario represents a unique approach to leverage the distinctive characteristics of both blockchain and semantic web technologies to enhance decentralised data systems.

As such, this first phase sets the conceptual foundations for the thesis and leads the way to more experimental and implementation-focused work explored in the subsequent phases.

1.2.2 Experimentation in storing semantic data into blockchain

After recognising the synergies between the semantic web and blockchain and envisioning how these technologies can be combined, the next step is to experiment on the basis of the scenarios proposed, as multiple scenarios were identified. Since blockchain was not initially designed to host data using semantic web standards (i.e., in the RDF format), efficient storage of such data requires particular strategies to ensure compatibility, efficiency and optimised functionality, analysing the strengths and weaknesses of each scenario.

The second phase experimentally addresses two different strategies for introducing RDF into the blockchain: storing RDF data directly into blocks or via smart contracts. The conclusions obtained through experimentation provide a practical response to the theoretical foundations established in the first phase and lead to the final phase of this thesis.

1.2.3 Towards the semantic blockchain

The results obtained in the previous experiments will lead us to the last phase, where the methods needed to represent and simulate a semantic blockchain will be created. This semantic blockchain will be based on Ethereum and will require different ontologies to represent the entire blockchain (i.e. the information of the blocks), the information stored in the blockchain, and the source code of smart contracts.

Smart contracts are usually part of a domain [103, 10], such as agreements on music rights, video games, supply chains, commercial transactions, etc. In programming languages, these domains can be represented using Knowledge Representation (a field of Artificial Intelligence dedicated to representing information of a domain and usually used to capture information about the world for solving complex problems), ensuring that the meaning and effect of contracts are preserved when they are translated into code. One technique for representing knowledge is the use of ontologies, providing the ability to process semantics since ontologies encompass a representation, formal naming, and definition of categories, properties, and relations between concepts, data, and entities that substantiate one, many or all domains of discourse.

One domain of knowledge representation is programming languages (notations for writing computer programs), for which ontologies already exist that cover or model popular programming languages [1], such as Java [83] and C [155]. The use of Knowledge Representation applied to programming languages has improved current educational techniques and complemented some of these languages, providing additional parsers, code generators, or code validation. However, despite the popularity of smart contracts and the fact that they are also programmes and thus expressed according to one programming language, there is no ontology orientated to represent smart contracts themselves.

Smart contracts could benefit from the knowledge representation capabilities provided by ontologies in areas such as data validation, performing richer queries over the smart contract (and the blockchain), or interoperability between different smart contract languages, among others. In this thesis, an ontology has been developed for a smart contract language, Solidity, which is the main smart contract language for one of the most relevant blockchain technologies, Ethereum [51], bringing knowledge representation for this language, linking the ontology to standard ontologies and making its knowledge reusable.

In addition, some ontologies were developed to represent different functionalities or properties of the blockchain [39, 16, 154]. However, these ontologies either represent the inner workings of blockchains (representing the iterations that occur with block creation) or represent part of blockchains (being ontologies that attempt to represent all blockchains but fall short of representing all the features that all blockchains have, such as Ethereum). Therefore, in this thesis, two more ontologies have been developed to represent the Ethereum blockchain: one to represent the blocks with all the Ethereum properties and another one to represent the iterations that take place in the contracts.

Using these ontologies, we have built a knowledge graph of the Ethereum blockchain and Solidity smart contracts deployed on the Ethereum blockchain. Having the Ethereum blockchain and Solidity smart contracts represented using RDF enables several benefits, such as querying

the contracts using SPARQL, providing richer data by linking the RDF smart contract instances to other datasets or aligning the knowledge graph ontology to others, and allowing semantic interoperability.

Once we have built the knowledge graph with the Ethereum blocks and Solidity contracts, we can store all the data related to the blockchain in this knowledge graph and relate the blocks with the smart contracts (each block has the records of the transactions) and thus simulate what a fully semantic blockchain could potentially look like.

1.3 Thesis structure

The remainder of the thesis is organised as follows:

- Chapter 2 analyses the current state of the art directly aligned to the topics of this work: semantic web and its main standard technologies, and blockchain and its principal components. This chapter also analyses the current limitations of existing approaches, which conduct the contributions of the presented work.
- Chapter 3 describes the objectives and main contributions of this thesis. Additionally, the assumptions, hypothesis, and restrictions of the work are presented.
- Chapter 4 analyses the benefits that the blockchain provides to the semantic web and vice-versa and proposes a series of scenarios where the blockchain and the semantic web can be combined.
- Chapter 5 provides an analysis of the cost of storing RDF and non-RDF data into the blockchain directly, analysing the advantages and disadvantages of this process, and analyses the cost of storing RDF data on the blockchain through a smart contract, comparing this storage with other types and analysing the advantages and disadvantages of this method.
- Chapter 6 describes an ontology of the block, ABI, and Solidity smart contract language and the advantages that an ontology brings to the blockchain and smart contracts.
- Chapter 7 provides the method to store in a semantic way and query the Ethereum blockchain and the Solidity smart contracts deployed.
- Chapter 8 describes the main conclusions of this thesis and identifies the future lines of research in the topic of blockchain and the semantic web.

1.4 Publications

The work presented in this thesis has been disseminated in the following international workshops, conferences and journals:

- The synergies between the semantic web and blockchain, and some of the scenarios where both technologies can be combined, located in chapter 4, have been published in: Juan Cano-Benito, Andrea Cimmino, and Raúl García-Castro. "Towards blockchain

and semantic web." Business Information Systems Workshops: BIS 2019 International Workshops, Seville, Spain, June 26–28, 2019, Revised Papers 22. Springer International Publishing, 2019.

- The study of inserting RDF and non-RDF data directly into the blockchain, located in chapter 5, has been published in: Juan Cano-Benito, Andrea Cimmino, and Raúl García-Castro. "Benchmarking the efficiency of RDF-based access for blockchain environments." 32st International Conference on Software Engineering and Knowledge Engineering (SEKE). 2020.
- The Solidity ontology, presented in chapter 6, has been published in: Juan Cano-Benito , Andrea Cimmino, and Raúl García-Castro. "Toward the Ontological Modeling of Smart Contracts: A Solidity Use Case." IEEE Access 9 (2021): 140156-140172.

1.5 Ontologies

In this thesis, three ontologies have been developed. The first one represents the entire Solidity language, which is the programming language used to write smart contracts on the Ethereum blockchain. This ontology acts as a data model that defines the structure and relationships between the elements that make up a smart contract written in Solidity. This can include the functions, variables, events, and other components that are integral to any smart contract. The ontology is available online via Github[4] and Zenodo[5] with a Creative Commons Attribution 4.0 International licence.

The second ontology represents the code stored on the Ethereum blockchain. In Ethereum, once the contract is put on the chain, it will store the main functions in a specific format indicating the functions and the input and output parameters of each function. The ontology is available online via Github and Zenodo[6] with a Creative Commons Attribution 4.0 International licence.

The last ontology represents the data of the Ethereum blocks, including transactions and withdrawals. The ontology is available online via Github[7] and Zenodo[8] with a Creative Commons Attribution 4.0 International licence.

1.6 Software

In the context of this thesis, the software called "Sancus" has been developed. This software is designed firstly to convert smart contracts into RDF. Sancus serves as a bridge between the smart contracts and the semantic web, effectively translating between the two domains. Sancus is available online via Github[9] with an Apache 2.0 licence.

[4] https://github.com/oeg-upm/Solidity-ontology
[5] https://zenodo.org/records/10635332
[6] https://zenodo.org/records/10707050
[7] https://github.com/oeg-upm/Solidity-ABI-ontology
[8] https://zenodo.org/records/10719799
[9] https://github.com/oeg-upm/Sancus

1.7 Knowledge graph

In this thesis, a knowledge graph has been created and is available online[10]. This knowledge graph represents part of the blockchain and the contracts stored in it, emulating a semantic blockchain. To create the public dataset of the knowledge graph, licenced contracts have been used for publication, and this dataset (which contains 2,608 blocks, 8,283 ABI contracts, and 36,047 contracts) was published on Zenodo[11].

1.8 Research stay

The main motivation during my three-month research stay at the Knowledge Media Institute (KMi), a laboratory belonging to The Open University located in Milton Keynes (UK), was to participate in projects related to blockchain and the semantic web. As a relatively new field, more and more researchers are gradually focussing on this combination of technologies, and the KMi group is one of them. During this stay, I participated in the "Ethical Black Box" project, which analyses the ethical use of SPARQL queries using blockchain technology (Ethereum) and in which leading researchers from other universities collaborate.

During my research stay, I was asked to collaborate as an organiser in two workshops. On the one hand, in the "Trusting Decentralised Knowledge Graphs and Web Data (TrusDeKW)" workshop at the Extended Semantic Web Conference (ESWC) 2023 conference, whose objective was to analyse the decentralisation of the Web and the problems associated with decentralised knowledge graphs, dealing with issues such as trust, verification, and validation of data when these are decentralised. On the other hand, I also participated in another TrusDeKW workshop during the International World Wide Web Conference 2023. The aim of this workshop was to showcase and explore various aspects of web technology, such as federated queries, RDF and distributed ledgers. Discussions focused on how these technologies can help overcome challenges related to the over-centralisation of web data, such as the misuse of personal data and the opacity of automated decision-making processes.

[10] https://kg.dlt.linkeddata.es
[11] https://zenodo.org/records/10719604

Chapter 2

State of the art

> The best way to predict the future is to invent it.
>
> *Alan Kay*

This chapter is structured into four sections. Each section addresses a specific aspect of the research landscape. Section 2.1 focuses on the semantic web, providing an overview of the concepts and principles that are essential to understanding in the context of this thesis, while section 2.2 explores the fundamental concepts and technologies related to blockchain. Third, in section 2.3, we analyse the related work concerning the combination of semantic web and blockchain technologies, showcasing the current state of research in this interdisciplinary area. Finally, section 2.3.4 presents the conclusions of the actual state of the art and why this thesis is an important contribution to the area of blockchain and the semantic web.

In addition to these core sections, another well-known concept, multi-agent systems, has been used. A software agent is a computer system capable of autonomous actions in a tailored-domain environment [171]. The means of the actions performed by an agent have the goal of meeting a set of design requirements. A system with two or more agents is called a multi-agent system. In the context of this thesis, the type of agents used are proactive agents with simple reflexes that are based on condition-action.

2.1 Semantic web

The semantic web aims to enable machines to understand and interpret the meaning of the data and content available online. Proposed by Tim Berners-Lee, the inventor of the World Wide Web, in collaboration with other researchers in the late 1990s, the semantic web seeks to enhance the current web infrastructure by adding a layer of machine-readable metadata, facilitating more efficient and intelligent data integration, sharing, and processing [14].

At the core of the semantic web are several well-known fundamental concepts and technologies, including Resource Description Framework (RDF), Web Ontology Language (OWL), and SPARQL Protocol and RDF Query Language (SPARQL). Over the years, the semantic

web has evolved and matured with the development and adoption of new standards, tools, and techniques. One notable advancement is the introduction of linked data, a set of best practices for publishing and connecting structured data on the Web using RDF, URIs, and HTTP. Linked data has led to the creation of the Linked Open Data (LOD) cloud, which consists of an ever-growing collection of interlinked datasets spanning various domains, such as government, life sciences, and media. The stack of the semantic web technology is depicted in figure 2.1.

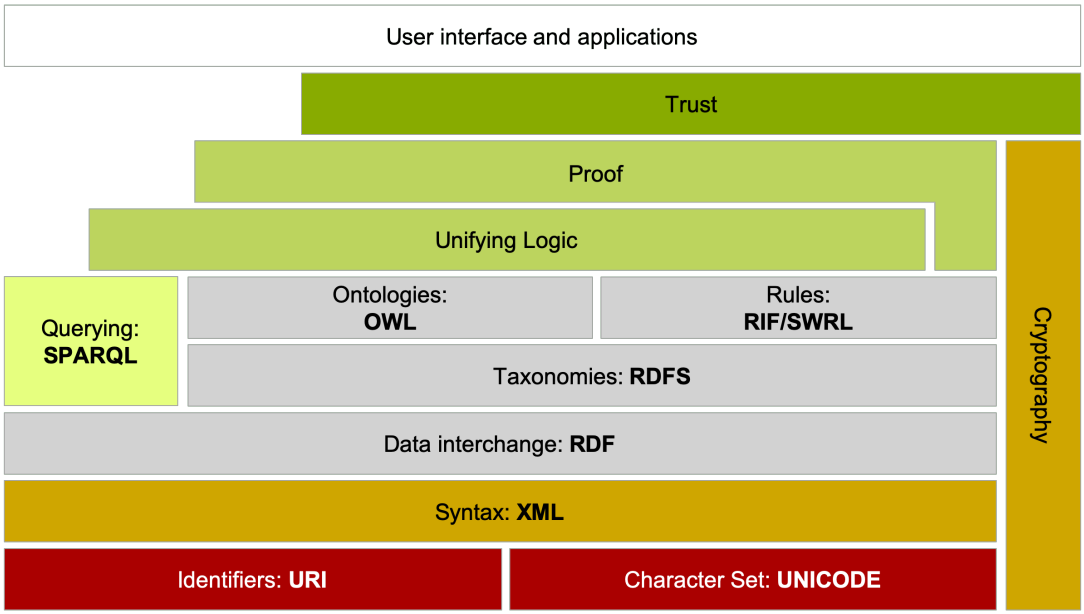

Figure 2.1: Semantic web stack [71]

Another significant development in the semantic web field is the incorporation of machine learning and artificial intelligence techniques to facilitate tasks like ontology learning, entity linking, and semantic annotation. These approaches have helped bridge the gap between unstructured data and structured semantic web representations, making it easier to extract and integrate valuable knowledge from diverse data sources.

2.1.1 URIs

A URI is a character string used to identify a name or a resource on the Internet. In the realm of linked data, each entity (e.g., an individual, place, or concept) is distinctly identified via a URI. This standard provides a consistent mechanism for machines to access and manipulate the information of the entity [13].

Each URI is unique to the resource it embodies, meaning that if two individuals or machines refer to the same URI, they are indisputably referring to the same resource. Furthermore, URIs in linked data are typically dereferenceable, meaning they can be used to access the resource's representation.

For instance, the URI `"http://dbpedia.org/resource/Albert_Einstein"` corresponds to the entity Albert Einstein in the DBpedia knowledge graph. This URI will be redirected to

the Albert Einstein DBpedia page, which hosts detailed, interconnected information about him.

A URI is bifurcated into two main sections [168]:

- Scheme: The scheme identifies the internet protocol to be employed. In the case of linked data, the most commonly utilised scheme is HTTP.
- Specific part: The specific part of the URI identifies the resource within the scope of the scheme and the naming authority.

To ensure that a dataset adheres to the principles of linked data, it is imperative that each entity is represented with a unique URI [100]. This allows entities to be easily referenceable, discoverable, and linkable on the web of data.

One of the advantages of using URIs is that they provide a standardised way of naming and accessing resources, regardless of where they are stored or how they are accessed. This makes linked data particularly valuable for connecting and amalgamating disparate datasets found in different locations on the Web [18].

2.1.2 RDF

The Resource Description Framework (RDF) is a fundamental technology and data model for the semantic web, designed to represent information in a machine-readable and interoperable format [22]. RDF was developed by the World Wide Web Consortium (W3C) and has become a widely adopted standard for encoding knowledge and metadata on the Web. Its primary goal is to facilitate the integration, sharing, and processing of structured data across different applications and domains.

RDF is based on the concept of subject-predicate-object triples, which express statements about resources in the form of directed, labelled graphs [65]. In RDF, resources are identified by Internationalised Resource Identifiers (IRIs), while predicates (also known as properties) represent the relationships between resources [161]. The object of a triple can be another resource or a literal value, such as a string, number, or date. RDF's simple yet expressive data model enables the description of complex relationships and the creation of rich, interconnected knowledge structures.

There are multiple standardisations of RDF serialisation formats, such as RDF/XML, Turtle, N-Triples, and JSON-LD. These formats enable the interchange of RDF data between different applications and systems, further promoting the vision of a more interconnected and machine-readable Web [159].

The growth of the linked data movement and the creation of large-scale RDF datasets, such as DBpedia, YAGO and Wikidata [127], demonstrate the potential of RDF to play a central role in the future of web-based knowledge management and intelligent information systems.

2.1.3 Linked data

The RDF data published following the principles proposed by Tim Berners-Lee [19] is known as linked data. In summary, linked data is simply about using the Web to create typed links between data from different sources. These principles refer to a set of data quality requirements that the data must meet:

1. Use URIs as names for resources.
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 (RDF, SPARQL).
4. Include links to other URIs so that they can discover more things.

According to these principles, two features must be remarked; on the one hand, the resources are identified relying on URIs, which are dependent on the domain name space (DNS) used to publish the data. This means that if the DNS changes the namespace provided, then the resources will no longer be identified by these URIs nor retrievable. On the other hand, the data is stored following a decentralised approach. However, when consuming such data, it will appear as a unique dataset thanks to the links between datasets and the online availability of the resources.

2.1.4 Knowledge graphs

Knowledge graphs are a representation of structured and interconnected knowledge used to store and manage complex relationships between entities in a machine-readable and semantically rich format. They provide a flexible and intuitive way to model real-world information, enabling machines to reason about and infer new knowledge from existing data. Knowledge graphs have become increasingly popular for integrating and organising information from diverse sources, supporting various applications, such as natural language processing, recommendation systems, and intelligent search.

Knowledge graphs are closely related to the semantic web, as they often leverage the underlying technologies and concepts, such as Resource Description Framework (RDF) [110], the Web Ontology Language (OWL) [99], and linked data [19]. These technologies enable the creation of rich semantic relationships and the interoperability of data across different knowledge graphs, facilitating the integration and reuse of information in a standardised and machine-readable format.

Over time, knowledge graphs have evolved in terms of size, complexity, and application. One of the most notable developments in the field is the creation of large-scale, publicly available knowledge graphs, such as DBpedia or Wikidata. These resources have significantly advanced the state of the art by providing vast amounts of structured and interconnected information, covering a wide range of domains and enabling a multitude of applications and research.

Knowledge graphs are usually represented using ontologies. One characteristic of ontologies is that they support reasoning over the data they model (by using the semantic web tools).

They allow data validation or generate new data that is not explicitly defined or stored using reasoning mechanisms. The literature has multiple standard ready-to-use ontologies for a wide range of domains [164].

2.1.5 Storing linked data

A triplestore, also known as an RDF store, is a specialised database management system designed to store, manage, and query RDF data in the context of the semantic web [102]. Triplestores are specifically optimised for handling RDF triples, representing information as subject-predicate-object statements, providing a scalable and efficient way to manage structured, semantically rich data. The primary goal of triplestores is to facilitate the integration, exploration, and analysis of complex, interconnected knowledge structures in various applications and domains [50].

In the last years, triplestores have experienced improvements in terms of scalability, performance, and functionality [30]. Early iterations of triplestores often grappled with storage efficiency and query performance issues, especially when managing large-scale RDF datasets [137]. However, recent advancements in indexing methods, query optimisation strategies, and distributed computing have given rise to more robust and scalable triplestore solutions. These enhancements now enable the handling of billions of RDF triples and the execution of complex SPARQL queries with greater efficiency [72, 169].

2.1.6 Accessing Linked Data

The semantic web is envisioned as a decentralised worldwide information space for sharing machine-readable data with a minimum integration cost. Its two core challenges are the distributed modelling of the world with a shared data model and the infrastructure where data and schemas can be published, found and used. Users benefit from getting information "raw and now" and in portable data formats. Providers often publish data embedded in a fixed user interface, e.g., in HTML.

Dereferenceable Uniform Resource Identifier (URIs) Internationalised Resource Identifiers (IRIs)

The Web makes use of the URI as a single global identification system. The global scope of URIs promotes large-scale "network effects". Therefore, in order to benefit from the value of linked data, government and governmental agencies need to identify their resources using URIs. While the use of URIs provides a unified system of identification on the Web, not all URIs are created equal. To truly harness the potential of linked data and ensure consistency, longevity, and accessibility of resources, especially in contexts like governmental agencies, there is a need for "Cool URIs"[1]. These are URIs designed with specific principles in mind to make them more effective and robust for the Semantic Web. The criteria defining a "Cool URI" ensure that the resources remain interconnected, unambiguous, and easily retrievable, fostering an integrated and efficient web of data[2].

[1] https://www.w3.org/TR/cooluris/
[2] https://www.w3.org/TR/ld-bp/#HTTP-URIS

The concept of "Cool URIs" is central to the semantic web and the realm of linked data. URIs, which stand for Uniform Resource Identifiers, are used to identify resources uniquely on the web. In the context of linked data and the semantic web, URIs have an essential role in establishing connections and relationships between diverse pieces of information. Some principles of URI design are [144, 20]:

- **Use a recognised standard format:** The data must be structured using recognised standards so that computers interrogating the data can process it consistently. For example, using standard protocols (like HTTP) ensures compatibility and accessibility across various platforms and tools.
- **Dereferencing URIs:** Cool URIs in the semantic web context should be dereferenceable, meaning that when accessed, they should provide useful information, ideally in RDF format, about the resource they identify.
- **Persistent URIs:** A cool URI should remain unchanged over time. This ensures that once data is linked to a particular URI, the link does not break due to changes in the URI, maintaining the integrity of data on the Web.
- **Descriptive URIs:** A URI should unambiguously identify a single resource. This ensures clarity and precision when linking and referencing data.

The design and choice of URIs are important in linked data because they ensure not only the accessibility of data but also its longevity and consistency. When data from various sources are interconnected through these URIs, a larger and more cohesive data network is created, which is why "Cool URIs" are an important part of the semantic web.

To further enhance this system, IRIs have been introduced, extending the functionality of URIs to better support international characters. This allows resources to be named in local languages, thus broadening the accessibility and inclusivity of the Web. In the context of the semantic web, IRIs are beneficial because they support identifiers that include characters from a wide range of writing systems [53, 45].

Data dump

In the sphere of linked data, data dumps constitute a common method for disseminating sizeable datasets. These dumps are files that hold a representation of a linked dataset in a specific format, such as Resource Description Framework (RDF), which can then be downloaded and processed locally.

Data dumps essentially facilitate the bulk transfer of linked data. They provide a convenient, one-off delivery mechanism that allows the full acquisition of a dataset without needing to individually access each URI. This approach can be instrumental when network connectivity is a challenge or when the computational resources for handling repeated dereferencing operations are limited.

For instance, a linked data dump from a bibliographic database could include RDF triples representing each book (subject), its authorship (predicate), and the corresponding author (object). Such triples can be readily processed by RDF-aware software, thereby allowing the

semantic enrichment and interlinking of the represented data.

While data dumps provide a viable solution for data access and exchange, they do come with a few challenges. Data dumps can be large and require significant storage and processing power. Also, maintaining the freshness of data obtained from data dumps can be a non-trivial task as data on the Web evolves and updates.

In conclusion, data dumps represent a crucial tool for the publication and consumption of linked data, allowing for the efficient transfer of large volumes of semantically rich data across diverse systems. However, considerations regarding data size, processing requirements, and data currency need to be carefully managed.

Linked Data Platform

The Linked Data Platform, or LDP, is a standard proposed by the World Wide Web Consortium (W3C) to facilitate the publication and interchange of linked data on the Web. It defines rules for managing collections of RDF resources and handling links between resources. LDP allows the creation, read, update, and deletion of RDF data using standard HTTP methods.

LDP servers expose their resources (or linked data) for clients to read and write. These resources are organised into LDP containers, which can be thought of as directories or folders. Each LDP container can contain RDF source resources or other containers, enabling a hierarchical organisation of the resources, similar to a file system.

Moreover, LDP provides a standard way for clients to discover how to interact with resources. By using standard HTTP headers and methods (GET, POST, PUT, DELETE), along with linking mechanisms from the HTTP protocol, LDP allows clients to navigate and manipulate the resources.

One significant advantage of using LDP is its integration with the HTTP protocol, which is ubiquitously supported across different systems and platforms. This means LDP can be used to create web applications that are capable of reading and writing linked data, making it a powerful tool for building data-driven applications on the Web.

LDP also supports the representation of resources in different RDF serialisations, thereby facilitating integration with existing linked data applications. This extensibility makes LDP an adaptable standard that can fit various use cases.

SPARQL

The SPARQL Protocol and RDF Query Language (SPARQL) is a query language and protocol for retrieving and manipulating RDF data in the context of the semantic web [62]. Developed by the W3C, SPARQL enables users to express complex queries over RDF graphs, allowing them to extract and transform information from multiple interconnected data sources. Its primary goal is to facilitate the exploration, integration, and analysis of structured data across various domains and applications.

SPARQL is built upon several core concepts, including variables, triple patterns, and solution modifiers. Variables are placeholders that can be matched against RDF terms (IRIs, literals,

and blank nodes) in a query. Triple patterns are similar to RDF triples but may contain variables instead of fixed terms. Solution modifiers, such as SELECT, CONSTRUCT, ASK, and DESCRIBE, define the output format and structure of the query results. Additionally, SPARQL supports advanced query features, such as optional patterns, unions, filters, and aggregations, enabling users to express sophisticated data retrieval and transformation operations. SPARQL can query across multiple and interconnected RDF datasets (federated queries), further contributing to the vision of a more interconnected and machine-readable Web [23].

Materialisation and virtualisation

On the one hand, RDF materialisation is a technique used in the semantic data integration context [64]. Usually, it refers to a piece of software connected to a data source and with a set of translation rules called mappings. These techniques are able to translate on-the-fly data in heterogeneous formats and models into RDF triples that represent the information in the original data but expressed according to a specific ontology, allowing a user to solve SPARQL queries over such data.

This technique can facilitate data interoperability and the use of semantic technologies on the Web. However, materialisation can be costly in terms of storage and processing time, as a new copy of the data needs to be created and stored. Also, any changes in the original data need to be reflected in the materialised RDF data, which may require frequent and expensive updates.

On the other hand, Virtual RDF is an alternative technique to materialisation that involves exposing existing data as RDF without physically converting it into the RDF format [93]. Instead of converting and storing the data as RDF triples, the data is "mapped" to an RDF model and converted into RDF format only when queried.

This is done through an RDF mapper that understands how data in the original format corresponds to the RDF model. When an RDF query is made, the mapper translates the query into the query language of the original storage system, fetches the results, and then converts them into RDF to return them.

The advantage of RDF virtualisation is that it allows users and applications to deal with the data as if it were RDF, without the storage and processing costs associated with materialisation. However, virtualisation may have slower performance in some cases, as queries require real-time translation and conversion of the data.

2.2 Blockchain

Blockchain consists of a data structure and a protocol for distributed transactional systems created in 2009 to operate with Bitcoin [107]. Unlike traditional distributed databases, a distributed consensus protocol avoids intermediaries by approving transactions.

All blockchain implementations follow a common pattern: the information contained in the chain is stored in blocks, and each block has a unique hash calculated with the content of the

block and the hash of the previous block, so the modification of one block of the chain requires the modification of all subsequent blocks, preventing the manipulation of blocks without consensus between nodes, as depicted by figure 2.2. Thus, the blockchain can be seen as a database shared between several parties that validates its content without the intervention of a third party.

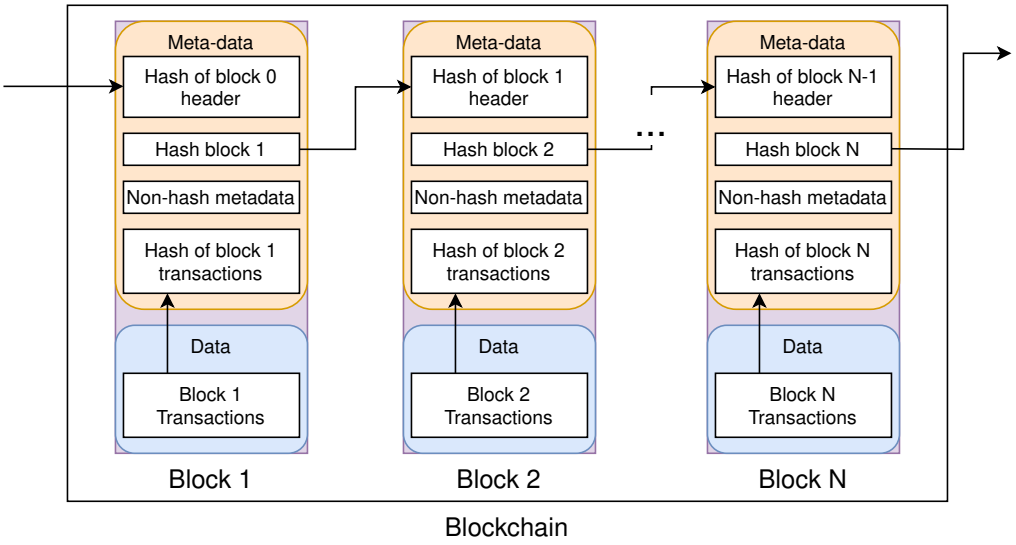

Figure 2.2: Blockchain common pattern

Blockchain has several implementations, e.g. Bitcoin or Ethereum, each of which has some differences. Nevertheless, all these implementations follow a common bottom line. The blockchain can be seen as a shared database among several peers who validate its content without the intervention of a third party. Some of the key concepts of blockchain are the following:

A **chain** is a list of blocks. When a peer aims at adding a new block, a specific procedure has to be followed: the hash of the new block is computed relying on the hash of the last block in the chain, and then the block is forwarded to all the peers who share the chain and is included only when the rest of the peers validate the new block. In addition, when something is included in the chain, it becomes immutable; thus, that information is no longer modifiable by anyone.

Blocks are the minimal storage unit of the blockchain. They have some meta-data, like the hash of the previous block, the hash of the current block, and the current data, i.e., their content. The content can be expressed in multiple formats, regardless of whether another block in the same chain uses a different one. Depending on the implementation, the meta-data may change. In addition, the content of a block is limited to a maximum amount of data that may change depending on the implementation; e.g., Bitcoin supports up to 80 bytes [9]; in contrast, Ethereum supports only 32 bytes [170]. Every block consists of two different fields:

- **Metadata**. It is stored in the block header, which is used by nodes to verify the data of a block. The metadata of a block contains various fields, such as the version of the blockchain, the hash of the current block, the hash of the previous block, or the time

when a block is created, inter alia.

- **Data**. Includes information about the transactions of the blockchain (smart contracts or transactions between nodes).

The name of the operation that writes or stores data inside a blockchain is called **transaction**. Depending on the size of the data being written, it requires more or less space in one block. This is a drawback due to the fact that blocks have a maximum size. As a result, if transactions require more space than the one available in a block, they will be written in more than one block.

Usually, a transaction has a virtual cost since it requires a certain amount of computing power. As a result, performing a transaction has an associated cost in public blockchains. Depending on the implementation of the chain this cost is commonly known as gas [170].

Depending on the peers and the grants to read/write in the chain that they have, three types of blockchains are distinguished [121]:

- **Public blockchain**: A public blockchain is an open blockchain platform in which any individual can participate, either to read or write blocks on the blockchain. This type of blockchain is fully decentralised and has no specific owner. Examples of public blockchains are Bitcoin and Ethereum.
- **Consortium blockchain**: In a consortium blockchain, the rights to read the blockchain are granted at the invitation of an administrator. This is a semi-private form of blockchain, generally used by a group of organisations working together to achieve a common goal. Once the invitation has been granted, all peers can write new blocks to the chain. This type of blockchain is less decentralised than the public blockchain but provides more efficiency in terms of transaction speed and energy consumption. In addition, it also allows for a higher level of privacy by limiting the visibility of information. Ethereum is a good example of this type of blockchain that can be adapted to consortium uses.
- **Private blockchain**: A private blockchain is similar to a consortium blockchain but with additional restrictions. In this setup, only a limited number of entities are allowed to participate in the network. Even after receiving an invitation from an administrator to read the chain, additional permissions must be granted for a peer to write to the chain. This type of blockchain is the most centralised of the three and is commonly used in enterprise environments where confidentiality and control are important. It provides the highest privacy and security but at the cost of decentralisation. An example of a private blockchain could be Hyperledger Fabric, which is popular in enterprise applications.

On the one hand, a well-known drawback of the blockchain is its scalability; blocks with large amounts of data entail that a larger storage space is required, and thus, the propagation of the block to the rest of the peers in the network will be slower [176]. On the other hand, since the chain is shared by several peers and the blocks are built relying on the previous ones, the blockchain is more secure than other data stores [96].

2.2.1 Blockchain technologies

There are many blockchain technologies, each with its own unique strengths, weaknesses and applications.

Bitcoin

Bitcoin was created in 2008 by an unknown person or group of people using the name Satoshi Nakamoto, introducing the world to blockchain technology. In the same year, the Bitcoin whitepaper titled "Bitcoin: A Peer-to-Peer Electronic Cash System" was published [108]. This paper outlined the theoretical framework for a decentralised peer-to-peer digital cash system that did not require intermediaries like banks or governments. The first block of the Bitcoin blockchain, known as the Genesis Block or Block 0, was mined by Satoshi in January 2009.

In Bitcoin, users use computers to solve complex mathematical problems that validate each transaction and add it to the blockchain. These users (called miners) are then rewarded with new bitcoins, a process known as "mining", and this is how new bitcoins enter circulation. This consensus between the users in order to validate new blocks is called Proof of Work (PoW).

Ethereum

Ethereum is an open-source, decentralised, blockchain-based platform that supports smart contracts and decentralised applications (dApps). Launched in 2015 by a team led by Vitalik Buterin [24], its primary innovation lies in the use of smart contracts, which are self-executing agreements with the terms of the contract directly written into code [25]. This enables trustless and decentralised interactions among parties, removing the need for centralised intermediaries.

Over the years, Ethereum has undergone significant upgrades and changes in its architecture. One of the most notable milestones in Ethereum's history is the transition from Ethereum 1.0 to Ethereum 2.0, also known as "Eth2" or "Serenity" [143]. This ambitious upgrade aims to address the network scalability, security, and sustainability issues. Ethereum 2.0 introduces several key changes, including the switch from a Proof of Work (PoW) consensus mechanism to a Proof of Stake (PoS) model, removing the miners from the equation. This change significantly reduces energy consumption and enhances the overall efficiency of the network.

Another crucial development in Ethereum 2.0 is the introduction of shard chains, which are smaller chains that interconnect and run parallel to the main Ethereum chain [172]. Shard chains enable Ethereum to process transactions and smart contracts more efficiently, increasing the network's throughput and reducing transaction costs [143]. This upgrade is expected to improve the user experience and drive more widespread adoption of Ethereum-based applications.

Ethereum has also seen the rise of decentralised finance (DeFi) and non-fungible tokens (NFTs) on its platform. On the one hand, DeFi applications leverage Ethereum's smart contract capabilities to recreate traditional financial products without intermediaries, such as loans, insurances, and asset managements. On the other hand, NFTs are unique digital assets

representing various forms of digital art, collectables, or even real-world assets, all secured and traded on the Ethereum blockchain.

Ripple

Ripple, the company, was first founded in 2012 in San Francisco by Chris Larsen and Jed McCaleb. They developed a digital payment protocol that operates both a cryptocurrency (XRP) and a technology protocol (RippleNet) for financial institutions [157]. XRP was created to function both as a cryptocurrency and a technology that could facilitate faster, low-cost international money transfers. Unlike Bitcoin or Ethereum, Ripple does not operate on a traditional blockchain; instead, it uses a consensus protocol to validate account balances and transactions, ensuring rapid processing and reduced energy consumption, but becoming less decentralised [135].

The Ripple payment ecosystem allows for cross-border transactions between any currencies. The XRP digital currency acts as a bridge currency to other currencies. It does not discriminate between real money and cryptocurrencies and thus makes it easy for any currency to be exchanged for another. Each currency in the ecosystem has its own gateway, such as a bank or a liquidity provider. Ripple works in a similar way to SWIFT, a system for messaging and cross-border payments between financial institutions worldwide [134].

In terms of its underlying technology, the Ripple payment protocol, RippleNet, provides a seamless system for global transactions. This system is composed of a network of banks and other financial institutions that accept the Ripple protocol. Ripple's protocol allows the transfer of digital and real currencies, and even commodities, over the network. The consensus algorithm applied by Ripple does not require mining, distinguishing it from Bitcoin and Ethereum. This means that transactions are processed immediately without waiting for blockchain confirmations, as is the case with many other cryptocurrencies.

Ripple and its digital token XRP are important because they attempt to facilitate global commercial transfers more efficiently than the current banking system [78]. The aim is to enable instant, secure and low-cost international money transfers. For banks, this could mean an opportunity to free up liquidity and offer better services to their customers. The broader vision is to enable an "Internet of Value" where any size and form of value can be transmitted as easily as information is on the Internet.

Hyperledger Fabric

Hyperledger Fabric was introduced in 2017 as one of the blockchain projects within Hyperledger. Hyperledger itself is an open source collaborative effort created to advance blockchain technologies between industries. It is hosted by The Linux Foundation, which was established in December 2015 [4]. Hyperledger Fabric, in particular, was initially contributed by IBM and Digital Asset in 2016. The first proposal for a codebase was a combination of IBM's previous work on a protocol known as Open Blockchain and the efforts of the Digital Asset team. By July 2017, Fabric had reached a significant milestone, version 1.0, marking it as a production-ready blockchain platform capable of supporting distributed applications. Since then, it has been continuously evolving and improving.

Hyperledger Fabric is designed as a platform for distributed ledger solutions underpinned by a modular architecture. This architecture delivers a high degree of confidentiality, resilience, flexibility, and scalability, which makes it adaptable to many industry use cases. Fabric has a unique feature of "channels" which allows a network to have many private blockchains, enabling only the participants in a particular channel to see the transactions in that channel. Smart contracts in Fabric, known as chaincodes, contain the business logic of the system that defines the transactions. It is run on the peers and creates transactions. Fabric gives flexibility in deciding which participants need to verify the transaction. This is defined in the endorsement policy. It even allows participants to keep the data private from other participants in the same channel with private data collection. It is a permissioned network, unlike Bitcoin, where anyone can participate, meaning all participants in the network have known identities.

The importance of Hyperledger Fabric lies in its wide industrial applicability. It has seen applications across numerous industries, from healthcare to supply chains, finance, banking, and many more. The ability of Fabric to create private channels and its modular nature enable broad applicability. This technology helps to promote transparency and trust among network participants, a key benefit of blockchain technologies. The permissioned nature of the network provides an additional layer of security as every participant in the network has known identities. Being a part of the Hyperledger project, Fabric benefits from a global community of contributors and the backing of The Linux Foundation. However, like all technologies, it has limitations and may not be the ideal solution for every use case. It is most suited for scenarios where all participants can be known and trust each other to an extent and where a robust system for complex transactions and contracts is required.

Cardano

Cardano is a blockchain platform that was launched in 2017 by Ethereum co-founder Charles Hoskinson through his technology company IOHK (Input Output Hong Kong). The project was named after Gerolamo Cardano, an Italian mathematician and physician. The development of Cardano was notable because it was the first blockchain system to evolve out of a scientific philosophy and a research-first-driven approach. The development team consists of a large global collective of engineering and academic experts who have meticulously designed the cryptocurrency using evidence-based methods based on rigorous and formal academic and scientific research [95].

At its core, Cardano is a platform for running smart contracts, similar to Ethereum, but it differentiates itself through a commitment to high-assurance software and layered architecture. Cardano has a unique two-layered structure: the Cardano Settlement Layer (CSL) and the Cardano Computation Layer (CCL) [91]. The CSL manages the cryptocurrency ADA and handles transactions, while the CCL is responsible for running smart contracts and other computational functions. This layered structure separates the ledger of account values from the reason why the values are moved from one account to the other. This approach improves functionality and security and allows greater flexibility.

In terms of consensus algorithm, Cardano uses a unique proof-of-stake system known as Ouroboros [112]. Ouroboros uses a lottery system to choose how nodes are selected to create

blocks. The chance of being selected scales with the number of tokens held. This makes the Cardano network faster and more energy-efficient compared to traditional proof-of-work systems like Bitcoin.

The importance of Cardano is its focus on scientific philosophy, peer-reviewed work, and formal methods for design that can create a more secure, flexible, and scalable platform. It also emphasises interoperability, sustainability, and regulatory compliance [112].

2.2.2 Smart contracts

Smart contracts are a central part of many blockchain platforms. These contracts allow the automatic execution of agreements once certain criteria are met, which can simplify transaction processes.

Solidity

Solidity is a high-level, statically typed programming language designed to write smart contracts on the Ethereum blockchain. Solidity's syntax is heavily influenced by JavaScript, C++, and Python, making it relatively easy to learn for developers with experience in these languages; however, it is not a Turing complete programming language [105], meaning that it is only programmed to perform a limited set of mathematical calculations. The language allows developers to create and deploy smart contracts that facilitate trustless, decentralised transactions and applications on the Ethereum network (or Ethereum-based networks).

Solidity's core concepts include variables, functions, contracts, data types, and inheritance. Variables store data, while functions define the logic and behaviour of smart contracts[3]. Contracts are the primary building blocks in Solidity, acting as self-contained, reusable units of code that can interact with other contracts and the Ethereum blockchain. Data types in Solidity include value types (e.g., integers, booleans, addresses) and reference types (e.g., arrays, structs, and mappings). Inheritance enables code reusability and modularity by allowing contracts to inherit properties and functions from other contracts.

Over time, Solidity has evolved through various releases, introducing new features and improvements to enhance its functionality and security [88]. In recent years, the Solidity community has focused on optimising the language for greater efficiency and gas cost reduction. This includes enhancements to the Solidity compiler, which converts the high-level Solidity code into low-level bytecode executable by the Ethereum Virtual Machine (EVM). These optimisations help reduce the overall gas costs associated with the deployment and interaction with smart contracts, making it more feasible for developers to create complex dApps and decentralised systems [163].

Once Solidity has been deployed in the Ethereum blockchain, the following key elements are identified:

- **Solidity code**. A contract in Ethereum is a set of codes. This code is not stored in the blockchain [17].

[3] https://docs.soliditylang.org/en/latest/

- **ABI**. An interaction layer in JSON format specifies the communication between contract binary and external applications, detailing functions and data structures [174].
- **EVM**. Once the contract is deployed, Ethereum converts the Solidity code into a low-level programming language that runs on the Ethereum blockchain and is responsible for executing smart contracts [2].

The Solidity ecosystem has grown significantly, with a wide range of development tools, libraries, and frameworks available to support developers in their work. Examples include Truffle, Remix, Hardhat, and OpenZeppelin, which provide testing, debugging, deployment, and reusable smart contract components.

Vyper

Vyper, originally known as Serpent, emerged as an alternative to Solidity. A notable advocate of Vyper is Vitalik Buterin, the co-founder of Ethereum, who has worked on the language's development and promotion.

Vyper has a Python-like syntax, which is aimed to be easy to understand and reduces complexity for smart contract developers. Unlike Solidity, Vyper does away with class inheritance and object-oriented programming, making it harder for developers to write malicious code intentionally. Furthermore, Vyper prevents unintended security vulnerabilities in the code by using the inbuilt libraries [47]. The goal of Vyper is not to be Turing-complete; instead, it aims to make it as difficult as possible to write misleading code. It is designed to explicitly forbid things that are not clear [28].

Vyper offers many properties to provide a secure and straightforward environment for writing Ethereum smart contracts. Its simplicity is a primary attribute, with a deliberate choice to limit features to reduce the likelihood of bugs and security issues. It lacks several programming features, such as class inheritance and function overloading, to avoid confusing or misleading programs. It emphasises security, language, compiler simplicity, and auditability. All Vyper contracts are publicly visible on the blockchain, supporting the notion of complete transparency. Furthermore, the bounds and overflow checking feature in Vyper enables it to prevent overflow and underflow errors [76].

Scilla

Scilla, which stands for Smart Contract Intermediate-Level Language, is a programming language developed for writing smart contracts on the Zilliqa blockchain. It was introduced in 2017 as a part of Zilliqa's efforts to address security vulnerabilities that had plagued smart contracts written in existing languages such as Solidity. The Zilliqa team, led by a group of researchers and academics, aimed to create a language that would enable safer and more secure smart contracts [**?**].

Scilla's design is intended to facilitate formal verification, a mathematical process used to prove the correctness of a program, helping developers prove the correctness of their smart contracts. The key to this is Scilla's structure as an intermediate-level language that is not Turing-complete. It does away with certain programming constructs, such as loops

and recursion, which can lead to complications in smart contracts. Scilla separates the communication aspect (sending and receiving funds) and computational aspect of a contract to avoid reentrancy attacks, a common vulnerability in smart contracts [146].

Scilla incorporates a suite of static analysers that check for potential issues, such as bugs or security vulnerabilities before the smart contract is executed. This feature serves as an extra layer of defence against potential problems in a smart contract. By offering more robust and secure contracts, Scilla aims to build trust and enhance the reliability of executing contracts on the blockchain.

In terms of its main properties, Scilla is built to prioritise safety and security. The language is designed to be simple and straightforward, reducing the likelihood of bugs and security issues. Its design is aimed at providing greater predictability compared to other smart contract languages, which can lead to unexpected behaviours due to their more complex features. However, Scilla's high level of security and safety comes with a trade-off in terms of reduced expressiveness and flexibility, which may limit its ability to facilitate certain types of complex smart contracts.

Other programming languages

While smart contracts programming languages are often associated with blockchain-specific, such as Solidity for Ethereum, some blockchain projects allow the use of general-purpose programming languages to write smart contracts.

- **Java**. Hyperledger Fabric allows the use of Java to develop smart contracts.
- **Go**. Hyperledger Fabric supports Go for chaincode development. In addition, Go is one of the core languages that can be used to develop smart contracts on Chainlink, a decentralised blockchain network built on Ethereum.
- **Python**. NEO is an open source blockchain platform that, in addition to its own programming language called NeoContract, allows developers to write smart contracts in Python.
- **Rust**. Solana, a high-performance blockchain platform, and Polkadot, a multi-chain platform, enable the development of smart contracts using Rust.

2.3 Related work on semantic web and blockchain

Due to the potential of combining the semantic web and the blockchain, a multitude of scientists have studied convergence, challenges, and research trends [149]. In this thesis, the related work is structured into three distinct sections. Initially, the first section (section 2.3.1) introduces the work on combining the semantic web and blockchain on a general level, i.e. methods for combining these two technologies, as well as projects that currently exist for this purpose. Subsequently, the second group (section 2.3.2) addresses work that has been done to store semantic data on the blockchain. Lastly, the third group (section 2.3.3) deals with possible ontologies and knowledge graphs related to blockchain technology.

2.3.1 Theoretical works on semantic web and blockchain

Héctor Ugarte first proposed the idea of combining the semantic web and blockchain[4] in 2016. Ugarte [162] gives the first steps to what the semantic blockchain should look like, giving three definitions of what a semantic blockchain is, describing the process of semantisation of the blockchain (creating ontologies that represent a blockchain and smart contracts, sharing RDF directly from the blockchain and finally, the creation of a semantic blockchain, where the core of the blockchain is based on RDF). In addition, Ugarte proposes that future lines of research should be where to store RDF data (in an external service, in an external blockchain or in the blockchain itself) and the creation of ontologies and triple-store representing blockchains.

English et al. [48] as mentioned before, provided an overview of what the semantic web can do for the blockchain and viceversa, without an in-depth analysis.

The work performed in this thesis [26], explained in detail chapter 4, analyses in depth the benefits that the semantic web can bring to blockchain and vice versa and provides six scenarios where the blockchain and the semantic web can be combined, expanding and concluding Ugarte's work.

Finally, Alsamani and Beckmann [3] surveyed the combination of blockchain and semantic technologies, where they analysed the advantages, disadvantages, and limitations of combining these two technologies and future directions (which are a summary of those proposed by Ugarte), some of these future directions are covered in depth in this thesis (analyse and test how to store RDF in the chain, the creation of ontologies, and triple store development). One of the main sources they used is the work done in this thesis.

In table 2.1 a summary of the work about the theoretical combination of semantic web and blockchain is presented.

Author	Provide advantages	Provide scenarios	Provide guidelines for future work
English et al [48]	Provides an overview of what the semantic web can do for the blockchain and vice-versa, without an in-depth analysis	No	No
Ugarte [162]	Describes the basic properties of blockchain and the semantic web	No	It proposes what the next steps should be
This thesis [26]	Conducts an in-depth analysis of the benefits that one technology can bring to the other and vice-versa	Provides 6 scenarios where the semantic web and the blockchain can be combined	The scenarios provide scope for future work
Alsamani et al [3]	Analyses the advantages provided by other authors and cites part of the contributions of this thesis	No	Yes, provides tips on future lines of research

Table 2.1: Summary of work about the theoretical combination of semantic web and blockchain

[4] Héctor Ugarte's blog

2.3.2 Storing semantic data into blockchain

In the literature, storing into a blockchain the RDF data of a knowledge graph (KG) has been addressed mainly from a theoretical point of view without reporting any quantitative analysis [11, 26, 84, 48, 68, 80, 139, 152, 162]. As far as we know, only three proposals provide a preliminary qualitative analysis [73, 90, 140, 138]. The majority of works describe specific applications that have stored their KGs in a blockchain without analysing the efficiency of this decision over other alternatives [40, 59, 6, 150].

Most approaches address how semantic web and blockchain technologies could work together to enhance their benefits without providing any analysis of their feasibility [26, 84, 48, 80, 162]. Some authors reported a qualitative analysis of how some specific domains could benefit from the use of these two technologies together. For example, for the domains of chemistry [152], smart cities [139], publications [68], or government [11] domains.

To our knowledge, only four articles provided a quantitative analysis of the combination of these two technologies. Ruta et al. [140, 138] performed an analysis of the discovery of Internet of Things resources whose meta-description was stored in a blockchain using RDF; they reported the discovery and query processing time over the RDF involved in this task. However, the results do not have enough granularity to establish only the reading time of the RDF, nor do they provide a comparison with other alternatives.

Le-Tuan et al. [90] presented a scenario of a small network of lightweight nodes. Each node processes 1 billion triples, but those triples are not stored in the blockchain, which instead contains a hash pointing to the online RDF documents. Therefore, although the authors report the time needed to write and query the data, these results do not refer to the blockchain itself. As a result, the cost of writing is neither analysed nor reported.

The only authors that have addressed the matter of the RDF data size are Ibañez et al. [73]. In their paper, they report the number of bytes that different serialisations of RDF have when expressing the same data. In addition, the authors considered the same information compressed with different algorithms. However, RDF was not stored in any blockchain, nor was any cost reported.

In conclusion, currently the literature lacks an analysis of the benefits of storing knowledge graphs inside a blockchain from the point of view of the cost of writing RDF instead of other serialisations, e.g., JSON. Additionally, as far as we know, no work has explored alternatives like materialisation proposals to have the same benefits of RDF when consuming data, but storing JSON in the blockchain.

2.3.3 Ontologies, knowledge graphs, and blockchain

Considering that smart contracts are a programming language, this section presents the work done to represent them through ontologies, first, the programming languages, and second, the ontologies created to represent blockchains and smart contracts.

Ontologies for programming languages

In the software engineering field, there are several ontologies for representing programming languages in the context of distance education [42, 83, 125], code analysis [5] or adaptive learning [92]. Diatta et al. [42] proposed an ontology for exercise solutions in Pascal programming, combining educational and programming language terms. Due to its academic purpose, this ontology focuses on the resolution of assignments rather than the entire language.

For distance education, on the one hand, Kouneli et al. [83] captured the semantics of Java concepts to be used in intelligent e-learning applications, representing the most important notions in the Java programming language. On the other hand, Pierrakeas et al. [125] proposed ontological models for two programming languages, Java and C, in an attempt to provide information about the domain concepts and their relationships to organise the process of design in distance learning.

Lee et al. [92] presented an ontology to facilitate the implementation of adaptive learning in Java, covering most aspects of Java. The work of Atzeni and Atzori [5] included an ontology that provides a formal representation of object-oriented programming languages and a parser that is able to analyse Java source code and serialise it into RDF triples.

Another approach for combining ontologies with programming languages is to generate programming code based on a domain. Goldman [57] used ontologies to generate VisualBasic and C# libraries. Besides, Clark and McCabe [36] extended the work of Goldman with Go.

Ontologies for blockchain and smart contracts

As a relatively new technology, there are few ontologies covering blockchain and smart contracts. In the area of blockchain, the work of Baqa et al. [8] presents an ontology and approach for implementing discovery with semantic web tools in the blockchain. Ethereum has an ontology called EthOn[5]. EthOn is an ontology that represents the Ethereum blockchain, including the modelling of smart contracts; however, this ontology only models the Application Binary Interface (ABI). The ABI is the standard way to interact with contracts in the Ethereum ecosystem, both from outside the blockchain and for contract-to-contract interaction; in other words, the ABI is basically how functions are called in a contract and how data is retrieved. However, the ABI has certain shortcomings, such as the absence of some elements or not covering the function's code. The EthOn ontology contains the function type, the input type (input parameters of the function) and the output type (return type of the function), without representing some aspects of a smart contract such as attributes or inheritance nor going into detail on the aspects named before.

In the area of ontologies representing a smart contract programming language, there is not a proposition yet, but there are works that combine the use of ontologies and smart contract technologies. The main focus of these works is to provide discovery, generation, validation or implementation mechanisms [32, 85, 81, 8, 46, 115]. As a result, they do not allow modelling smart contract languages. Norta et al. [115] develop an ontology with the principal concepts and properties of smart contracting for the management of decentralised

[5] https://github.com/ConsenSys/EthOn

autonomous organisations, which can be used in blockchain. Dwivedi et al. [46] specify a smart-legal-contract markup language for the creation of legally binding commercial and legal contracts. To achieve this, the authors first design and verify an ontology that describes the legal and business semantics; then, they translate the ontology to a markup language. After that, they demonstrate and evaluate the translation through the specification of other contracts and finally translate the markup language into Solidity to demonstrate its feasibility.

Choudhury et al. [32] propose a method to generate smart contracts in the Go language for Hyperledger using the OWL language, using the Semantic Web Rule Language to expand the expressiveness of OWL and abstract syntax trees to generate the code of the smart contract given a template. However, this article focuses only on the Go language and has no general ontology that can create any kind of contract; instead, for each domain, a specific ontology model has to be designed and created in order to generate the contract. Kruijff and Weigand [85] designed an ontology to represent the interaction of smart contracts in the blockchain. Kim and Laskowski [81] design an ontology for smart contract interaction, tracing the origin and destination of a smart contract.

Author	**Domain**	**Blockchain Technology used**	**Smart Contract treatment level**
Baqa et al. [8]	Discover smart contracts deployed on the blockchain	Ethereum	ABI
Choudhury et al. [32]	Code generation	Go	Domain-level
Dwivedi et al. [46]	Knowledge representation	Solidity	Domain-level
Kruijff and Weigand [85]	Interaction with blockchain	Undefined	Smart contract blockchain interaction
Kim and Laskowski [81]	Interaction with smart contracts	Solidity	ABI
Norta et al. [115]	Knowledge representation	Undefined	Domain-level

Table 2.2: Summary of the main differences in related work about ontologies and smart contracts

Considering the related work, the treatment of smart contracts can be classified into different levels:

- **Programming language**. Authors cover smart contracts by formal definition in high-level code.
- **ABI**. Covers smart contracts by using a list of contract functions and arguments.
- **Smart contract - blockchain interaction**. Smart contracts are covered, focusing on the interaction between blockchain and smart contracts.
- **Domain-level**. The authors use a specific domain that is not related to blockchain and smart contracts; useful mechanisms are developed for these.

Several ontologies combined with programming languages or smart contracts have been proposed, but no ontology has been carried out that represents a respective smart contract language, as shown in table 2.2. In chapter 6 we develop an ontology to represent the most important smart contract language, Solidity.

There are three proposals for models to represent smart contracts: using UML [126, 75], a variation of UML [153], and using OWL [27]. Although two ontologies related to smart

contracts exist, up to the authors' knowledge, there is no proposal that provides smart contracts as RDF triples.

Pierro [126] proposes a graphical representation model to visualise smart contract source code in UML. However, the proposed model only displays certain elements without representing features, such as the version of Solidity used in the contract, the code of the functions, or modifiers.

Jurgelaitis et al. [75] use a UML model to generate Solidity smart contracts. The proposed UML model represents the majority of the Solidity language and the behaviour logic of the contract.

Skotnica and Pergl [153] propose a domain-specific language that represents smart contracts in a user-friendly way. Nevertheless, this article presents the smart contract in a domain-specific way using DEMOSL, a variation of UML.

Note that UML, or a domain-specific language based on UML, does not allow querying any of the represented elements. Instead, this feature is enabled in the ontologies since their models are defined using RDF, and therefore, are queryable by means of SPARQL.

The EthOn ontology[6] models contextual data belonging to the context of Solidity and Ethereum. EthOn uses OWL as a language model, and this model allows querying and modelling context data; however, EthOn is focused on modelling Ethereum as a State Transition System.

EthOn	**Model language**	**Models smart contracts**	**Is model queryable?**	**Allows modelling context data?**
EthOn	OWL	Solidity ABI	Yes	Yes
Skotnica and Pergl [153]	DEMOSL	Solidity code	No	No
Pierro [126]	UML	Solidity code	No	No
Jurgelaitis et al [75]	UML	Solidity code	No	No

Table 2.3: Proposals modelling Solidity smart contracts.

Chapter 7 discusses how to use the Ethereum, ABI, and Solidity ontology developed in this thesis to create a knowledge graph. This graph will include blocks and contracts that have already been deployed and validated on the Ethereum blockchain. The generated knowledge graph provides a representation of an Ethereum semantic blockchain with the benefits of the semantic web, such as linked data, validation, data and meta-data validation, search over smart contracts, interoperability, code generation, and representation in multiple formats, inter alia.

2.3.4 Conclusions

The combination of the semantic web and blockchain is a recent topic. The lack of common standards and protocols is a barrier to the widespread adoption of the semantic web in

[6] https://ethon.consensys.net/index.html

blockchain technology. Although there are ontologies proposed to represent blockchain metadata and data in RDF format, no standard ontology has been established yet. This limits interoperability and hinders the implementation of efficient queries and data analysis across different blockchain platforms.

In addition, technical limitations also pose challenges. For example, RDF formats, such as N-Triples, are very verbose and require a large number of characters, which can increase the size of blocks on the blockchain and decrease the efficiency of the system. In this sense, more research and development is needed to find effective and efficient ways to store RDF data on the blockchain.

The first step of this thesis is to analyse the benefits that the semantic web can provide to the blockchain and vice-versa and to study under what conditions these two technologies can be combined. The second step is to determine the most effective and efficient methods for storing RDF data on the blockchain.

The third step is the creation of ontologies and triple stores to represent the blockchain. Developing such a model would further enhance the compatibility of blockchain technology with the semantic web and offer new ways to query and navigate blockchain data.

The final step is to create a representation of a semantic blockchain with all the tools and ontologies developed in the thesis. The process of achieving the representation of a semantic blockchain can be summarised as depicted in figure 2.3.

In fact, Ugarte [162] offers different perspectives on how a semantic blockchain could be created, and the steps that he proposed are aligned with this thesis. However, Ugarte aims to reach the semantic blockchain, where the core of the blockchain is based on RDF.

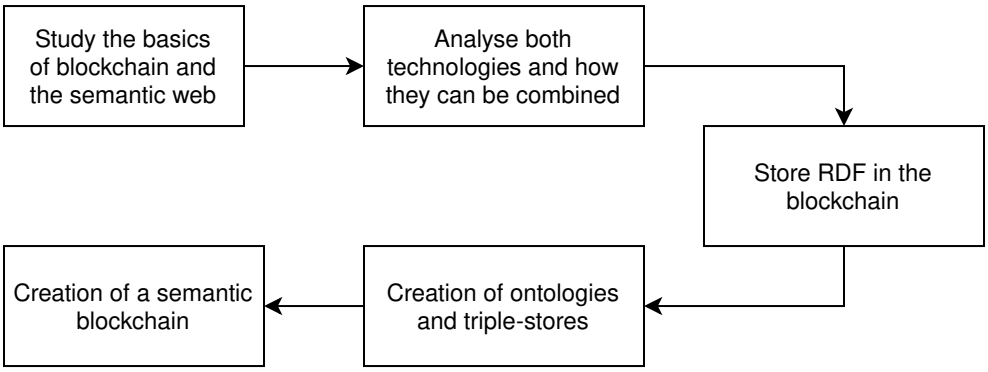

Figure 2.3: Creating a semantic blockchain representation from scratch

In this thesis, Ethereum is the blockchain technology used to experiment with the semantic web. Ethereum is a blockchain platform that offers a suitable environment to explore the integration of blockchain technology with the semantic web, mainly due to its unique properties and features. Ethereum introduced the concept of smart contracts to the blockchain world. Smart contracts are programs that automatically execute when certain predefined conditions are met. Ethereum facilitates these smart contracts that could aid in the integration with the semantic web. Smart contracts can be used to create automatic rules and protocols that interact with semantic data, which could automate a wide range of processes in numerous

industries, from finance to healthcare. The choice of the Solidity programming language for development on Ethereum is worth noting since Solidity is the primary programming language for writing smart contracts in Ethereum.

Ethereum is an open platform and thus, is another valuable feature for exploring the merger of blockchain and the semantic web. Being an open platform, Ethereum allows anyone to create and publish their own decentralised applications. This offers great flexibility to experiment with different methods of integrating the semantic web with blockchain.

In addition, Ethereum is one of the most widely adopted blockchain platforms, implying that any experimentation or development done on Ethereum could have a significant impact and could be quickly adopted by a wide range of users and developers. This widespread adoption could help drive semantic web adoption as well, by demonstrating practical and beneficial cases that could be achieved by combining these technologies.

In conclusion, Ethereum supports smart contracts and is an open platform. The widespread adoption, future enhancements in Ethereum 2.0, and the use of Solidity as a programming language provide compelling reasons to choose it as the platform to explore the combination of blockchain technology and the semantic web.

Chapter 3

Objectives and contributions

> The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them.
>
> *Sir William Lawrence Bragg*

As discussed in the previous chapters, this thesis is focused on analysing the benefits of combining the blockchain and the semantic web, planning optimal methods to store RDF into the blockchain, building ontologies to represent the main functionalities of the blockchain, and exploring the combination of the blockchain and the semantic web. Additionally, in order to evaluate the results, a set of methods and experiments are designed. This chapter presents the objectives of our work and identifies our contributions to the current state of the art. This chapter enumerates assumptions considered at the beginning of this thesis and describes the most relevant hypotheses and restrictions that delimit the scope of this thesis.

The main research question that supports this thesis is: **How does the combination of blockchain technology and the semantic web affect the storage, access and verification of information in decentralised systems?** This can be subdivided into a series of sub-questions:

RQ1. In which practical cases can blockchain technology be combined with the semantic web?

RQ2. What are the costs and performance of writing and reading data from the blockchain when the data are stored in a standard semantic format?

RQ3. What are the advantages and disadvantages of storing blocks and contracts in a triple store?

3.1 Objectives

The main objective of this thesis is to **explore and develop methods for the integration of the semantic web with blockchain, with a specific focus on the Ethereum**

blockchain, to achieve a representation of the semantic blockchain, and it is composed of the following four sub-objectives:

O1. Detail the benefits of combining blockchain with the semantic web and identify potential scenarios that can be used to combine blockchain with the semantic web.

O2. Determine optimal strategies to introduce RDF into the blockchain, evaluating them in terms of efficiency, cost, and scalability.

O3. Develop new ontologies to represent the Ethereum blockchain that enable an accurate representation of the Ethereum blockchain structure.

O4. Design and implement a method that allows the representation of the Ethereum blockchain in a knowledge graph, including block structure, ABI contracts, and smart contracts in Solidity.

In order to achieve these objectives, the following limitations have been identified in the state of the art that need to be addressed:

L1. The state of the art lacks an in-depth analysis of the benefits and scenarios of combining the two technologies.

L2. The state of the art presents studies that store semantic data in external sites and the blockchain serves to ensure the integrity of those data, but there are no studies that analyse data storage on the blockchain itself.

L3. There is a lack of ontologies developed specifically to effectively represent the concepts and structures of the Ethereum blockchain, in particular, and of the blockchain in general.

L4. There are technical challenges to integrate the Ethereum blockchain into knowledge graphs, such as mapping Ethereum data, transactions, and ABI and Solidity smart contracts within the blockchain to a semantic model.

3.2 Contributions

In this work, our aim is to provide solutions to the objectives described in the previous section.

C1. Analysis of the benefits and limitations of integrating the semantic web with blockchain technology compared to other technologies or methods. In addition, we present scenarios that demonstrate how the combination of semantic web and blockchain can be applied in different applications, studying the advantages and disadvantages of each scenario. This contribution is presented in chapter 4.

C2. Analysis of the feasibility of storing semantic data into the blockchain, addressing challenges such as efficiency, scalability, and cost, first by integrating such data directly into blocks through regular transactions and second by doing it through smart contracts. This contribution is presented in chapter 5.

C3. A set of ontologies to represent the Ethereum-based blockchain and Solidity

smart contracts. This contribution is presented in chapter 6.

C4. A method for generating a knowledge graph that provides a representation of the Ethereum blockchain according to the ontologies developed. This contribution is presented in chapter 7.

3.3 Assumptions

Our work is based on the assumptions listed below. These assumptions provide a background to facilitate the comprehension of the decisions taken during the development of this thesis.

A1. It is assumed that blockchain technology is used to combine the semantic web and the blockchain as a public blockchain technology. This assumption is made on the basis that both technologies have a universally decentralised platform approach that allows cooperation between their users.

A2. The effectiveness of the integration between blockchain and the semantic web can be evaluated by considering the gas costs and efficiency of the read and write operations, taking into account the time required for data storage and retrieval. It is also assumed that Solidity is the most suitable programming language for the implementation of smart contracts on the Ethereum blockchain due to its wide adoption, compatibility with other blockchain technologies and support. In addition, Solidity is recognised as the standard language for smart contract development on Ethereum.

A3. It is assumed that the ontologies developed will be applicable to a wide range of applications and use cases on the Ethereum blockchain.

A4. It is assumed that the data contained in the Ethereum blockchain are sufficiently rich and accessible to generate a meaningful and useful knowledge graph.

3.4 Research hypotheses

After identifying the assumptions, we can describe the research hypotheses of this thesis. These research hypotheses cover the general characteristics of the contributions.

H1. The proposed combination of blockchain technology and semantic web (C1) is beneficial to the semantic web in terms of security and decentralisation; it also increases interoperability between different blockchain technologies.

H2. It is possible to store and manage complex and semantically rich data in a decentralised way (C2) without incurring prohibitive costs.

H3. The proposed ontologies (C3) will facilitate the integration of semantic web and blockchain technology, improve interoperability, and enable more accurate queries of blockchain data.

H4. The proposed knowledge graph (C4) significantly improves the way data stored on the blockchain are interacted with and extracted, facilitating the exploration and analysis

of the Ethereum blockchain data.

3.5 Restrictions

Finally, a set of restrictions describes the limitations and defines the future work objectives. The restrictions are as follows:

R1. The proposed synergies rely heavily on public blockchain technology. For example, a public blockchain offers a level of decentralisation that a private blockchain cannot.

R2. As a recent technology, due to the constant evolution of blockchain, more approaches could emerge in which blockchain and semantic web can be combined.

R3. The work in this thesis focusses on storing RDF on the blockchain and filling current gaps, but it only tests a few serialisations and does not take advantage of the full potential that semantic web tools have.

R4. The work in this thesis only covers the Ethereum blockchain, so the necessary ontologies have been built only for this blockchain technology.

R5. The work in this thesis covers only the Solidity smart contract, so the necessary ontologies have been built only for this programming language.

R6. Only a sample of the Ethereum blockchain has been represented in a knowledge graph without exploring other blockchains, whether public or private. This stored information would also consume a large amount of storage space if the entire blockchain was represented.

R7. The construction and analysis of the knowledge graph are heavily based on third-party tools, because Ethereum does not store the code of smart contracts and must be collected externally.

Figure 3.1 shows the mapping between the objectives, the contributions, and the assumptions, hypotheses and restrictions.

3.6 Experimental methodology

Regarding the hypotheses defined in this thesis, the following experiments have been performed to validate them:

E1. (for H1.) The goal of this evaluation is to research both technologies individually and jointly, to explore synergies between them and to provide hypothetical integrations between blockchain and the semantic web.

E2. (for H2.) The goal of this evaluation is to analyse two storage prototypes: one that embeds semantic data directly into blocks through regular transactions and one that uses smart contracts for this purpose. A series of experiments will be performed for each prototype, followed by a comparative analysis between them. This approach will determine the most efficient and cost-effective method for integrating semantic data

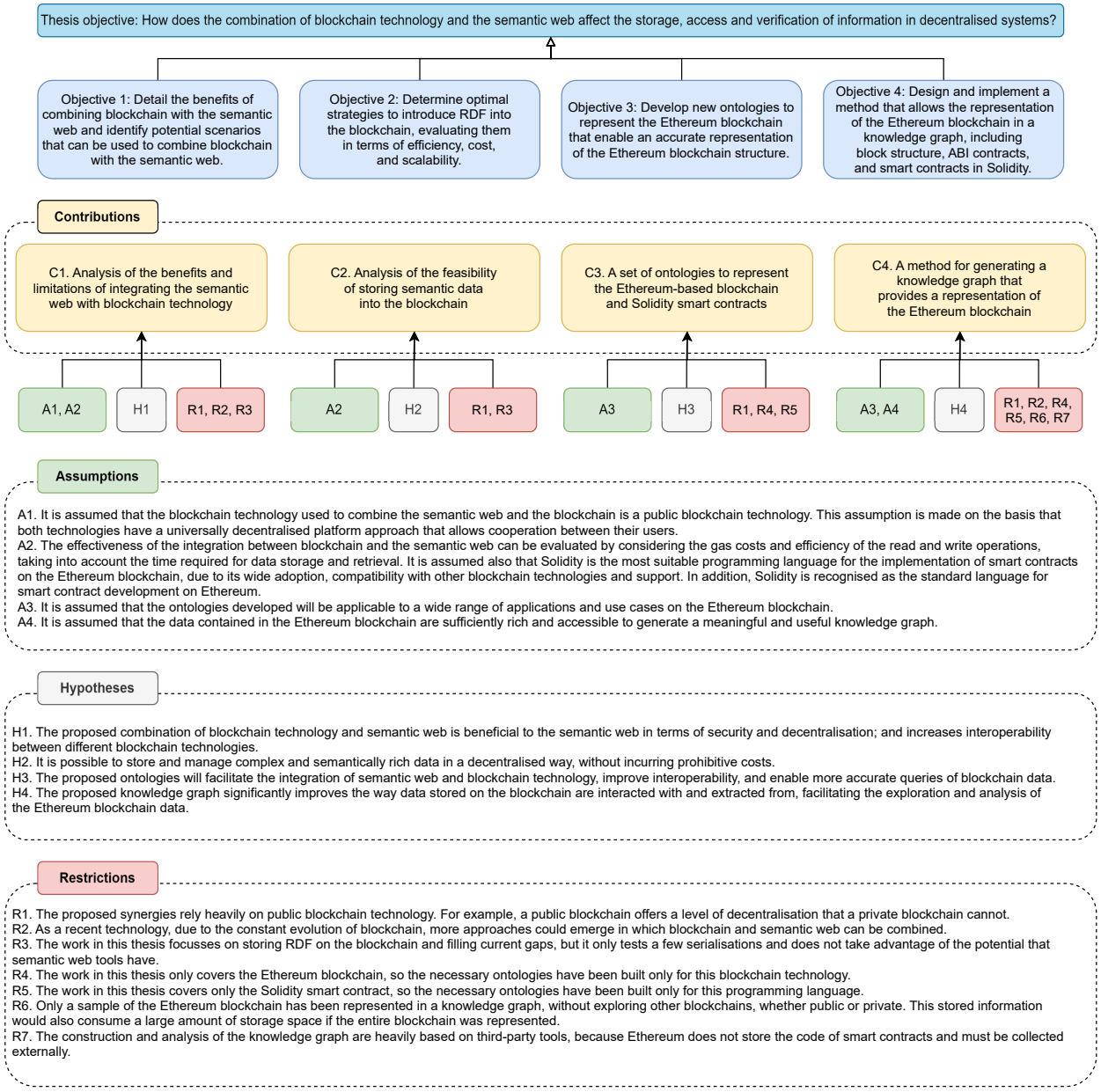

Figure 3.1: Mapping between the objectives, contributions, assumptions, hypotheses, and restrictions.

into the blockchain.

E3. (for H3.) The goal of this evaluation is to design and build ontologies that reflect the structures of the Ethereum blockchain and Solidity smart contracts. The ontologies developed will be validated through different methods and use cases, facilitating the integration of the semantic web with blockchain technology and improving the interoperability and queryability of blockchain data.

E4. (for H4.) The goal of this evaluation is to determine the effectiveness of the knowledge graph in improving interaction and data mining on the blockchain.

3.7 Research methodology

This section presents an overview of the research methodology followed during the development of this thesis. The research methodology outlines the methodological and technical inputs that were used to achieve the objectives of this thesis and to provide several contributions to open research problems. These contributions are then combined to achieve each specified goal.

In order to deliver the contributions of this thesis, both methodological and technological inputs are considered, as shown in Figure 3.2. Additionally, community feedback has also been taken into account to validate and guide this thesis. Figure 3.2 also shows that the work is framed by a set of objectives, restrictions and assumptions.

1. Methodological guidelines for performing literature reviews. Part of the methodology followed in the thesis relies on the literature reviews performed in the fields of interest. For performing such literature reviews, methodological guidelines by Kitchenham [82] have been followed.
2. State of the art analysis in combination between blockchain and semantic web. This analysis helps to identify the most suitable method for storing semantic data in the blockchain.
3. Existing ontologies. New models were designed and proposed to obtain an overview of the main properties and characteristics of the Ethereum blockchain and Solidity smart contracts.
4. Software libraries. For developing software to support the Sancus software proposed in this thesis, existing software libraries such as Jena and Spring were used.
5. Community feedback. In the development of this goal, feedback from ontology experts and researchers of the semantic technology recommendations was taken into account.

In the development of this thesis, an incremental process has been followed, leading to four different phases: initialisation, development, implementation, and refinement. Figure 3.3 shows an overview of the main contributions of each phase, including (A) the order in which the phases were carried out, (b) the task carried out during each phase, and (c) the publications obtained from each phase. Figure 3.3 intends to be self-contained, and therefore, the citations represented by integers between brackets correspond to the enumerated bibliographic resources at the bottom of the figure. Next, the four phases of development are described:

- **Initialisation.** During this phase, the state of the art of blockchain and semantic web was analysed. As a result, the initial objectives of this thesis were defined (related to C1).
- **Development.** During this phase, the initial version of the testing approaches to combine the semantic web and blockchain was evaluated (related to C2). As a result, a set of ontologies related to the different components of Ethereum was developed (related to C3).
- **Implementation.** During this phase, existing software libraries were used to provide

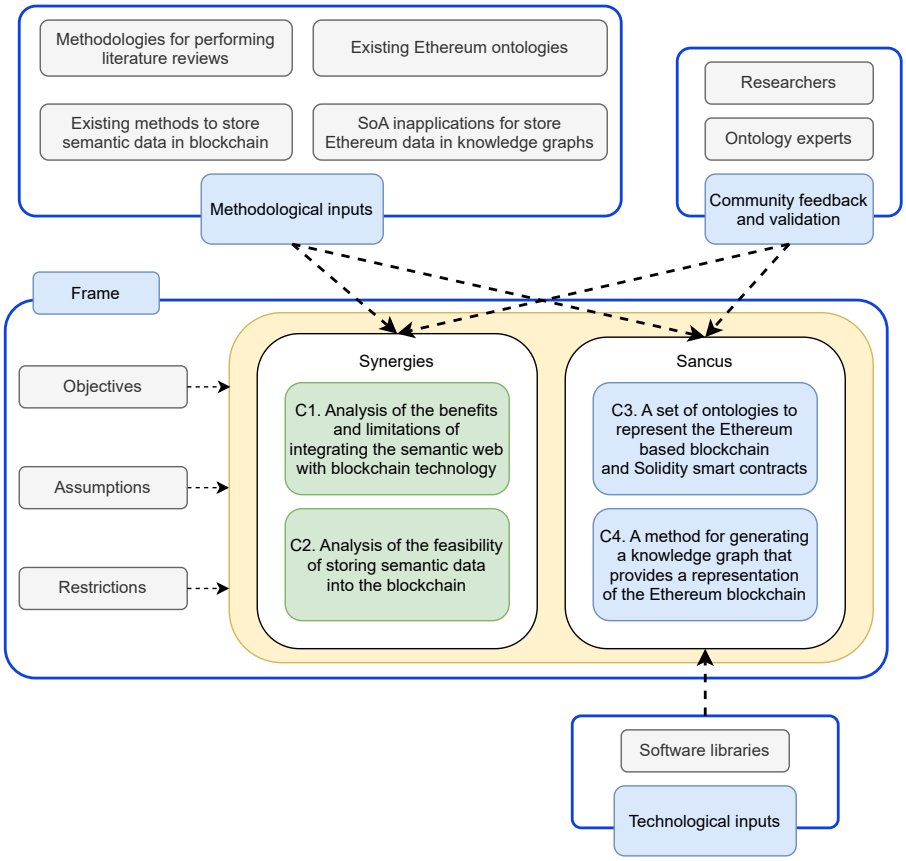

Figure 3.2: Inputs taken into account for delivering thesis' contributions.

the technological contribution of this thesis, Sancus, the tool created to build knowledge graphs with Ethereum blockchain data (related to C4). At the same time, the ontologies were extended to fulfil additional requirements to represent more elements in Sancus (related to C3).

- **Refinement.** During this phase, continuous refinements and improvements were performed over the main contributions of this thesis.

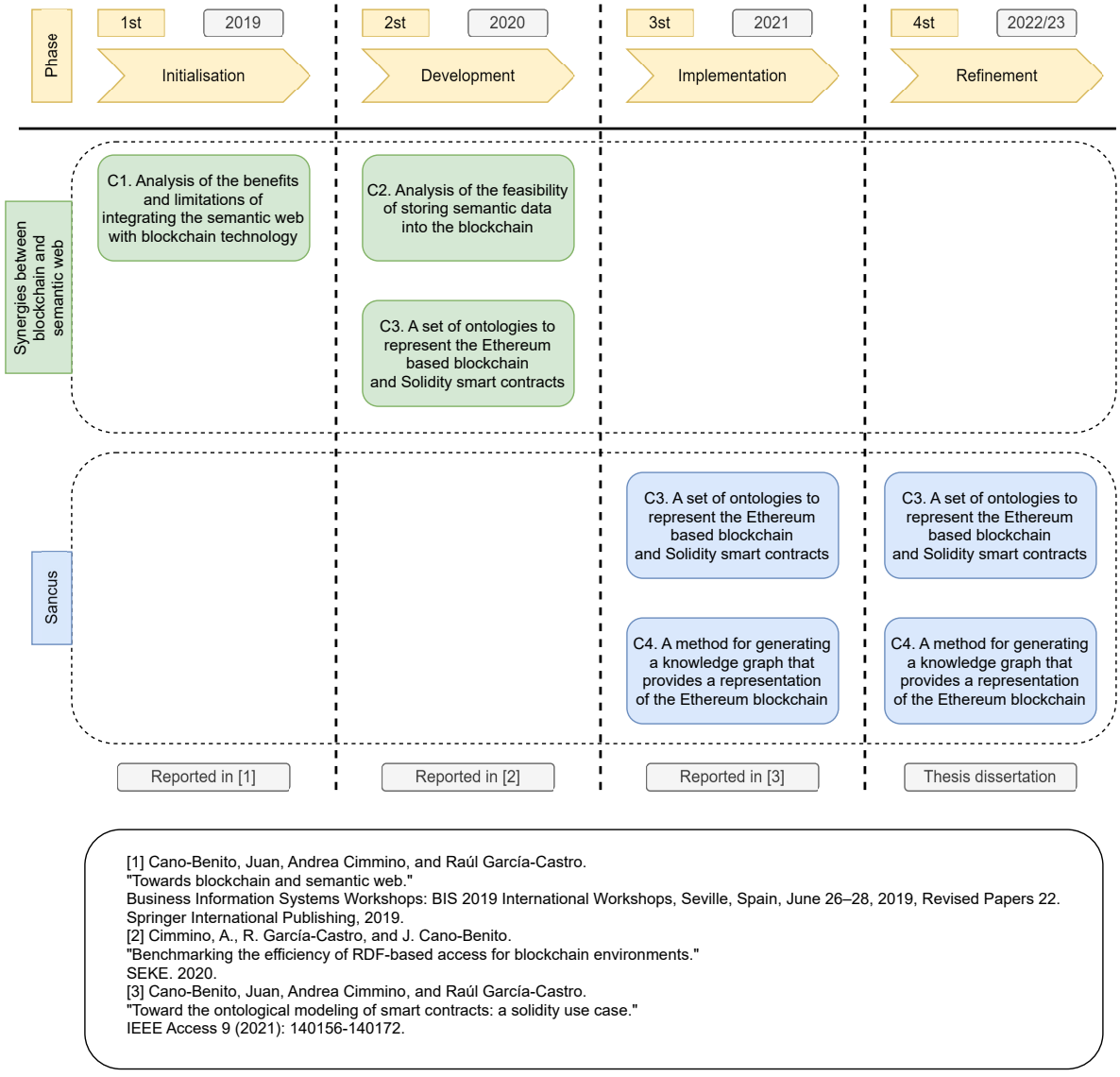

Figure 3.3: Phases of the thesis development

Chapter 4

Synergies between blockchain and the semantic web

Unity makes strength

Motto originally used by the Dutch Republic

4.1 Introduction

This chapter presents how the combination of the semantic web and blockchain can provide a richer and more efficient environment for decentralised information access and management. To this end, we extend the work of English et al. [48] and Ugarte [162] with an overview of the different scenarios and approaches to combine blockchain with the semantic web by analysing the advantages and disadvantages of the different scenarios.

Section 4.2 details the benefits of combining blockchain and the semantic web, and viceversa; and section 4.3 introduces the different scenarios that we have identified to combine blockchain and semantic web. Finally, section 4.4 concludes with the recap and conclusions of this chapter.

4.2 Benefits combining blockchain and the semantic web

The work presented by English et al. [48] introduced namely one benefit that semantic web offers to the blockchain, i.e., by using ontologies to model the metadata of the blocks practitioners may perform queries using SPARQL. Unfortunately, such an approach does not query the content of the blocks since their data was not expressed in RDF, as pointed out by the authors. On the other hand, the benefit that the semantic web may find by using blockchain, according to English et al., is the following one: IRIs to identify resources in the semantic web depend on a DNS, whereas by using blockchain technologies, the identification of resources can be relayed on the hashes of a blockchain that will point to their related

properties, achieving a decentralisation of the domain name system (DNS). However, after analysing both technologies, we have reached a richer and more detailed list of benefits, which are presented next.

Blockchain benefits when combined with the semantic web

Considering semantic web technologies, the blockchain may find the following benefits:

- **One language multiple formats:** RDF is not bound to a specific format; therefore, the data described following this standard can be expressed in multiple formats, e.g., RDF/XML, JSON-LD, Turtle. Blockchain may rely on RDF to write the content of the blocks, using the format that better suits a specific domain problem.
- **Model data following well-known standards:** semantic web technologies like RDF, SPARQL, RDFS, or OWL are well-known W3C standards. This entails that there is a global consensus which makes them reliable and trustworthy. In addition, there is a large amount of standard ontologies ready-to-use to describe the data of multiple domains easing the modelling task for a given problem.
- **Linking of data:** A key feature of RDF is its ability to link to data from other datasets. This facilitates a unified global view of the data, despite the storage of various segments being distributed across the Web. Linking data is a well-known and largely addressed challenge in the semantic web community [111]. Blockchain may benefit from this feature by linking the content of the blocks with external datasets or even with the content of other blocks in the same or in a different chain.
- **Multiple data models:** ontologies count with alignments to relate their properties and classes to other ontologies. This feature means that when a blockchain relies on an ontology that has alignments with another ontology, the data described with the former ontology can be translated to the model of the latter automatically. This is especially suitable when data must be modelled differently at the same time, depending on who is consuming such data.
- **Search over blockchain:** assuming the blockchain relies on ontologies and RDF to describe their metadata and their content, practitioners may use SPARQL to query the chain. However, this capability requires a third party service that reads the blockchain and executes the queries.
- **Blockchain data and metadata validation:** once a chain relies on ontologies and RDF, it may benefit from logical restrictions to validate its data and metadata. The restrictions report the errors in the model and in the content of a specific RDF document.
- **Blockchain consistency validation:** the semantic web offers reasoning engines for checking the consistency of the data. In this case, the blockchain should communicate with a third-party service to perform the reasoning and have its metadata expressed relying on an ontology.
- **Data inference:** data expressed in RDF is not always stored. Instead, such data are sometimes generated on the fly. This generated RDF is known as virtual RDF. One

approach consists in using reasoning engines, which infer new data. An alternative approach is to infer RDF relying on data shapes. Finally, another approach relies on graph embedding that analyses current data and creates new knowledge based on machine learning proposals [94].

- **Linked data virtualisation:** semantic web has engines that rely on specifications that are able to translate on-the-fly data from heterogeneous datasources into RDF [93, 43]. A blockchain may benefit from this kind of engine, known as virtual RDF services, by storing data in non-RDF formats and relying on these third-party services to generate at the same time their content in RDF.
- **Interoperability:** relying on semantic web technologies, and using standard ontologies to model data, both metadata and content, the blockchain becomes interoperable. Being interoperable means that an information system is able to transparently interact with other interoperable systems, e.g., other blockchains, databases, or services. In addition, third-party systems can discover interoperable systems and know how to access their data automatically.

Semantic web benefits when combined with blockchain

The benefits that the semantic web may find by using blockchain are:

- **Data decentralisation:** on the one hand, the semantic web aims to store data using a decentralised approach; on the other hand, blockchain is a decentralised data store. Therefore, the semantic web may benefit from the decentralised nature of the blockchain in order to store data. In addition, the fact that the chain is shared by several peers increases the availability of data, which normally depends only on one service.
- **Identifiers for RDF resources:** one of the well-known benefits that blockchain brings to the semantic web is the generation of identifiers that do not depend on a DNS [48], e.g., Ethereum Name Service (ENS).
- **Data immutability:** the semantic web has been adopted by several public entities to provide open, accessible, and transparent data, e.g., open government initiatives [151]. These scenarios require that the data published and its provenance can be trusted. One of the well-known properties of blockchain is that once data are published, it cannot be modified. This makes blockchain the perfect technology to be used in these scenarios.
- **Data transparency and privacy:** since blockchain supports different reading and writing permissions, the data stored is immutable, and information is shared and decentralised. Thus, applications of the semantic web that aim to promote transparency with privacy policies find blockchain the perfect technology to rely on. A clear scenario that may benefit from this is clinical data, in which the data is sensitive and must follow strict privacy policies [116].
- **Crowdsourcing data:** a large number of semantic web applications rely on the participation of external entities, humans or machines. Public blockchains bring the perfect technology for this kind of application, since external entities will be able to publish data and at the same time, provenance, trust, and transparency will be

guaranteed.

4.3 Semantic web and blockchain scenarios

There is only one article that addresses some preliminary ideas about how to combine these technologies, i.e., Ugarte [162], who presented how the semantic web and blockchain could be combined in 5 scenarios. Starting from this article and analysing the state of the art, we have identified a total of six different scenarios, which we explain in the following sub-sections.

4.3.1 Scenario 1. Blockchain with semantic metadata

This scenario, depicted in Figure 4.1, features a blockchain where each block contains data and is linked to previous and subsequent blocks. However, this scenario consists of a blockchain in which the metadata is expressed in the RDF format following an ontology and the content of the blocks is expressed in a non-RDF format (other data that is not RDF). Some ontologies have been proposed to represent the metadata of some blockchains, such as Ethereum [124]. However, there is no standard ontology yet. In this scenario, it is the blockchain itself that must return the metadata in RDF format, as this metadata is generated when the block is created, and therefore, the user cannot introduce the information.

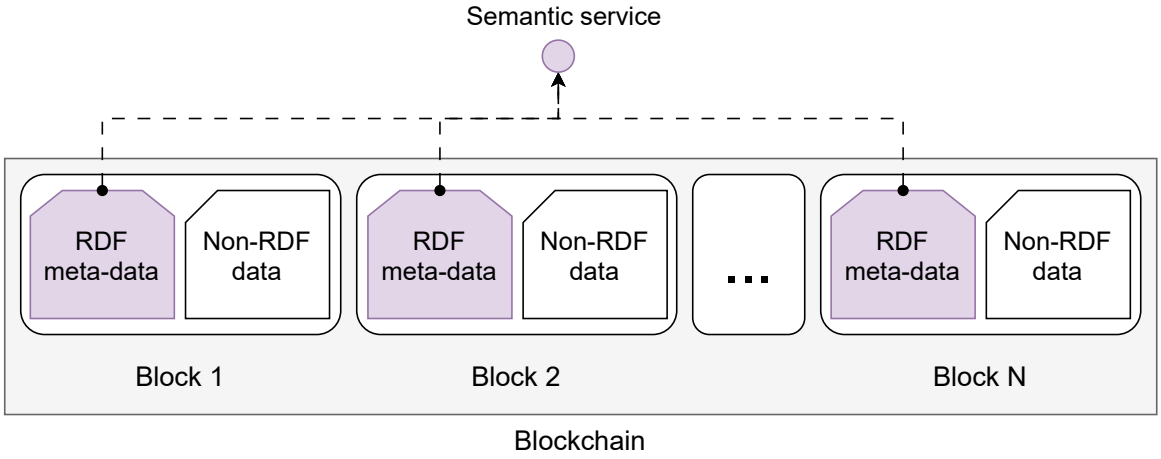

Figure 4.1: Blockchain with semantic metadata

The main benefit of this scenario is that practitioners may perform search queries considering the metadata of the blocks. However, this benefit requires to have an external service that reads the blockchain and is able to process SPARQL queries. For instance, with this benefit, a user could search all the blocks whose hash follows a provided regular expression. Alternatively, APIs, like Apache Jena or RDFLib, can be used to read and manipulate RDF metadata in a programmatic way. Also, one can publish these RDF metadata on the Web, making the data accessible and linkable across the Internet.

This scenario is proposed because in traditional blockchains, searching and analysing the metadata can be difficult due to the structure of the blocks and the lack of a standard data schema. By expressing metadata in the RDF format, this data becomes much more

accessible and interoperable. This is especially useful for performing complex searches and analysis on blockchain metadata. The use of ontologies provides structure and context to the metadata, making it easier to interpret and analyse. It can also help standardise metadata across different blockchains, which in turn facilitates interoperability between different blockchain-based systems.

The main drawback of this approach is that only the metadata of the blocks is expressed using semantic web technologies. Therefore, this scenario allows reasoning, linking, or executing SPARQL queries only over the metadata, inter alia.

4.3.2 Scenario 2. Blockchain with semantic data

In this scenario, the approach to combine semantic web and blockchain relies on storing data in the blocks using RDF and following an ontology, as shown in Figure 4.2. This scenario is complementary to the previous one. The content of the blocks may be expressed in any RDF serialisation, e.g., JSON-LD or Turtle.

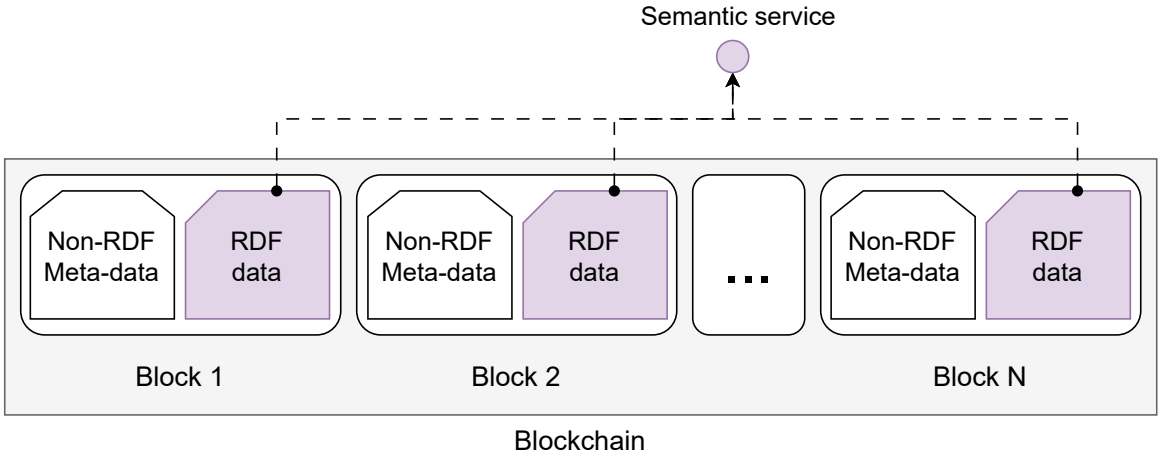

Figure 4.2: Blockchain with semantic data

Assuming there is an external service that reads the blockchain and is able to process RDF (e.g., making SPARQL queries), in this scenario, the blockchain obtains all the benefits previously analysed. On the one hand, practitioners can execute SPARQL queries over the content of the blocks (and their metadata if scenario 1 is also supported), which are described using an ontology. On the other hand, links to other data sources or ontologies could be stored in the chain.

The main motivation of this scenario is to improve the interoperability, accessibility, and searchability of data on the blockchain. By having the data in the RDF format, it can be queried and linked more easily, allowing for a more complex analysis. There are three ways to store RDF data in the blockchain, the first explores introducing RDF directly into the block, the second explores introducing RDF data using smart contracts, and the last one is a scenario where smart contracts can read and query RDF data.

Scenario 2.1. Writing semantic data into the block directly

As depicted in Figure 4.3, the RDF data is stored in the block directly. The main drawback of this scenario is that some RDF formats, such as N-Triples, are very verbose and require a large number of characters. As a result, depending on the serialisation, using RDF entails that the chain may contain larger blocks to express the same information that could be expressed with a non-RDF format and, at the same time, a larger number of blocks. Having a large number of blocks that contain a large amount of data may decrease the efficiency of the blockchain.

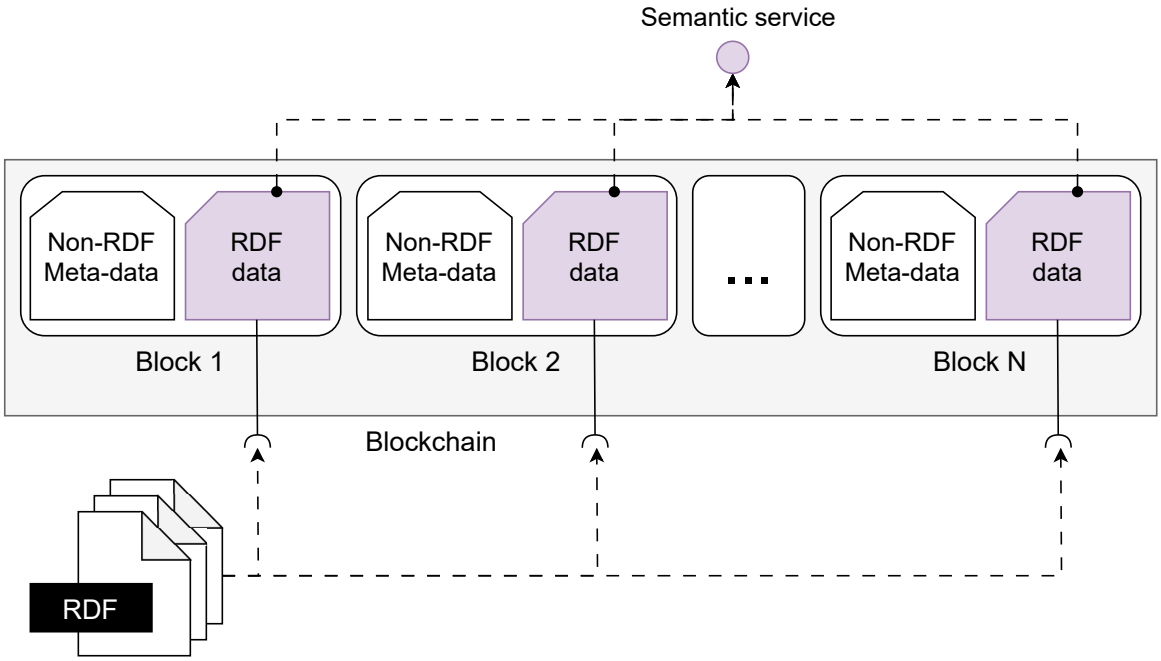

Figure 4.3: Blockchain with semantic data into the block directly

Scenario 2.2. Writing and querying semantic data into the block using smart contracts that implement semantic services

As depicted in Figure 4.4, the smart contracts store data in the blockchain. In some blockchain technologies, the information of the smart contracts is written in a general-purpose programming language, such as Hyperledger fabric [52], then it is possible to store JSON-LD in the blockchain and embed a triplestore in the smart contract, making it possible to store data and execute semantic queries from the contract itself, without the need of third-party services. Due to the nature of this scenario, new features like decentralised knowledge graphs can be implemented.

4.3.3 Scenario 3. Blockchain with virtual semantic data and metadata

This scenario consists in a blockchain and a virtual RDF service. Virtualisation services take as input a data source, i.e., the blockchain, and generate RDF as depicted by Figure 4.5.

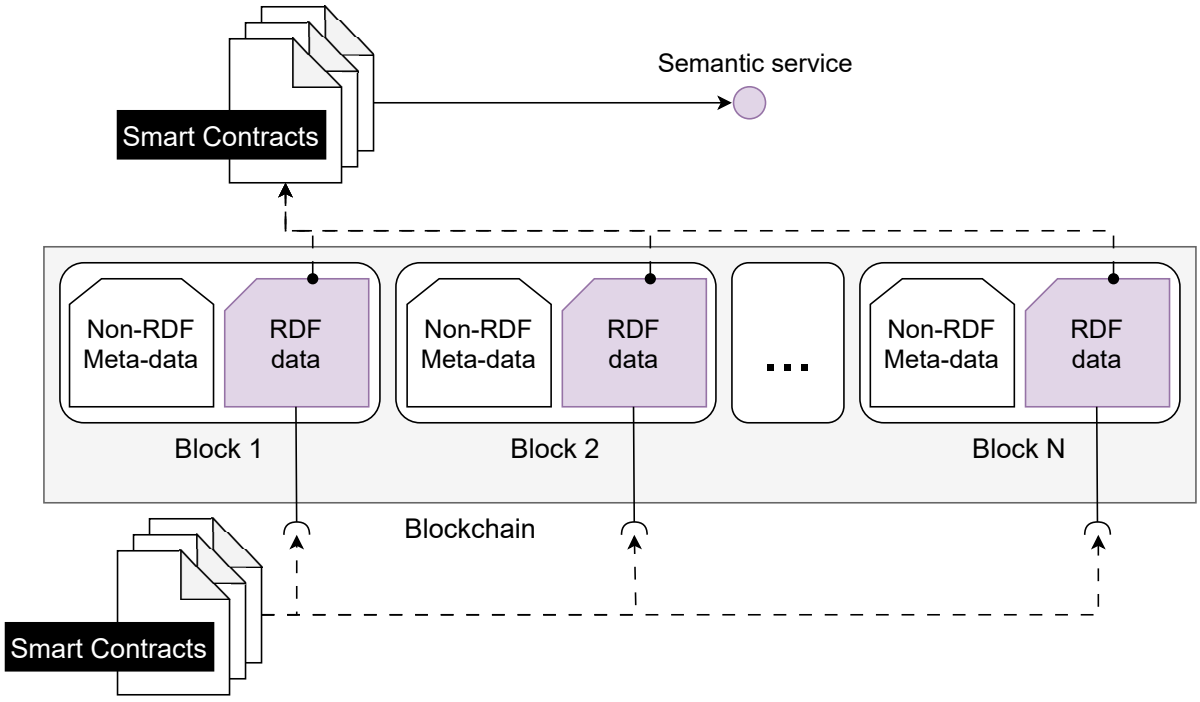

Figure 4.4: Blockchain with semantic data into the block using semantic smart contracts

Some services publish the data as a dataset and count with a SPARQL query endpoint; others only generate an RDF dump that must be stored in a triple store in order to query the data.

The main difference with scenario 4.3.2 is that in this scenario, the entire blockchain is represented by an external service in RDF, not just the data. In this way, scenarios 4.3.2 and 4.3.2 could be fully represented in this scenario.

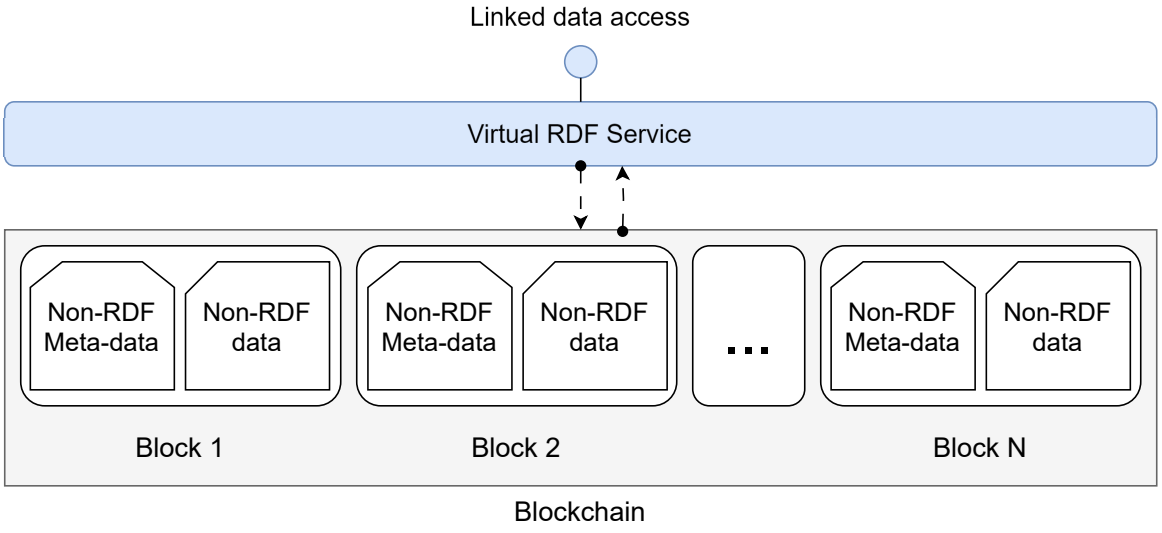

Figure 4.5: Blockchain with virtual RDF data and metadata

This approach counts with all the benefits that we reported without the problem that entails

storing RDF directly in the blockchain and, thus, is an evolution of the previous scenarios. In this scenario, the metadata and the data stored in the blockchain can be represented in the RDF format. Most of the virtual RDF services offer the capability of linking data and combining several data sources. Therefore, links between data can be generated on the fly or stored in another data source and combined with the virtual RDF data from the blockchain. Something similar can be done with the ontology mappings.

This scenario improves the efficiency, flexibility and scalability of blockchains by avoiding direct storage of RDF data in the blocks and instead using a virtualisation service to generate RDF from existing data in the blockchain. The main drawback of this approach is that it relies on a third-party service to generate virtual RDF. As far as we know, no research work presents a third-party service that generates virtual RDF from a blockchain.

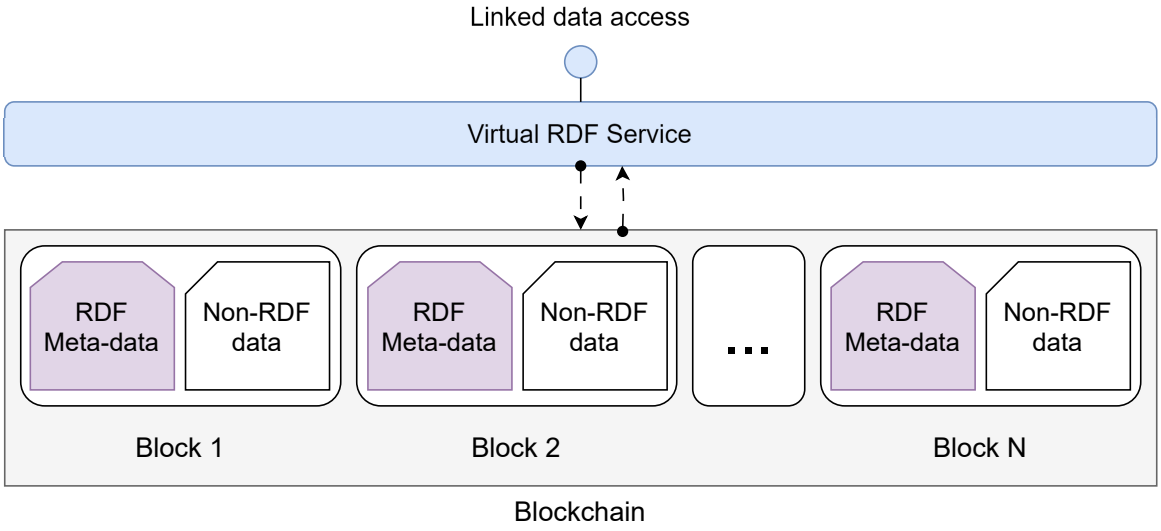

Figure 4.6: Blockchain with RDF metadata and virtual RDF

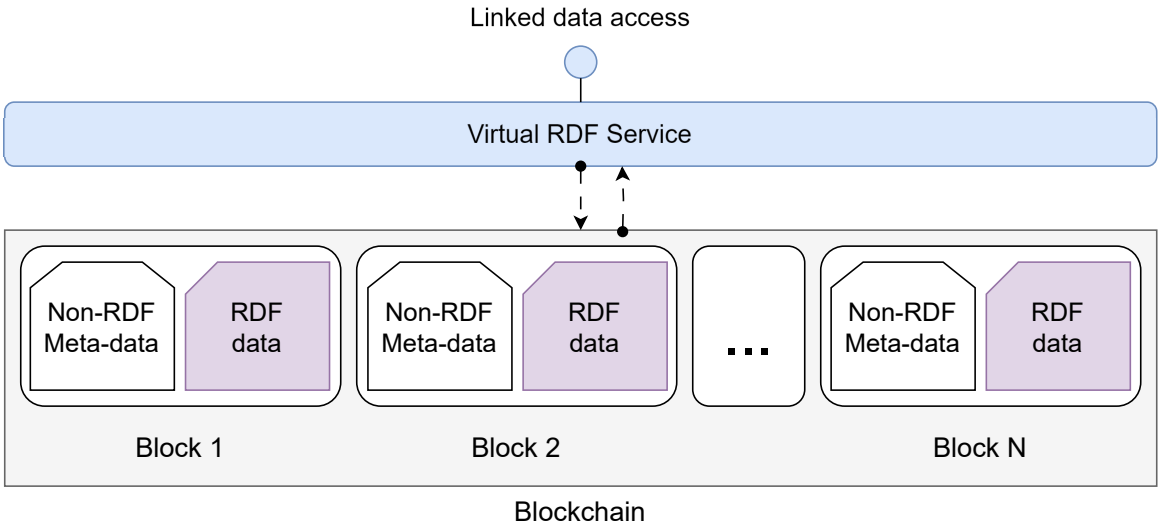

Figure 4.7: Blockchain with RDF data and virtual RDF

This proposed scenario has two different variants, where the metadata (Figure 4.6) or the data (Figure 4.7), but not both, are stored in RDF format, with an additional layer with a virtual RDF service. The purpose of this modified scenario is to harness the advantages of RDF in areas where it provides the most value without introducing unnecessary overheads to the entire blockchain system. By keeping part of the blockchain in its original format and applying RDF only to either the metadata or the data, a balance can be struck between maintaining operational efficiency and enhancing semantic richness and interoperability.

The motivation for both scenarios stems from the need for semantic interoperability, especially in environments where there is a benefit in querying and integrating the RDF-formatted blockchain component with other datasets. For example, if the metadata contains essential information that needs to be interoperably shared and linked with other datasets, storing metadata in RDF format would be beneficial. Conversely, if the transaction data itself is the main area of interest, converting it into RDF would provide enhanced data interoperability and querying capabilities.

Scenario 3.1. Writing data into the block using smart contracts

As depicted in Figure 4.8, smart contracts store data into the blockchain. In some blockchains, such as Hyperledger Fabric, once the smart contract is deployed, the contract information is stored in JSON; therefore, it is possible to store JSON-LD in the blockchain without the drawback of scenario 4.3.3. In other blockchain technologies, the information of smart contracts is stored in other formats, so software is required to convert the data stored in the blockchain by smart contracts to RDF; so, in this specific case, this scenario is an intermediate step between scenario 4.3.2 and scenario 4.3.3.

4.3.4 Scenario 4. Blockchain referencing another blockchain with semantic data

In this scenario, there are two blockchains as depicted in Figure 4.9. One chain is used to identify RDF resources that are stored in the other chain following any of the approaches reported in this section. As a result, in this scenario, the RDF data will be immutable, transparent, and double identified (by the URIs and the hashes of the first blockchain).

In this scenario, the semantic web counts with all the benefits that we reported from the blockchain.

4.3.5 Scenario 5. Semantic blockchain

This scenario consists of a new blockchain implementation that is meant to use semantic web technologies for the deployment of the blockchain, being the final state of the combination of semantic web and blockchain, i.e. the semantic blockchain, as depicted in Figure 4.10. As far as we know, there is no such implementation yet, but considering the relevance of the benefits that the semantic web offers to the blockchain and vice-versa, it is likely that an implementation will be proposed.

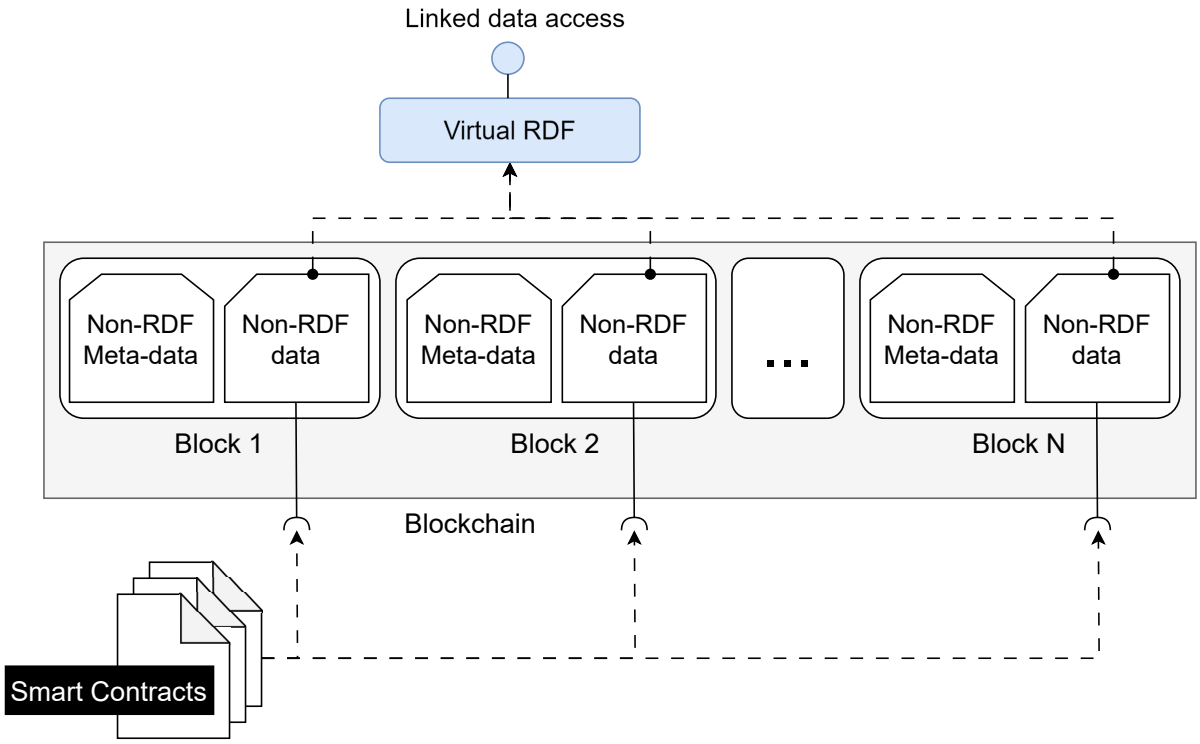

Figure 4.8: Blockchain with data into the block using smart contracts

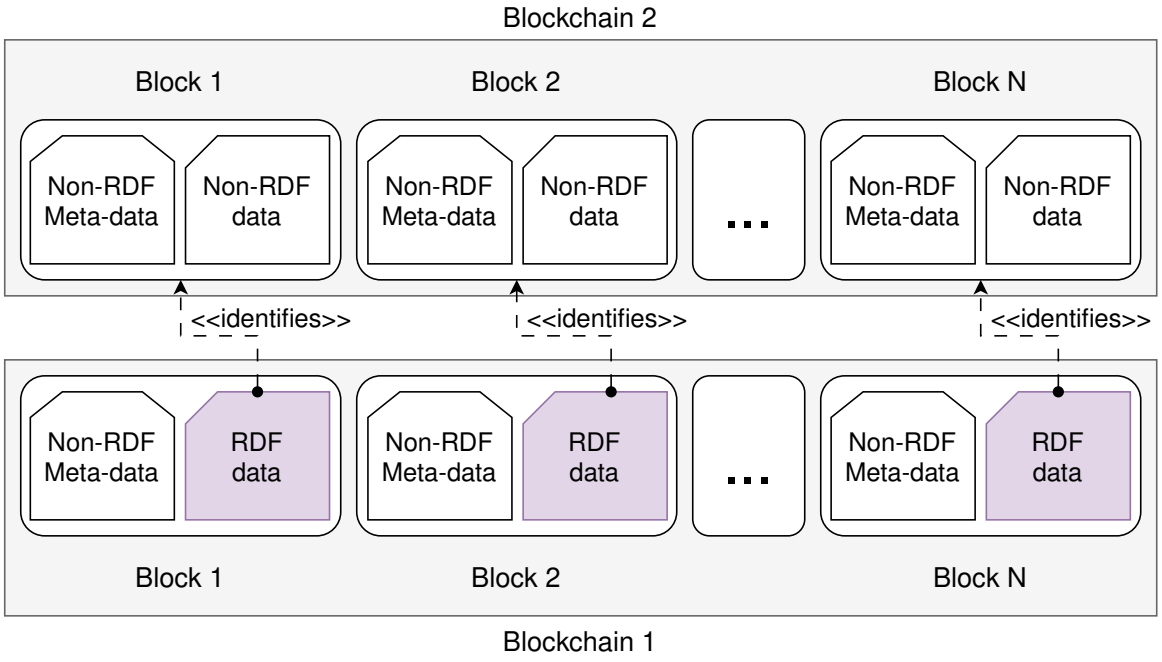

Figure 4.9: Blockchain referencing another blockchain with semantic data

This scenario takes full advantage of the synergies between blockchain and semantic web technologies. By creating a blockchain implementation that is designed to work with semantic web technologies from the outset, mutual benefits could be maximised.

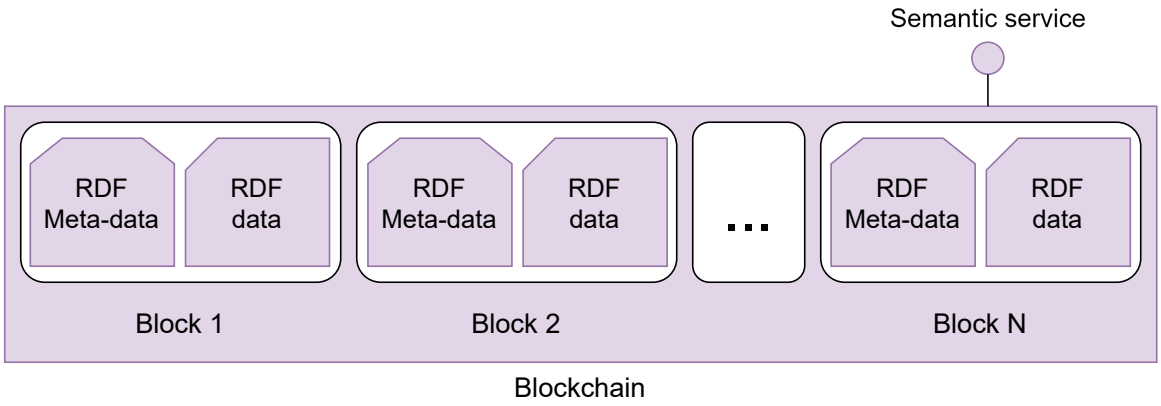

Figure 4.10: Blockchain implementation relying on semantic web technologies

4.4 Discussion

Once the scenarios have been explained, it should be noted that blockchain has many implementations, such as Ethereum, Bitcoin, or Hyperledger Fabric; so these scenarios may or may not apply, depending on implementation. Due to the fact that Ethereum is the second most famous blockchain and allows smart contracts, overcoming the limitations of Bitcoin [165], it has been decided to study this implementation in this thesis. In public blockchains (like Ethereum), these implementations often associate a cost to the amount of data that peers write in the chain. As a result, the same data written in the chain with a verbose format has a higher cost to be paid by a peer compared to having the data represented with a simpler format.

Finally, table 4.1 provides a summary of the scenarios that combine blockchain technology with the semantic web. The table presents the benefits and limitations of each scenario, helping to inform decisions about which approach might be the best fit for a given application.

Furthermore, table 4.2 summarises the benefits found in the blockchain that was previously analysed using the semantic web for each scenario, and table 4.3 summarises the benefits found in the semantic web using blockchain.

Scenario	**Advantages**	**Disadvantages**
Scenario 1	Facilitates search and analysis of blockchain data using SPARQL queries over metadata.	Only metadata of the blocks is expressed in RDF, so queries can only be used over metadata, not the content in the blocks.
Scenario 2.1	Allows SPARQL queries both over the block content and metadata. Facilitates the interlinking with other data sources.	Storing data in RDF format can require more space. In addition, it requires the use of a third-party service to generate virtual RDF and the metadata is not queryable.
Scenario 2.2		Requires the use of a third-party service to generate virtual RDF and metadata is not queryable.
Scenario 2.3		Metadata is not queryable.
Scenario 3, Scenario 3a & Scenario 3b	Offers all the benefits of the previous scenarios without the need to store RDF directly on the blockchain. Both metadata and data stored in the blockchain can be represented in the RDF format.	Requires the use of a third-party service to generate virtual RDF.
Scenario 4	RDF data is immutable, transparent, and double identified. Suitable scenario if immutable data is required.	The need to maintain and synchronise two blockchains may complicate implementation and increase storage and processing costs.
Scenario 5	Combines all the benefits that semantic web and blockchain can offer to each other.	No such implementation exists to date, so bringing it to practice may pose a significant technical challenge.

Table 4.1: Summary of the advantages and disadvantages of the scenarios proposed

Benefit	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**	**Scenario 5**
One language, multiple formats	X	X	X		X
Model data following well-known standards	X	X	X		X
Linking of data	X	X	X		X
Multiple data models	X	X	X		X
Search over blockchain	X	X	X		X
Blockchain data and metadata validation	Only metadata	Only data	X		X
Blockchain consistency validation	X	X	X		X
Data inference		X	X		X
Linked data virtualisation			X		
Interoperability	X	X	X		X

Table 4.2: Benefits to the blockchain when combined with the semantic web

Benefit	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**	**Scenario 5**
Data decentralisation	X	X	X	X	X
Identifiers for RDF resources				X	X
Data inmutability	X	X	X	X	X
Data transparency and privacy	X	X	X	X	X
Crowdsourcing data	X	X	X	X	X

Table 4.3: Benefits to the semantic web when combined with blockchain

Chapter 5

Analysing the efficiency of storing RDF data into the blockchain

> The analysis of the thing is not the thing itself.
>
> *Aaron Allston*

5.1 Introduction

The cost of writing data becomes especially relevant when a blockchain is used to store a knowledge graph (KG) in RDF format, which is known to be verbose in some of its serialisations. This verbosity incurs a notably higher cost in terms of energy, gas (the inherent cost of each transaction within the blockchain), and monetary expense. Therefore, although storing a KG within blockchain clearly has benefits in comparison to other formats when consuming data, it also has an expected higher cost. There is an ever-growing number of proposals that combine KGs with blockchain [167, 106, 31, 173], but there is a lack of analysis in terms of gas, querying data from the blockchain and storage size when the blockchain and the semantic web are combined.

In this chapter, two strategies for integrating RDF data into the blockchain are explored. First, the direct integration of RDF into the blockchain through regular transactions is studied. This approach allows us to evaluate the efficiency of the process in terms of gas consumption (the cost inherent to each transaction in the blockchain), block size and other relevant metrics. Secondly, the use of smart contracts as an alternative means of managing RDF within the blockchain is analysed. Once both studies have been performed, we will compare which approach is better for storing RDF on the blockchain and which type of serialisation is better for it.

This chapter is organised into two main sections: section 5.2 analyses the efficiency of storing RDF in different serialisations into blockchain and section 5.3 analyses the efficiency of storing RDF using smart contracts into blockchain. Finally, section 5.3.6 recaps the discussions and main findings.

5.2 Storing RDF directly into the blockchain

This section is organised as follows: subsection 5.2.1 defines the main objectives for this experiment, subsection 5.2.2 provides an analysis of which of the scenarios described in chapter 4 are the most suitable for storing RDF directly into the blockchain; subsection 5.2.3 presents the architecture followed in our experimental analysis; subsection 5.2.4 describes the data used in the experiment; subsection 5.2.5 explains how the experimentation was carried out and reports our results; and finally, subsection 5.2.6 recaps the discussions and main findings.

5.2.1 Objectives

The goal of this experiment is to provide an empirical analysis of how suitable it is to store RDF inside a blockchain due to the cost that it entails. Alternatively, storing JSON and using a virtualiser could bring the same benefits without the drawbacks of the former approach. The analysis consists in measuring how costly it is to store Turtle and JSON in terms of gas and how time-efficient it is to query both data.

Since most blockchains support storing data in their blocks, but not all of them support smart contracts, the scenarios to choose will be those that allow storing Turtle or JSON in transactions without using smart contracts.

The experiment is contextualised in a simulated research laboratory with 15 light sensors (photometers), an occupancy sensor, and a temperature sensor. The sensors send data to an agent that writes their data into two different Ethereum blockchains. In one of them, data are written in plain JSON, whereas in the other one, data are expressed in Turtle using the VICINITY ontology [33].

The analysis carried out aims to explore the following research questions:

- **RQ1**. What format has a higher cost in terms of gas when writing data in the blockchain, Turtle or JSON?
- **RQ2**. What format is faster when reading all the data from the blockchain, Turtle or JSON?
- **RQ3**. Considering a virtualiser that transforms on the fly JSON data into RDF, what is faster to query: a blockchain storing RDF in Turtle format or a blockchain that stores JSON and generates virtual RDF?

The scope of this subsection is to provide an empirical analysis of how suitable it is to store RDF inside a blockchain without smart contracts, as not all blockchain technologies support smart contracts, in the most appropriate scenarios described in Chapter 4.

5.2.2 Evaluation of scenarios

The first step is to analyse the scenarios defined in chapter 4 to identify in which of them RDF data can be stored into a blockchain.

- Scenario 1. This scenario does not allow storing data in the RDF format, since what is stored in RDF is the metadata of the blocks.
- Scenario 2. This scenario allows RDF data to be stored within the data of each block. Although it may be costly to store data directly in each transaction, this scenario fits the purpose of this chapter. Scenario 2.1 fits the purpose of this scenario, as it feeds data directly into the blocks of the chain. Scenarios 2.2 and 2.3 use smart contracts; therefore, they are out of the scope of the purpose of the experiment.
- Scenario 3. This scenario can store JSON data in the blockchain using a virtual RDF service that generates semantic data from heterogeneous data. This scenario fits the purpose of this chapter.
- Scenario 4. This scenario stores in the blocks the address of another chain that stores RDF; therefore, this scenario is out of the scope of this experiment because data storage would take place in a scenario 2 blockchain, and this blockchain would only link its blocks with those of the other blockchain.
- Scenario 5. This scenario is theoretical, and there is no implementation yet.

Therefore, it can be concluded that the best scenarios for storing RDF data and testing a blockchain with semantic content are scenarios number 2 and 3. Turtle is one of the most common formats to represent RDF and is not as verbose as RDF/XML [12]. Therefore, Turtle is the chosen format to store semantic data in the blockchain in this experiment.

Based on the chosen scenarios, the different experiments designed to address the three research questions formulated in this chapter are: first, the gas spent when storing Turtle (scenario 2) and JSON (scenario 3) data is measured in order to validate the first research question. Second, the time required to read from the Turtle and the JSON blockchains (scenario 2 and 3), respectively, is measured in order to validate the second research question. Finally, the time that it takes to perform the same query relying on the virtual RDF (scenario 3) and on the one provided by the RDF blockchain (scenario 2) are measured to validate the third research question.

5.2.3 Experimental architecture

In this chapter, data from IoT devices were chosen because in the context of data and blockchain, due to the decentralised nature of IoT and blockchain, storing data generated by IoT devices fits perfectly into the purpose of storing data on the blockchain. Currently, there are over 20 billion IoT devices deployed across a wide range of applications such as home automation, industrial, medical, military, etc [147]. Due to their importance in a wide range of applications, these IoT devices are often commercial and provide access (through databases, MQTT protocol, REST API, etc.), formats (presenting information in different formats such as JSON, CSV, XML, etc.), and heterogeneous models (even with the same format, information can be represented differently).

This heterogeneity poses a challenge for interoperability. Interoperability refers to the ability of a system to interact seamlessly with other interoperable systems, such as other IoT devices,

databases, or services. To address the heterogeneity issue, companies and academia have proposed various solutions, including leveraging semantic web and ontologies.

These IoT devices are, by definition, a decentralised network of devices, and the semantic web aims at storing data following a decentralised approach, so blockchain has key characteristics such as data decentralisation, data immutability, or data transparency and privacy, creating a perfect environment for IoT devices [70].

In this subsection, the architecture presented in Figure 5.1 has been endowed in order to perform the desired analysis. The architecture consists of a set of sensors whose data are stored inside a blockchain using the JSON format, and also the same data are stored using Turtle inside another blockchain. A multi-agent system is used to store the data generated by the IoT devices. Within this system, several agents are responsible for reading the data produced by IoT devices. As these agents receive the IoT data, they store these data in the blockchain through transactions. Then, relying on this infrastructure, a set of tests have been performed to find answers to the research questions reported in subsection 5.2.1.

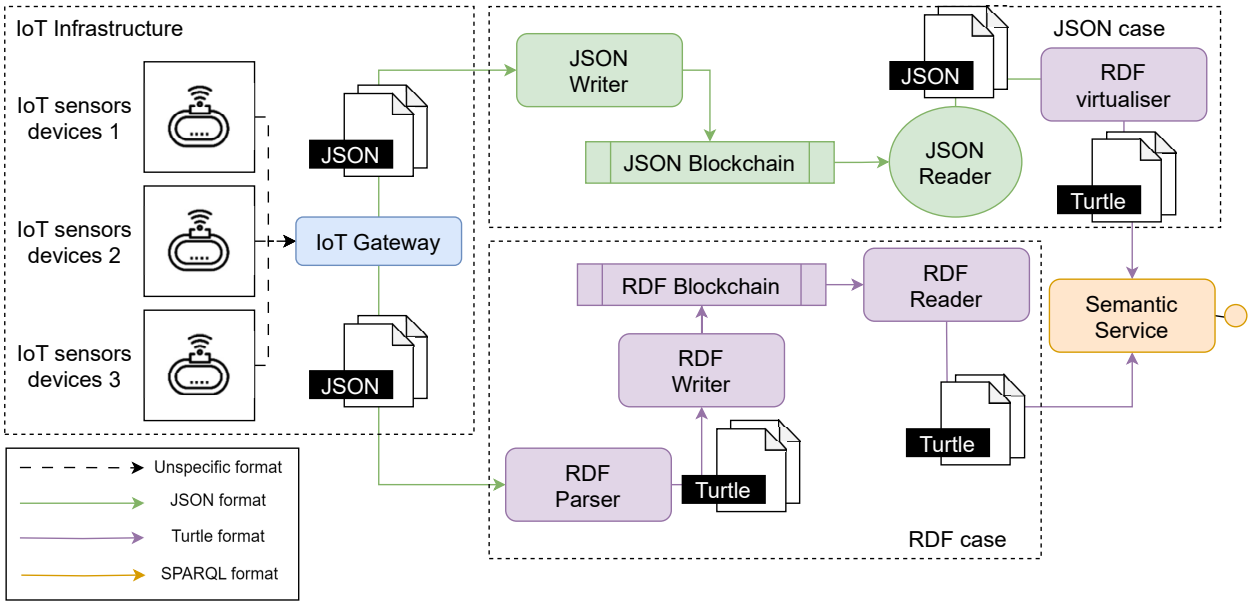

Figure 5.1: Experimental architecture

Next, the different components of the architecture are explained:

- **IoT gateway:** the sensors within the architecture are 15 lightbulb sensors, 1 temperature sensor, and 1 occupancy sensor. All the data reported by the sensors are converted into JSON. These devices send their data to the *IoT Gateway*, which forwards such information to the *JSON Writer* and the *RDF Parser*.
- **RDF Parser:** this component receives the JSON data from the *IoT gateway* and, using a fixed template, injects such data into the template using JSONPath expressions. Then, it forwards the RDF data to the *RDF Writer*.
- **Blockchain RDF/JSON Writer:** although the architecture features two different agents for this task, conceptually, they perform the very same task. Both receive a

document and write such document in the blockchain. The *RDF Writer* stores the received RDF documents in the RDF blockchain, serialised as Turtle, and the *JSON Writer* stores the JSON documents received in the JSON blockchain.

- **Blockchain:** in the architecture, data (either in RDF or in JSON) are stored in an Ethereum blockchain. Their functionality is the same since the blockchain is agnostic to the data format.
- **JSON/RDF Reader:** similarly to the writer, although the architecture counts with two different agents for this task conceptually, they perform the same task. These agents read the information within their respective blockchains. As a parameter, they can receive the number of transactions to be read, providing as a result the collection of documents that were stored in those transactions.
- **RDF virtualiser:** this component in the architecture is implemented with a software called Helio [35]. It reads a number of transactions from the blockchain and, relying on a set of translation mappings, generates an RDF graph of the documents stored in the RDF blockchain, i.e., the *Virtual RDF*. Therefore, the virtual RDF produced is exactly the same as the one provided by the *RDF Reader*.
- **Semantic service:** this component receives a SPARQL query and returns the query result. Depending on how it is configured, it provides an endpoint to query JSON data through the *Virtual RDF* or RDF data through the *RDF Reader* to answer the query.

The goal of this architecture is to provide a playground where different measurements can be taken. First, the gas consumption when storing the Turtle or the JSON documents. Second, the time that it takes to read Turtle and JSON documents from the blockchain. Third, the time that it takes to answer a query with the data stored in a set of transactions. The architecture is composed of two of the scenarios previously analysed and how the blockchain works in each scenario is explained in detail below.

On the one hand, Figure 5.2 depicts the scenario in the case of storing Turtle within the blockchain. It can be seen that this scenario is the same as the one presented in scenario 2.1 of chapter 4. The data flow would be as follows, documents in Turtle format are progressively introduced in the first block one by one (step 1), until a block is generated and these documents would be introduced in the second block (step 2), and so on (step 3). If a query is desired, the semantic data of the desired blocks will be collected (step 4) and finally with these semantic data the SPARQL query will be performed (step 5).

On the other hand, Figure 5.3 depicts the scenario of storing JSON within the blockchain. It can be seen that this scenario is the same as the one presented in scenario 3 of Chapter 4. The data flow would be as follows; documents in the JSON format are progressively introduced in the first block one by one (step 1) until a block is generated and these documents would be introduced in the second block (step 2), and so on (step 3). If a query is desired, the JSON data of the desired blocks will be collected and converted into RDF on the fly (step 4) and finally, with these semantic data, the SPARQL query will be performed (step 5).

As a result, by performing these measurements in terms of gas and time for reading data and performing SPARQL queries, the research questions introduced in subsection 5.2.1 will be

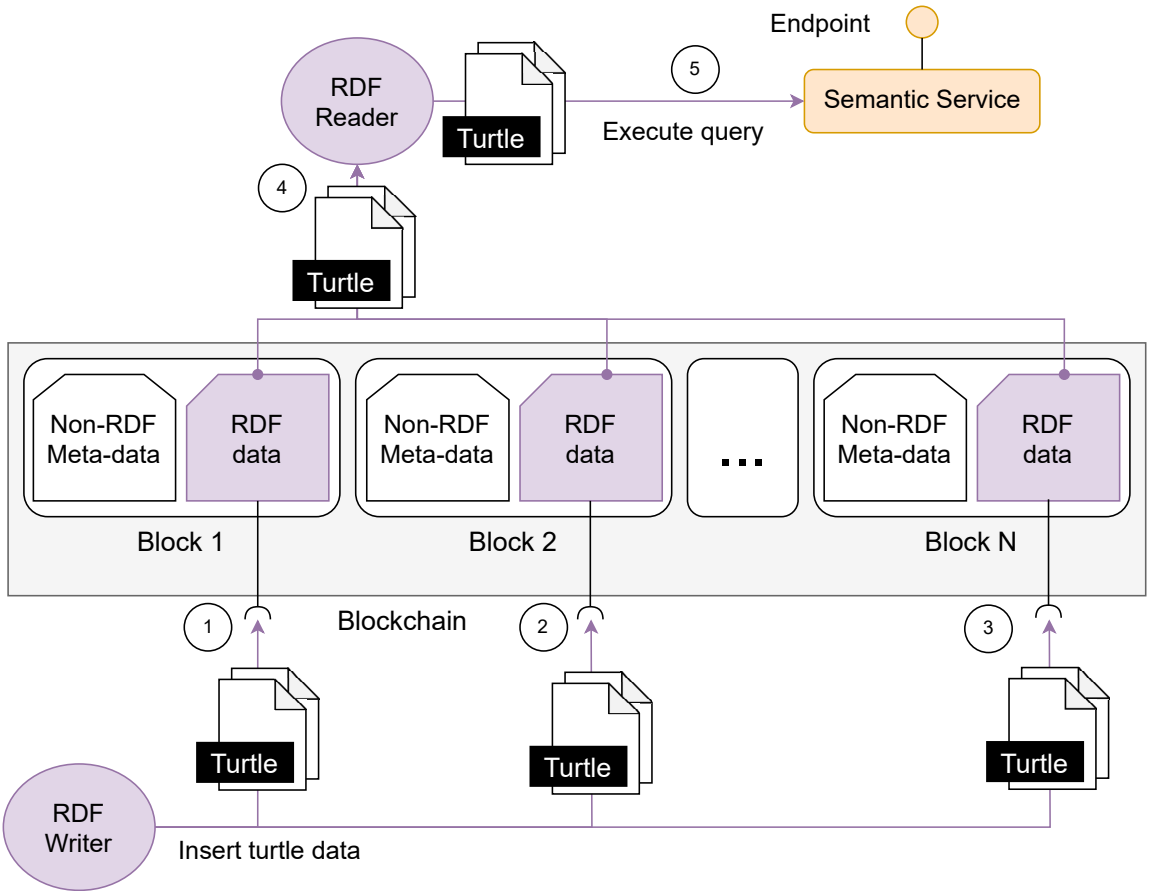

Figure 5.2: RDF case scenario

validated, analysing the feasibility of storing Turtle directly or storing JSON and using a virtualiser to keep the RDF benefits.

5.2.4 Data used for the experiment

The data to be introduced in the blockchain consists of different devices: 15 light sensors, 1 occupancy sensor and 1 temperature sensor. The set of 15 light sensors (listing 5.1), integrated into smart bulbs, transmit data on their operating status, indicating whether they are activated or not. The occupancy sensor (listing 5.2) is designed to provide information on the current number of occupants in a home. The data is structured in the JSON format and contains attributes such as sensor type, number of occupants, a timestamp indicating when the data was generated and other relevant metadata such as device name, city, country and building characteristics. Finally, the temperature sensor (listing 5.3) provides a structured data output indicating the current ambient temperature, expressed in degrees Celsius, accompanied by the corresponding time stamp. The data stored in both blockchains contains the same information, although in different formats.

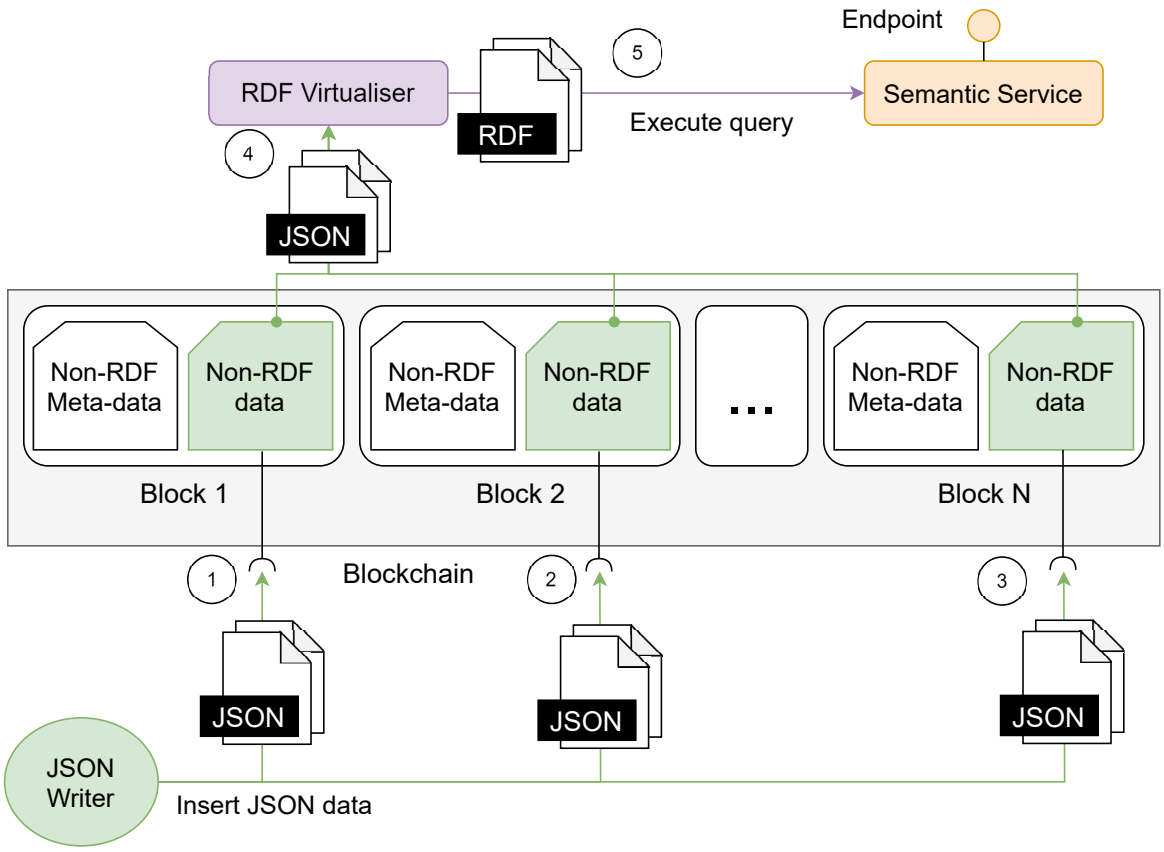

Figure 5.3: JSON case scenario

```
{
"type":"Light",
"id": i,
"state":lightBulb.get(i),
"timestamp": time.millis(),
"DeviceName":"Light hardware",
"City":"Madrid",
"Country":"Spain",
"Building":"MySmartHouse"
}
```

Listing 5.1: Template of light data payload

```
{
"type":"occupation",
"number":workersInside,
"timestamp":time.millis(),
"DeviceName":"Occupacy hardware",
"City":"Madrid",
"Country":"Spain",
"Building":"MySmartHouse"
}
```

Listing 5.2: Template of occupancy data payload

```
{"type":"Temp",
"T":temp,
"Type":"Celsius",
"timestamp":time.millis(),
"DeviceName":"Temp hardware",
"City":"Madrid",
"Country":"Spain",
"Building":"MySmartHouse"}
```

Listing 5.3: Template of temperature data payload

5.2.5 Experimental analysis

For our experiments, we compared the results of each 2,000 transactions in order to determine how the results vary across the transactions (in terms of gas and time to read the data). Firstly, the gas consumption for storing data in both Turtle and JSON formats was measured. Specifically, the amount of gas required for each transaction involving the storage of these formats was recorded in order to assess the efficiency and cost implications of using one or the other format on the blockchain.

Secondly, all the times reported as box plots are measured in seconds, reporting the results of executing 10 times each experiment. The statistical test performed to establish whether the results have statistically significant differences is the well-known Iman–Davenport test [74], with a confidence level of 95%. This test outputs a p-value; if this value is greater than 0.05, it means that there are no statistical differences between the results, i.e., they can be considered the same.

Finally, SPARQL queries were executed on the stored data in Turtle; the times for reading data from the blocks and querying these data were measured. In the case of JSON, the times to read data from the blocks, virtualise the JSON into RDF and query these data are measured.

Analysis of gas consumption

In this experiment the Turtle and JSON data generated were stored in different blockchains. Both data types contained the same information, however data expressed in Turtle required around 6,000 characters, whereas JSON data required approximately 550 characters to encode the same information. Table 5.1 shows the total average gas consumed after storing all the Turtle and JSON documents.

Method	Average Gas Consumed
JSON	32,850
Turtle	227,782

Table 5.1: Gas consumed by Turtle and JSON

This table shows the average gas consumed by all transactions. This is because the price of gas does not increase with time on our Ethereum blockchain, so the cost of storing similar content does not change but could change if the price of gas increases.

As it can be observed, storing data in Turtle requires an amount of gas that is approximately 10 times more than the one required by the information serialised in JSON. In this case, there is no need of applying any statistical test since the magnitude of such difference makes results clear.

Analysis of the time required to read transactions

In this experiment, we measured the time that takes reading a set of 2,000 transactions from the JSON and the RDF blockchains, respectively. In addition, the time to send the data

stored in the blockchain from the *JSON reader* to the *RDF virtualiser* is included in the results. Figure 5.4 depicts the results of this experiment.

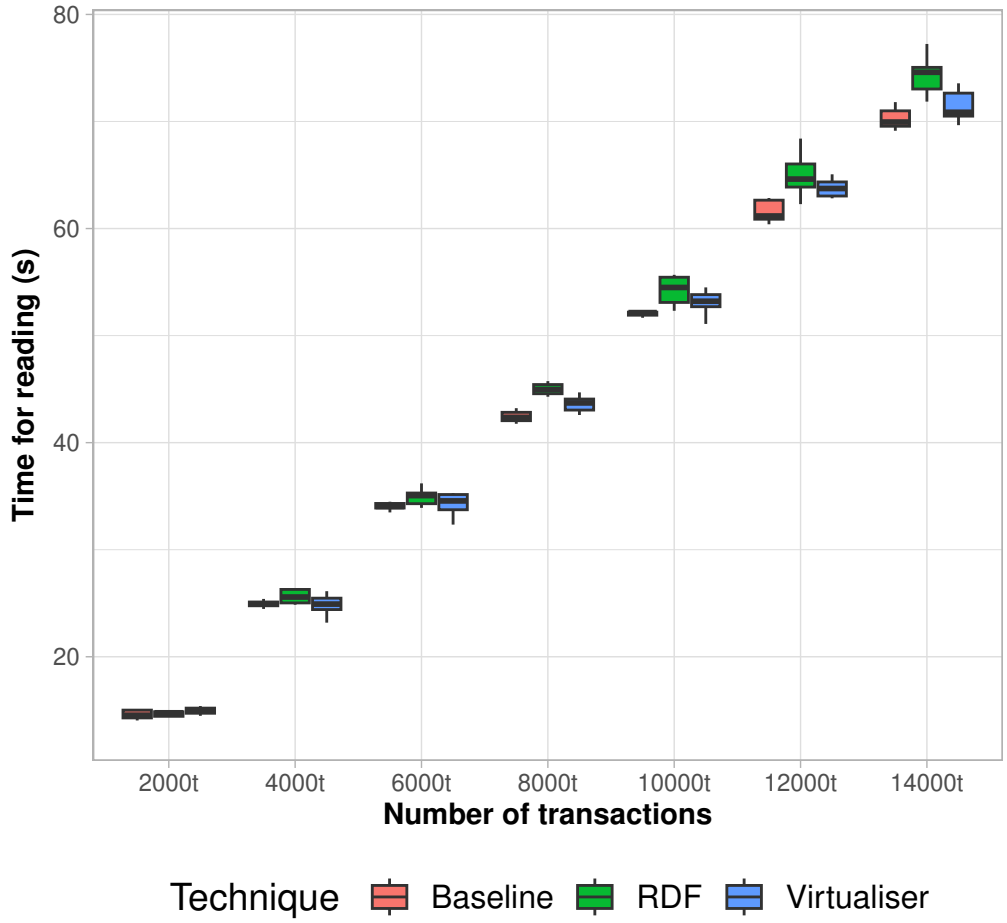

Figure 5.4: Time for reading transactions

The JSON data stored in the blockchain is used as a baseline, without applying any method to convert the JSON data into RDF; the RDF is the Turtle stored in the blockchain, and the virtualiser is the time to read JSON data and convert it into RDF. As can be observed, the reading times for the three cases are close enough. The statistical test applied over their average values in order to ensure their statistical equivalence returns the following p-values: between JSON and the RDF generated by the virtualiser, the output is 0.13; between JSON and RDF, the output is 2.66×10^{-4}, and between the virtualiser and RDF 0.02×10^{-4}. As a result, it can be concluded that the reading times are statistically equivalent between JSON and the virtualiser. Instead, between RDF and JSON and between RDF and the virtualiser, there is a statistical difference. As a result, reading JSON and optionally feeding the virtualiser is faster than just reading the RDF.

Analysis of the time required to issue SPARQL queries

In this experiment, the time that it takes to read the blockchain plus the time that it takes to solve a SPARQL query was measured (this includes, in the case of the virtualiser, the time to read, virtualise the JSON data and query the RDF generated). By evaluating these steps together, it captures the full read time from retrieving data from the blockchain to process queries. The query issued asked about all the known data in the blockchain. Figure 5.5 depicts the results of this experiment, and the query performed can be found in the listing 5.4.

```
1 SELECT * WHERE {
2     ?s ?p ?o .
3 }LIMIT 14000
```

Listing 5.4: SPARQL query for the last experiment

In light of the results, using the virtualiser allows us to solve SPARQL queries faster than just aggregating RDF documents from the *RDF Reader*. The difference is due to the fact that the translation to RDF is faster and produces a whole RDF document; instead, the *RDF Reader* needs to aggregate all the documents into a former one before solving the query and due to the size of this data in the blockchain, it is slower than reading JSON and producing RDF externally with the virtualiser. This behaviour explains the linear growth of the results in the chart.

The Iman-Davenport test outputs a p-value of 0.02, therefore it can be concluded that there are statistical differences and, thus, using a virtualiser in this context is better than storing, reading and querying RDF directly.

5.2.6 Discussion

This subsection presents an empirical study that aims at answering the research questions proposed in subsection 5.2.1. These questions revolve around whether storing Turtle in a blockchain is efficient and if alternatives exist in order to keep the benefits of RDF while avoiding its drawbacks. The experimental results led to the following answers.

RQ1: at the light of the results reported in the previous subsection we can conclude that writing Turtle is more than 10 times more expensive than writing JSON.

RQ2: results from the previous subsection advocate that reading JSON is faster than Turtle; even feeding with the read data, a virtualiser is faster than reading Turtle.

RQ3: the previous subsection also proves that querying the virtualiser is faster than reading and querying Turtle from the blockchain.

As a conclusion of our empirical analysis, storing Turtle in a blockchain brings clear benefits for consuming data, e.g., being able to query data. However, Turtle has some drawbacks: i) reading the data from the blockchain takes more time than reading the same data in another format like JSON, and ii) writing Turtle in a blockchain has an elevated cost in terms of gas.

As a result, in this subsection a virtualiser to translate JSON on the fly into RDF was

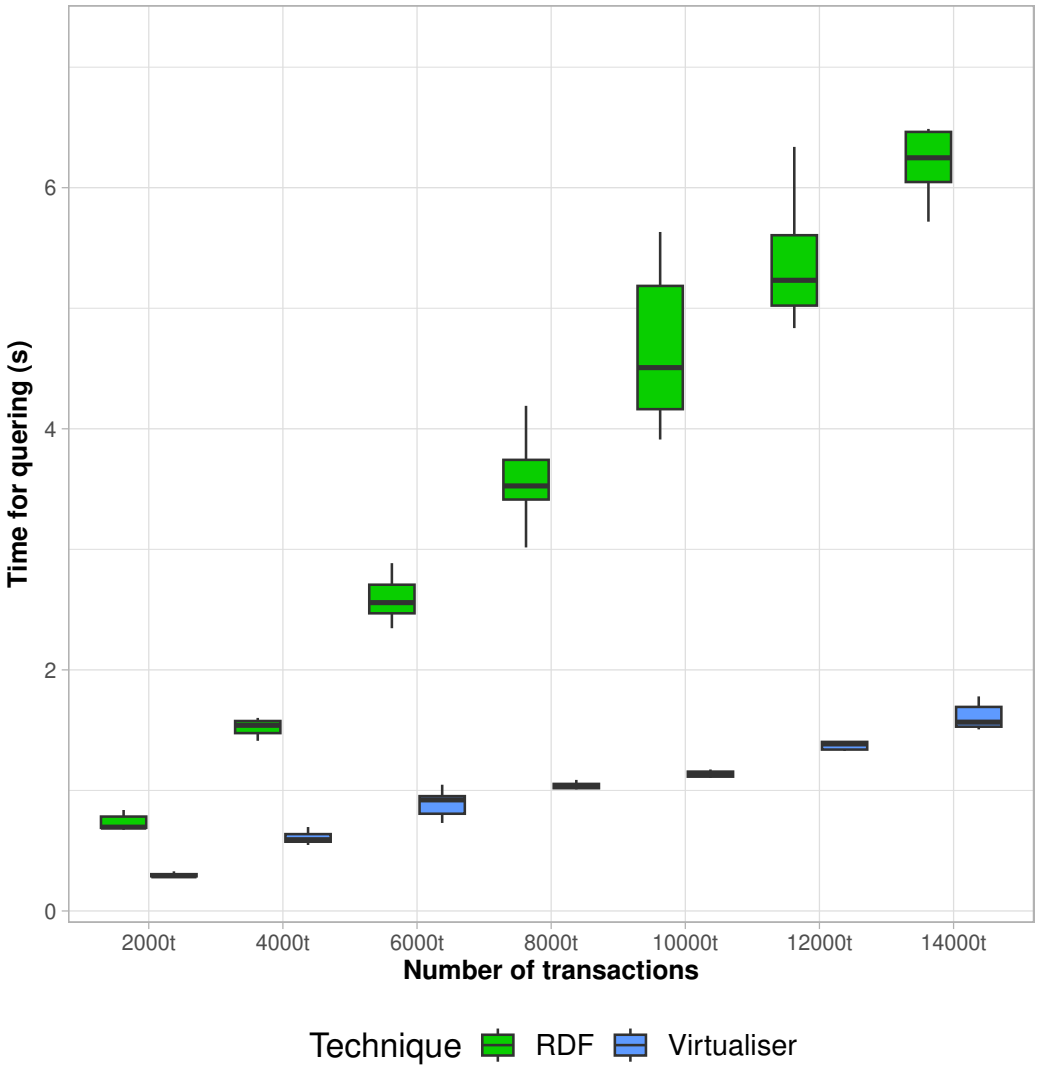

Figure 5.5: Time for querying data in the blockchain

analysed. The experimental results achieved prove that using a virtualiser under the studied circumstances is more efficient than using Turtle. It has the same benefits, but none of its drawbacks. However, this subsection only analyses Turtle, but the RDF supports several serialisations, one of which is JSON-LD. Given the similarities between JSON and JSON-LD in terms of their structure and nature, it can be inferred that the results of storing data in JSON would be analogous to those of storing data in JSON-LD.

5.3 Using smart contracts to store RDF into the blockchain

As evaluated, storing JSON on the blockchain is more efficient than storing Turtle. As JSON has a similar structure to JSON-LD, in this section, the data generated by IoT devices will be stored on the blockchain using smart contracts with a JSON-LD format to analyse the

differences between storing data through transactions and through smart contracts. This comparative study allows one to discern which scenario is most suited to the goal of embedding relevant JSON-LD information in smart contracts or in blockchain.

This section is organised as follows: subsection 5.3.1 defines the main objectives for this experiment, subsection 5.3.2 analyses the scenario that is best suited to bring data into the blockchain through smart contracts on Ethereum; subsection 5.3.3 presents the architecture followed in our experimental analysis; subsection 5.3.4 describes the data used in this experiment; and, finally, subsection 5.3.5 explains how the experimentation was carried out and reports our results.

5.3.1 Objectives

This section presents an experiment in which data is stored into blockchain, in which an empirical analysis is performed to establish the benefits and costs of storing a RDF data on a blockchain using smart contracts in contrast with storing RDF data using the JSON-LD format directly in the blockchain, using previously analysed scenarios.

Therefore, this chapter presents an experiment in which heterogeneous data from IoT devices become semantically interoperable and, due to the decentralised nature of the IoT devices, stored in a blockchain-based environment. The experiment is based on established techniques for data harmonisation, extended to adapt to the blockchain. Additionally, the experiment provides an interoperable semantic layer over the IoT data stored on the blockchain. This layer can be used for other operations on integrated data, such as visualisation, linking with other storage systems, or performing SPARQL queries, inter alia.

The experiment is contextualised in a series of buildings containing IoT devices with varied information. The sensors send data to an agent that writes its data into two different Ethereum blockchains. In one of them, data are written in plain JSON, whereas on the other one, data are expressed in RDF in the Turtle serialisation using the AURORAL[1] and the ETSI Smart Applications REFerence (SAREF)[2] ontologies.

Within the blockchain ecosystem, interacting with a smart contract is a transaction, so it has been decided to separate the terms. In this subsection, the term transaction will refer to a transaction without the interaction with the smart contract, while the term smart contract will refer to a transaction that interacts with a smart contract deployed on the Ethereum network. The analysis carried out aims to explore the following research questions:

- RQ1: What has a higher cost when writing data in the blockchain, transactions or smart contracts?
- RQ2: What is faster when writing data from the blockchain, transactions or smart contracts?
- RQ3: What is faster when reading data from the blockchain, transactions or smart contracts?

[1] https://auroral.iot.linkeddata.es
[2] https://saref.etsi.org/

5.3.2 Evaluation of scenarios

The scope of this chapter is to provide an empirical analysis of how suitable it is to store RDF inside a blockchain using smart contracts and compare it to the use of transactions without the use of smart contracts. As in section 5.2, the first step is to analyse the scenarios in which RDF can be stored in an Ethereum blockchain by using smart contracts. This evaluation will only consider Ethereum-based blockchains and other blockchain technologies are out of the scope of this thesis.

- Scenario 1. This scenario does not allow storing RDF since what is stored in RDF is the metadata of the blocks.
- Scenario 2. This scenario allows RDF to be stored within the data of each block. Therefore, this scenario fits the purpose of this chapter.
  - Scenario 2.1. Due to the fact that this scenario does not use smart contracts, it is out of the scope of the purpose of the experiment.
  - Scenario 2.2. This scenario fits the purpose of this scenario, as it feeds data directly into the blocks of the chain through smart contracts.
  - Scenario 2.3. Despite this scenario using smart contracts, it is out of the scope of this experiment because Ethereum does not support this type of contract.
- Scenario 3. This scenario stores content (JSON) on the blockchain using a virtual RDF service that generates semantic data from heterogeneous data. Although this scenario fits the purpose of this chapter, in this scenario it is assumed that the entire block will be converted to RDF, therefore, this scenario is out of the scope for the purpose of this chapter.
- Scenario 4. This scenario stores in the blocks the RDF address of another chain, so this scenario is out of context of this chapter.
- Scenario 5. This scenario is hypothetical and there has been no implementation yet.

Therefore, it can be concluded that the best scenario for storing RDF using smart contracts is scenario 2.3; furthermore, the best scenario for storing RDF directly into the blockchain is scenario 2.2.

5.3.3 Experimental architecture

In this subsection, Figure 5.6 presents an overview of the architecture that has been developed in this experiment. The architecture can be divided into four main sections. The first one is the IoT Infrastructure, where the IoT data to be entered into the blockchain is generated. Then, on the one hand, there is the scenario where the data is stored through smart contracts and, on the other hand, where the data is stored directly in the blockchain. Finally, an endpoint is provided to perform SPARQL queries.

Next, the different components of the architecture are explained:

- **IoT gateway:** regarding the data to be introduced in the smart contracts, this

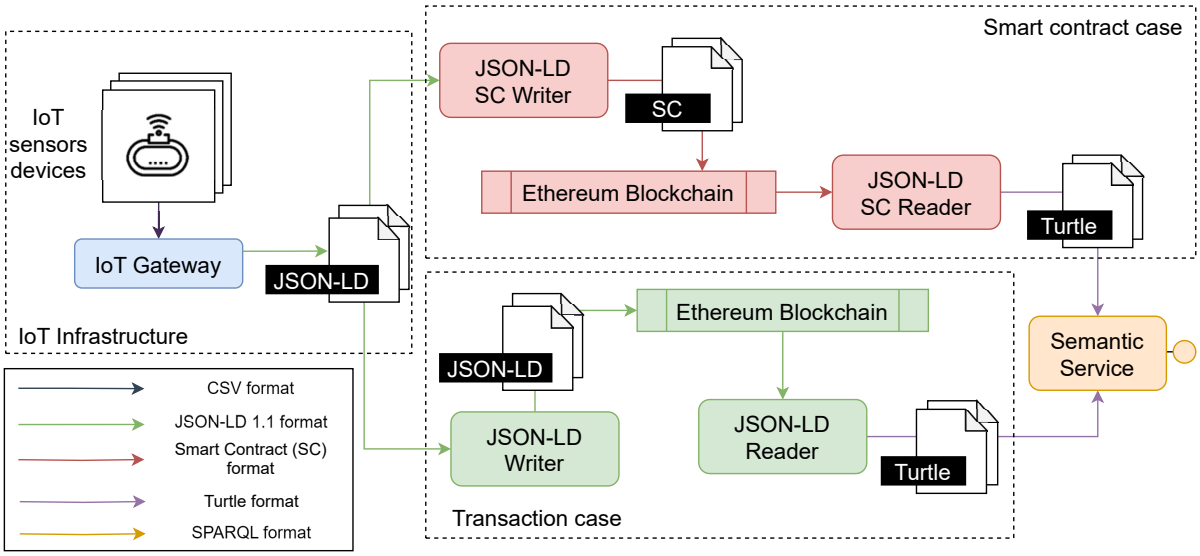

Figure 5.6: Experimental architecture

experiment used data uploaded to the Kaggle, a platform for predictive modelling and analytics competitions where users can upload public datasets. The chosen source is a dataset of IoT sensors in an office building that includes five types of measurements: CO2 concentration, room air humidity, room temperature, brightness and motion sensor data [69]. These devices send their data to the *IoT gateway* that forwards such information to the *JSON-LD Writer* and the *JSON-LD SC Writer*. The data reported by the dataset is in CSV, and in this component, the data are converted to JSON-LD 1.1.

- **JSON-LD SC Writer:** this component receives the JSON-LD data from the *IoT gateway*, converts the JSON-LD data with the data model of the smart contract and injects it into the blockchain via smart contract transactions.
- **JSON-LD Writer:** this component receives the JSON-LD data from the *IoT gateway* and injects it into the blockchain via transactions.
- **Blockchain:** in the architecture, JSON-LD data is stored in an Ethereum blockchain. Their functionality is the same since the blockchain is agnostic to the data format.
- **JSON-LD SC Reader:** similar to the *JSON-LD SC Writer*, the architecture has a component to read the data stored in the blockchain via smart contracts.
- **Smart contract parser:** Since the blockchain stores data in its own format, the data is retrieved in JSON-LD using a template.
- **JSON-LD Reader:** similar to the *JSON-LD Writer*, the architecture has a component to read the data stored in the blockchain via transactions.
- **Semantic service:** this component receives a SPARQL query and returns the query result. Depending on how it is configured, it relies on the *JSON-LD SC Reader* or on the RDF output by the *JSON-LD Reader* to answer the query.

The objective of this architecture is to establish a testing environment where various metrics can be assessed. Initially, the focus is on comparing the gas consumption between JSON-LD storage via smart contracts and JSON-LD storage through transactions facilitated by the Writers. The next measure involves the duration required to write and read JSON-LD, either stored by smart contracts or by transactions, with the latter dependent on the Readers. Lastly, the architecture evaluates the time necessary to respond to a query using data from a series of transactions when the JSON-LD is sourced from the RDF reader.

The scenario depicted in Figure 5.6 has been configured to perform the desired analysis. This configuration comprises a set of sensors that publish data in CSV format. This data is stored in a blockchain via smart contracts in JSON-LD format. In addition, the same data is stored in another blockchain via transactions, also using the JSON-LD format. Based on this infrastructure, a series of tests have been performed to answer the research questions posed in subsection 5.3.1. Figure 5.7 shows the scenario where JSON-LD is stored within the blockchain through smart contracts. As can be seen, this configuration matches the one presented in scenario 2.2 of Chapter 4.

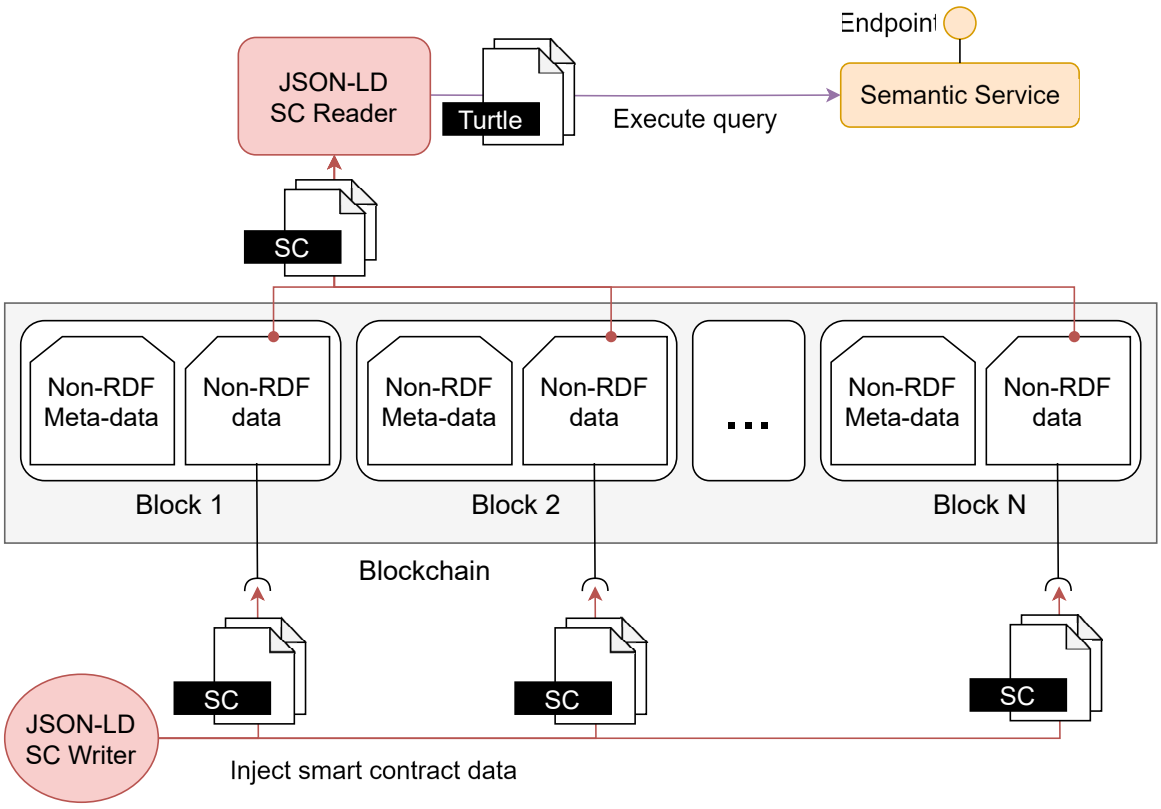

Figure 5.7: Smart contract case scenario

Figure 5.8 depicted the scenario in the case of storing JSON-LD within the blockchain through transactions. It can be observed that this scenario is the same as the one presented in scenario 2.1 of chapter 4.

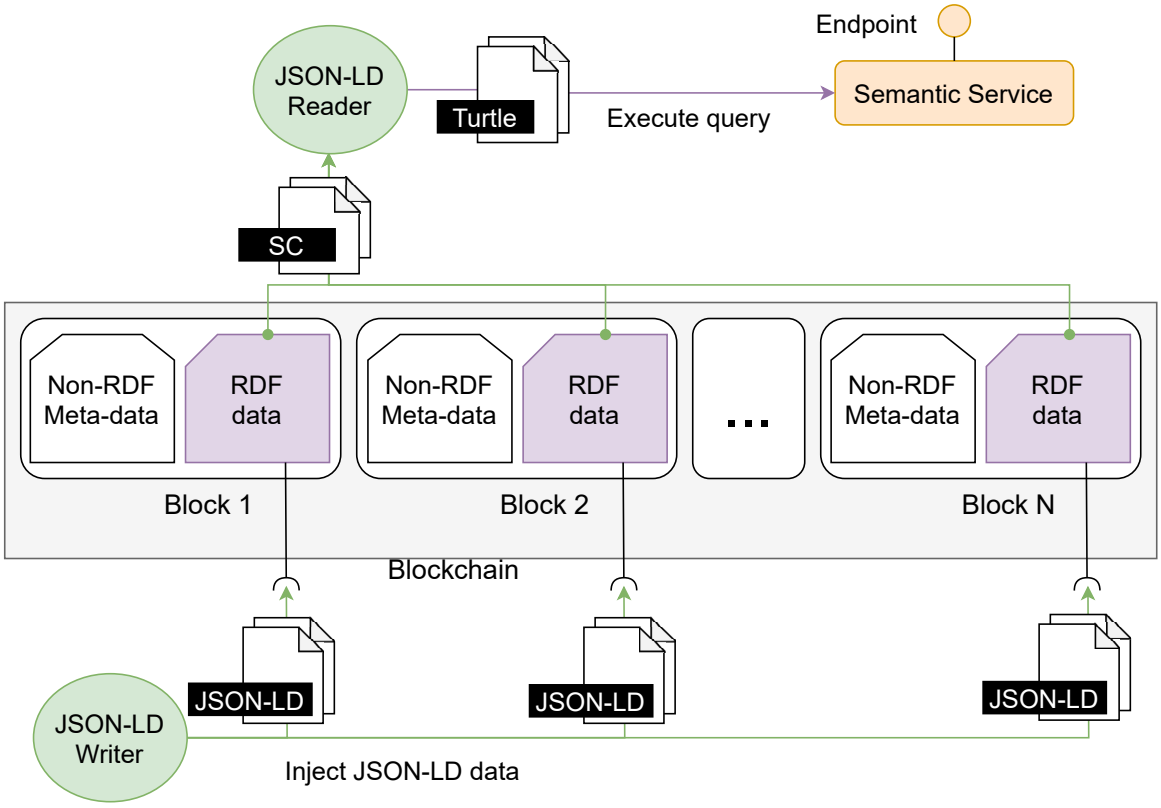

Figure 5.8: Regular transaction case scenario

5.3.4 Data used for the experiment

As in the previous chapter, in the context of data and blockchain, data generated by IoT devices fits perfectly the purpose of storing data on the blockchain, as explained in subsection 5.2.5. The data to be stored on the blockchain consists of different devices located in different buildings, and each device provides information on humidity, CO2, temperature and light level. The data are structured in CSV format and openly published on the Kaggle platform [69]. In order to have the data according to an RDF serialisation, JSON-LD is used. Listing 5.5 shows the template of the data generated by the *IoT gateway*.

```
1  {
2      "@context": "https://raw.githubusercontent.com/jucanbe/Semantic-BD-
           Blockchain/main/context/devices.json",
3      "@id": "https://example.es/example/building1",
4      "hasId": "Ethereum_Hash_Address",
5      "isSpaceOf": "Edificio Central",
6      "city": "Madrid",
7      "hasSpace": "Office 101",
8      "contains": [
9          {
10           "@id": "https://example.es/example/humiditySensor1",
11           "@type": "humiditySensor",
12           "measurement": {
13             "timestamp": "2023-09-12T10:05:00Z",
```

```
            "value": 65.4,
            "isMeasuredIn": "humidity"
             }
        },
        {
          "@id": "https://example.es/example/co2Sensor1",
          "@type": "co2Sensor",
          "measurement": {
            "timestamp": "2023-09-12T10:05:00Z",
            "value": 400,
            "isMeasuredIn": "co2"
            }
        },
        {
         "@id": "https://example.es/example/lightbulb1",
         "@type": "lightbulb",
         "measurement": {
           "timestamp": "2023-09-12T10:05:00Z",
            "value": 500,
           "isMeasuredIn": "lux"
            }
        },
        {
         "@id": "https://example.es/example/thermometer1",
         "@type": "thermometer",
         "measurement": {
            "timestamp": "2023-09-12T10:05:00Z",
            "value": 24.5,
            "isMeasuredIn": "degreeCelsius"
            }
        }
    ]
}
```

Listing 5.5: Template of a device data payload

Since the data stored in the blockchain is JSON-LD 1.1, and JSON-LD 1.1 is part of the semantic web, these data follow an ontology. The purpose of ontologies is their reusability, so it was decided to use ontologies that most closely align with this work: the AURORAL and SAREF ontologies. These ontologies represent the recurring domains in the domain of intelligent applications (in this work, the ontology covers all devices and properties that are used). From the AURORAL ontology, the AURORAL adapters ontology was used, which is depicted in Figure 5.9. Then, the context provided in the template was built using the "Device" class of the AURORAL core ontology. This class its related with the SAREF ontologies.

Each device has an Ethereum node; then, the field *hasId* is the hash generated for each device by Ethereum.

Three ontologies were used from SAREF. The ontology **SAREF** is a shared model of consensus that facilitates the matching of existing assets in the smart applications domain. Secondly, the **SAREF4BLDG** extension was created for building information and to allow the representation of devices and physical objects in building spaces. Finally, the **SAREF4CITY**

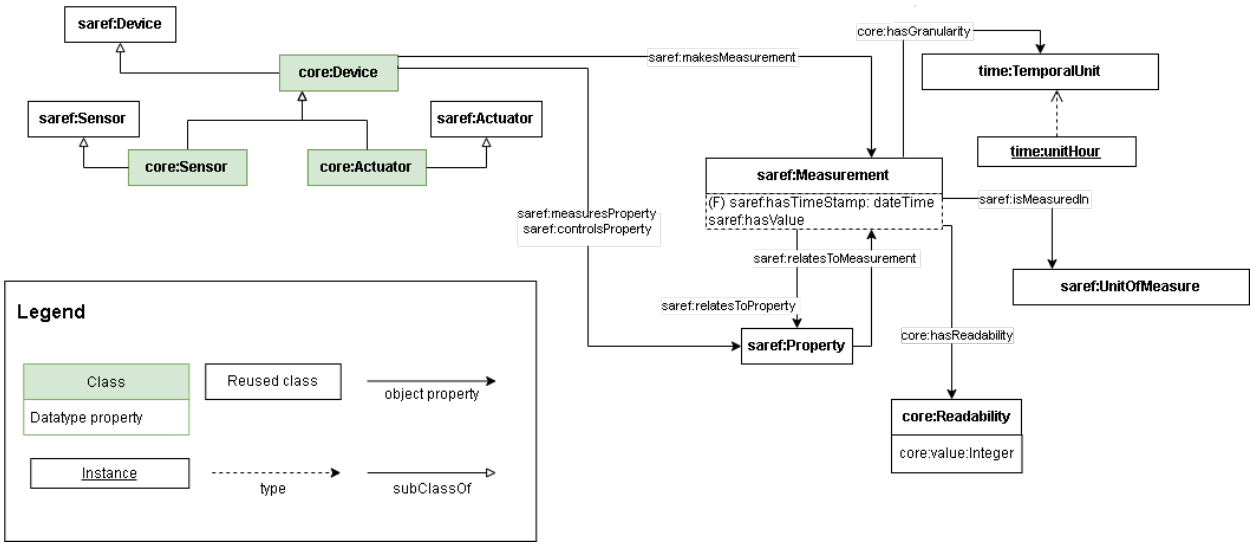

Figure 5.9: AURORAL adapters ontology [58]

extension is a SAREF extension for the smart cities domain.

Solidity was used for the smart contracts. Solidity is the most popular smart contract language [51] and was developed specifically for use in Ethereum. The contract that has been created can be found in listing 5.6, and available on GitHub[3]. This contract has an event function ("Event"), which will be used to generate an event every time an "emit" call is issued. With this function, the state generated by the event will be recorded in the blockchain. The contract has a mapping, which works similarly to a Java Hashmap function. With this mapping, an address of the chain (each address is unique to each device) of blocks is linked to the "Struct" data structure so that the last state of a device can be queried. Finally, the contract has a storage function, which stores the state of the device in the mapping and emits an "Emit" call with which the "Event" function will be called to query the transaction history.

```
1  pragma solidity 0.8.17 <0.9.0;
2
3  contract Sensor_IoT {
4
5      struct sensor{
6          string context;
7          string identifier;
8          string buildingName;
9          string location;
10         string office;
11         uint256 timestamp;
12         uint256 lux;
13         uint256 co2;
14         uint256 humidity;
15         int256 temp;
16     }
17
```

[3] https://github.com/juancanobenito/Semantic-BD-Blockchain

```
18      mapping(address => sensor) public deviceInfo;
19
20      event MyEvent(string context, address identifier, string buildingName,
            string location, string office, uint256 timestamp, uint256 lux,
            uint256 co2, uint256 humidity, int256 temp);
21
22      function storeDeviceStatus(string memory context, address identifier,
            string memory buildingName, string memory location, string memory
            office, uint256 timestamp, uint256 lux, uint256 co2, uint256
            humidity, int256 temp) public {
23          deviceInfo[identifier] = sensor(context, identifier, buildingName,
                location, office, timestamp, lux, co2, humidity, temp);
24          emit MyEvent(context, identifier, buildingName, location, office,
                timestamp, lux, co2, humidity, temp);
25      }
26  }
```

Listing 5.6: Sensor contract in Solidity

The fields of the contract are:

- **@context**: The URL of the ontology context, which will be used to generate the JSON-LD.
- **Identifier**: Device address. This address will be assigned according to the address provided by the Ethereum blockchain (represented in the JSON-LD with "hasId").
- **Building name**: Name of the building from which the information is extracted (represented in the JSON-LD with "isSpaceOf").
- **Location**: Geographic location of the building (represented in the JSON-LD with "city").
- **Office**: Office number within the building (represented in the JSON-LD with "hasSpace").
- **Timestamp**: Date of data publication (represented in the JSON-LD with "timestamp"). As all data is sent at the same time, it will only need to be stored once.
- **Lux**: Unit that measures the illumination level. It is indicated in Lux (represented in the JSON-LD with "value" inside the "lightbulb" type).
- **CO2**: Carbon dioxide gas meter. It is measured in PPM (represented in the JSON-LD with "value" inside the "co2Sensor" type).
- **Humidity**: Ratio of vapour to total mass in the room. It is measured in RH (represented in the JSON-LD with "value" inside the "humiditySensor" type).
- **Temp**: The air temperature. It is measured in Fahrenheit or Celsius, although the data obtained in this experiment always appear in Celsius (represented in the JSON-LD with "value" inside the "thermometer" type).

To perform the experiments, 80,000 smart contracts and 80,000 transactions (the smart contract and the transactions have the same information in every iteration) were stored.

The data were measured for each set of 10,000 contracts/transactions to see if there were significant differences between each set.

5.3.5 Experimental analysis

Analysis of gas consumption between smart contracts and transactions

A total of 80,000 transactions have been introduced. This experiment compares in terms of gas the transactions and the smart contracts stored in Ethereum. The transactions and smart contracts store the same information. Table 5.2 shows, on the one hand, the average gas consumed storing all the data generated by the *IoT gateway* by transactions and, on the other hand, the gas consumed storing all the data generated by the *IoT gateway* through smart contracts.

	Transaction	**Smart contract**
Gas consumed	25,646	38,059

Table 5.2: Gas consumed per transaction and smart contract.

As it can be observed, storing smart contracts has a higher cost in terms of gas than a transaction. This is because the contract has an array, and storing data in the array is very expensive [41]. However, the array used in the contract has the advantage of providing the last state of the device directly, unlike in the case of storing JSON-LD directly in the block, since in this case the whole chain must be browsed to find the last state.

Analysis of time to read and write data between smart contracts and transactions

In this subsection, an analysis of the time required to write and read transactions and smart contracts from the blockchain is provided. For this purpose, a measurement was made every 10,000 iterations.

This experiment exposes the time to write transactions and smart contracts, detailed in Figure 5.10. And the differences to read transactions and smart contracts recorded in the blockchain are detailed in Figure 5.11.

With the data obtained from writing transactions (Figure 5.10) the graph shows better results in the case of transactions. However, since the time is in milliseconds, a statistical test was performed to establish if the results have statistically significant differences, which is the Iman-Davenport test [74], explained in subsection 5.2.5. The Iman-Davenport test outputs a p-value of 0.0017; therefore, it can be concluded that there are statistical differences, and thus using transactions in this context is better than smart contracts.

```
1 SELECT * WHERE {
2     ?s ?p ?o .
3 }LIMIT 80000
```

Listing 5.7: SPARQL query for the last measurement

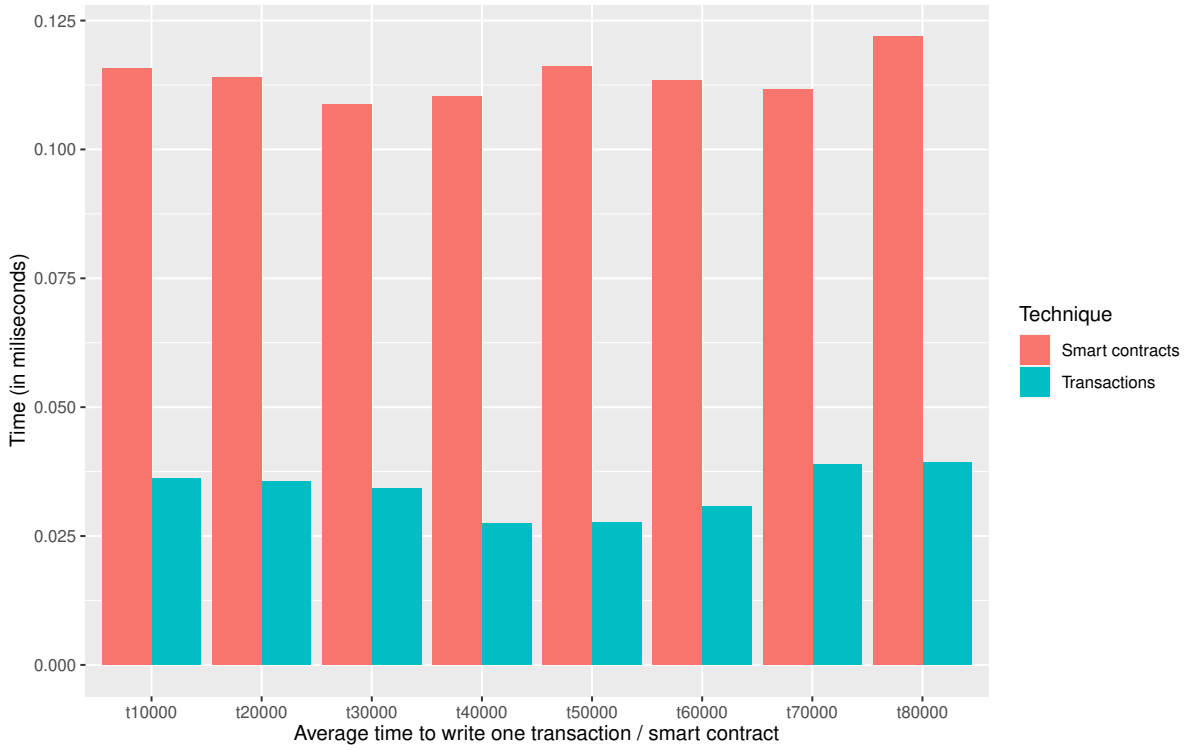

Figure 5.10: Average time to write one transaction / smart contract

The time to read all the data in the blockchain is depicted in Figure 5.11. The time difference is due to the fact that while reading the data stored in transactions requires going block by block and reading all the transactions, Ethereum incorporates a mechanism to read smart contracts more easily without having to retrieve the information from the entire chain. Meanwhile, the time to read all the transactions is 441.54 seconds, and the time to read all the smart contracts is 4.18 seconds. Table 5.3 shows the time differences in reading the whole chain. An example of a query can be found in listing 5.7.

	Transaction	**Smart Contract**
Read all the chain	441.54 seconds	4.18 seconds

Table 5.3: Time needed to read all the transactions.

5.3.6 Discussion

This chapter empirically analyses different methods to store RDF into the blockchain. As demonstrated, storing data on the blockchain without using smart contracts is highly inefficient, as it leads to longer query time, requires higher cost in terms of gas and there is no significant difference in writing time. The experimental results led to the following answers.

RQ1: in light of the reported results, it can be concluded that storing data through smart contracts may be more expensive than through transactions. However, this cost for smart

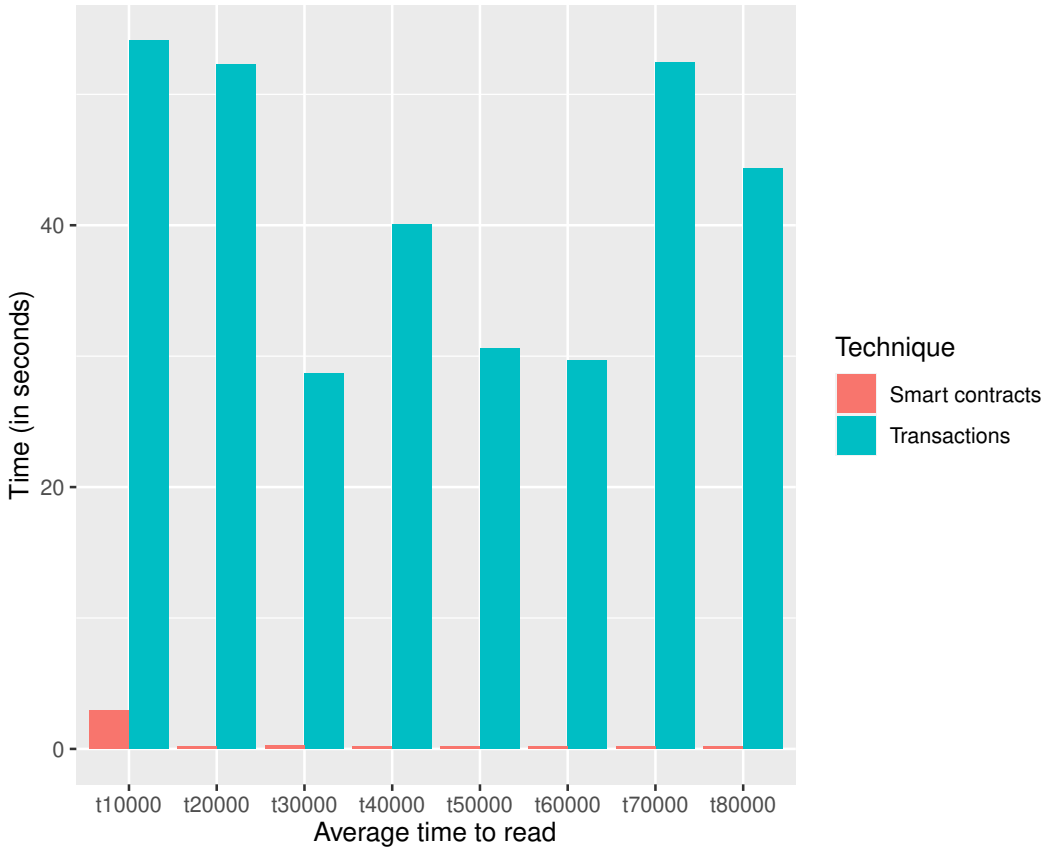

Figure 5.11: Average time to read one transaction / smart contract

contracts is subject to the code used and the information that is stored within the blockchain (as already mentioned, storing arrays within the blockchain is very expensive).

RQ2: the reported results demonstrate that writing JSON-LD via transactions is faster than writing JSON-LD via smart contracts.

RQ3: The results obtained in this experiment show that reading data from smart contracts is much faster than via transactions.

In conclusion, we can deduce: First, while storing data via smart contracts may involve a higher cost compared to transactions, this cost is influenced by the specific code used and the nature of the data being stored, with arrays, for example, being notably expensive to store on the blockchain. Second, writing JSON-LD via transactions is more efficient than doing so via smart contracts. Conversely, reading data from smart contracts is substantially faster than reading data from transactions.

Chapter 6

Building the ontological model of Ethereum

> The details are the details. They make the product. The connections, the connections, the connections. It will in the end be these details that give the product its life.
>
> *Charles Eames*

6.1 Introduction

The evaluation obtained in the previous chapter suggests that the best way to store RDF content inside a blockchain is through smart contracts. In order to make the semantic blockchain interoperable, ontologies representing the entire blockchain (which includes the smart contract languages) must be designed. Ethereum was the first blockchain that introduced the so-called smart contracts [66], computer programs that act as a contract between people or organisations, helping in the negotiation and definition of the registered agreements. The objectives of smart contracts are the reduction of the need for trusted intermediaries, arbitration and enforcement costs, and fraud losses, as well as the reduction of malicious and accidental exceptions. In blockchain, blocks validate the contracts, and when a preprogrammed condition is triggered, the smart contract executes the corresponding contractual clause. The main smart contract language in Ethereum is Solidity, an object-oriented programming language.

There are several blockchain technologies, such as Bitcoin, Ethereum, Tron or Hyperledger, and different smart contract languages developed specifically for this purpose, like Vyper, Solidity, Scilla, or Bitcoin Script [47]. Other well-known languages, such as Java or Go, work in a minority of the blockchains; however, since these well-known languages were originally not designed for smart contracts, the global state of the ledger needs to be imported through

special functions that are typically not available in these languages. Moreover, these languages that are not designed for use in smart contracts often have support for infinite loops and recursion, which are not desirable in the blockchain context. Furthermore, particular types such as assets or units also need to be defined appropriately [63].

Due to the fact that the well-known Ethereum blockchain was the first to introduce smart contracts, with its own programming language oriented exclusively to smart contracts, Solidity, it has led to Solidity being the most popular programming language to develop smart contracts in blockchain [67] and is the language most used in Ethereum [51].

Smart contracts bring several advantages; nevertheless, their execution is based on conventional procedural code, so they are not capable of solving complex problems [77]. One way to solve this drawback is using knowledge representation (a field focused on designing computer representations that capture information about the world that can be used to solve complex problems) and using ontologies to represent this knowledge.

An ontology is a formalisation of concepts and relations within a domain encompassing a representation, formal naming and definition of the categories, properties and relations between concepts, data, and entities that substantiate one, many or all domains of discourse [156]. As a result, ontologies not only introduce a shareable and reusable knowledge representation but can also add new knowledge about the domain. Some of the benefits of using an ontology are:

- **Sharing knowledge**. A user can extract and add information from different ontologies related to each other with a common term, enhancing knowledge and, thus, enriching user queries.
- **Ontology reuse**. Given an ontology, it can be reused and expanded to other domains or integrated to improve an existing ontology or other existing standards.
- **Analyse domain knowledge**. Once an ontology has been developed for a domain, the knowledge of that domain can be analysed to reuse existing ontologies and expand them.
- **Data validation**. Ontologies contain mechanisms to validate data, depending on the data and their input and output concepts. Furthermore, they can rely on W3C standards like SHACL to validate data expressed according to an ontology.
- **Semantic interoperability**. Ontologies can be integrated with other systems based on other standards. Semantic web technologies provide a solution to interoperate data semantically.

Due to the nature of ontologies and the benefits that they provide, smart contracts can take advantage of the properties of ontologies and expand them in aspects such as:

- **Code generation**. Programming code can be generated using knowledge representation through an ontology.
- **Code validation**. If a contract is aligned to a domain and it has an ontology, ontology restrictions can be used to validate the correct alignment, imposing language restrictions, and avoiding inaccuracies.

- **Interoperability**. The representation of knowledge of another similar programming language focused on smart contracts can lead to the development of frameworks to make translations between contracts possible.
- **Linked Data**. Smart contracts can extract information from other real-world domains, enhancing the contract by adding new knowledge.

6.2 The Ethereum ontology

All the ontologies presented in this chapter were built following the LOT (Linked Open Terms) methodology [131]. This methodology, based on agile techniques, is composed of a four-stage workflow: requirement specification, implementation, publication, and maintenance.

- **Ontology requirements specification.** To capture the requirements, the different versions of Ethereum and Solidity were studied. For Ethereum, the latest version was chosen, known as Ethereum 2.0[1]. For Solidity, the latest completed and revised version was chosen, which corresponds at the time of writing this chapter to version 0.8.0[2], which supports contracts developed in version 0.8.0 or below. Developed contracts supporting these versions were also collected and studied.
- **Ontology implementation.** This phase was split into three sub-tasks. Firstly, a conceptualisation of the basic concepts of Solidity was extracted, and the relationships between these concepts were identified. These concepts were taken from the Ethereum and Solidity documentation, and the subsequent study of smart contracts. Secondly, in the encoding step, a model was generated in the OWL language from the ontological model using Protégé. Finally, a group of experts supervised the ontology to check that it has no syntactic, modelling, or semantic errors and complies with all the requirements captured in the previous phase.
- **Ontology publication.** The ontologies are published in a GitHub repository and a webpage[3][4][5]. The ontologies include the code in OWL, a human-friendly documentation with a description of the classes, properties and data properties, and a graphical representation of the ontology.
- **Ontology maintenance.** The ontologies will be updated to correct possible errors or implement new requirements that will be incorporated in future versions of Ethereum and Solidity.

Figure 6.1 depictes an overview of the Ethereum ontology. The ontology is centred on the Ethereum blockchain. This ontology is connected with the Application Binary Interface (ABI) ontology, and this one, is connected to the Solidity ontology. The purpose is to capture the overview and the main classes and relationships of the Ethereum ontology.

[1] https://ethereum.org/en/roadmap/
[2] https://docs.soliditylang.org/en/v0.8.0
[3] https://w3id.org/def/Ethereum
[4] https://w3id.org/def/SolidityABI
[5] https://w3id.org/def/SolidityOntology

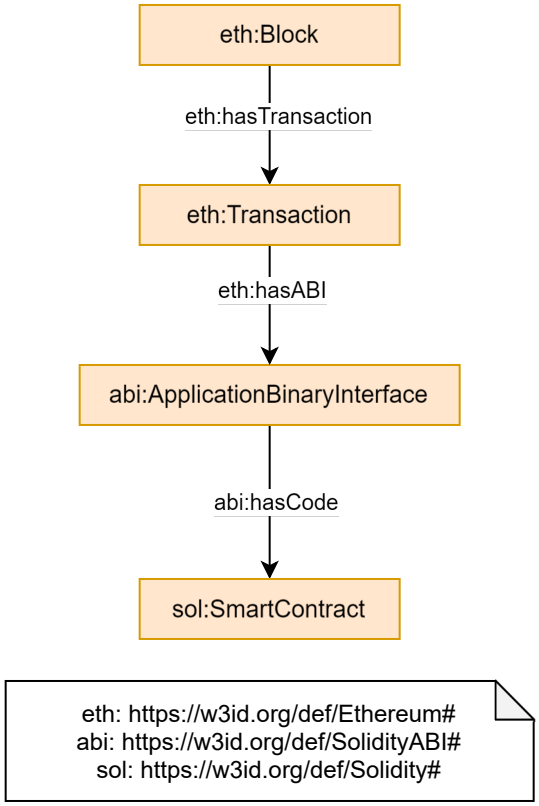

Figure 6.1: Ethereum ontology overview

The primary classes and name-spaces are represented by eth, abi and sol. The eth namespace represents the Ethereum block ontology, explained in section 6.3; the abi namespace represents the Application Binary Interface ontology, explained in section 6.4; and finally, the sol namespace symbolises the Solidity programming language ontology, explained in section 6.5.

The eth:Block class represents a block within the Ethereum blockchain. Each block in Ethereum can encapsulate multiple transactions. The eth:hasTransaction property shows that an Ethereum block may own multiple transactions. This relation binds the eth:Block class to eth:Transaction, representing an individual transaction within the Ethereum network. A transaction may have associated some ABI data, represented by the eth:hasABI property. This relation connects the transaction to its respective Application Binary Interface. The abi:ApplicationBinaryInterface class represents the main class of the ABI ontology, which describes the interaction interface for the contract. A key aspect is the associated contract implementation code, represented by the abi:hasCode property. This indicates the connection between the ABI and its underlying code, specifically the contract implementation in Solidity. The sol:contract is the main class of the Solidity ontology to represent the contract code in the Solidity programming language, which encapsulates the logic and functions of the contract.

The Ethereum Block, the ABI, and the Solidity ontology are related to other ontologies. These ontologies and their elements are identified using Internationalized Resource Identifiers (IRIs). For the sake of simplicity, in the rest of this chapter, these IRIs will be referenced by using the prefixes defined in Listing 6.1.

```
@PREFIX sol: <https://w3id.org/def/Solidity#>
@PREFIX abi: <https://w3id.org/def/SolidityABI#>
@PREFIX eth: <https://w3id.org/def/Ethereum#>
@PREFIX xsd: <https://www.w3.org/2001/XMLSchema#>
```

Listing 6.1: Predefined namespace prefixes

6.3 The Ethereum block ontology

At the core of Ethereum technology are the blocks, which are the main element of the blockchain and guarantee its integrity, transparency, and functionality. Each block records recent transactions and is linked in a linear chronological order, forming what is known as a blockchain.

The header of an Ethereum block contains metadata about the block. This includes the hash of the previous block, which cryptographically secures the link between consecutive blocks. In addition, an Ethereum block contains a unique identifier known as the block hash. Within the block, the data are stored, and individual transactions (smart contracts or transactions themselves) are processed.

Figure 6.2 depicts the Ethereum block ontology. The presented ontology provides a complete structural overview of the components of an Ethereum blockchain, focusing on the main classes and attributes of a block.

A central element of this ontology is the Ethereum block, denoted eth:Block class, which captures all the attributes that detail various aspects of a block. For example, it includes metadata such as the author of the block (eth:author class), the difficulty of the block (eth:difficulty class) and the cumulative gas used in the block (eth:gasUsed class). In addition, cryptographic attributes such as the hash of the block (eth:hash class), the parent hash (eth:parentHash class) and the root state (eth:stateRoot class) are vital to ensure the integrity of the block and its position within the blockchain. These attributes, among others, collectively provide a detailed snapshot of the specifications of the block and its state within the Ethereum network.

Related to the eth:Block class is the eth:Transaction class, and this relationship is carried out through the eth:hasTransaction property. The eth:Transaction class captures details about individual transactions, including the hash of the associated block (eth:blockHash class), the index of the transaction (eth:transactionIndex class), and the addresses of the sender and receiver (eth:from and eth:to classes, respectively).

A transaction may contain a smart contract. The application binary interface (ABI) serves as an interface between smart contracts and the applications that invoke them. The ontology is related to another ontology representing ABIs through the eth:hasABI property.

The block stores the withdrawals produced in the block, represented by eth:Withdrawal. This component defines attributes related to withdrawals, such as the associated address (eth:address) and the amount of the withdrawal (eth:amount).

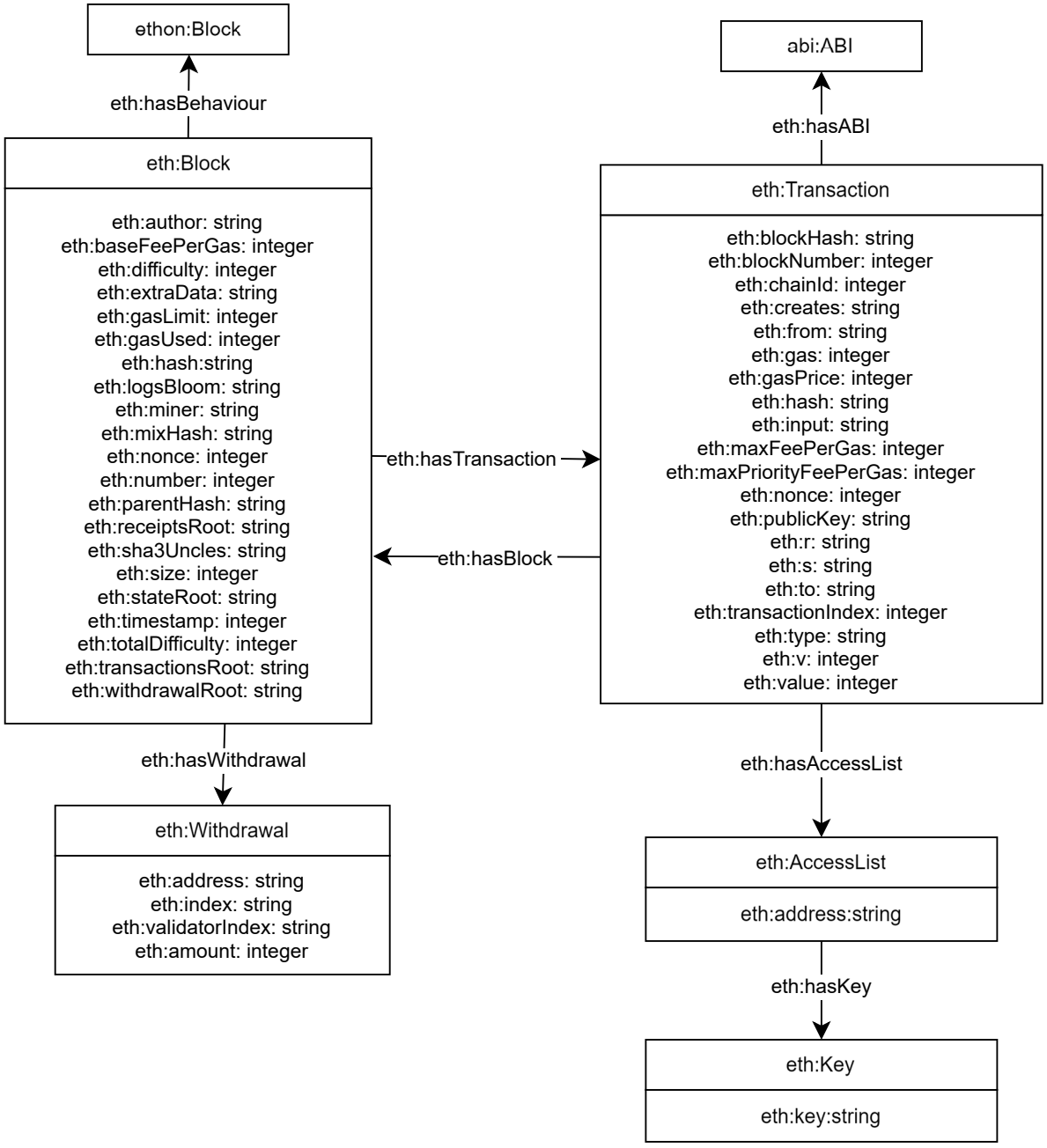

Figure 6.2: Ethereum block ontology

Finally, the access control system is shown through eth:AccessList and eth:Key. This system outlines the permissions and cryptographic keys required for specific actions and data accesses within the Ethereum ecosystem.

6.4 The Application Binary Interface ontology

Each smart contract stores its Application Binary Interface in Ethereum, known as ABI. The ABI serves as a bridge or translator. Smart contracts are written in high-level programming languages, such as Solidity. However, once these contracts are deployed on the Ethereum blockchain, they are converted into bytecode, a form that is not directly understandable by external applications. The main function of the ABI is to provide a structured way for applications to understand and interact with this bytecode. In addition, the ABI guarantees the integrity of the interactions. Specifying the exact structure of the data ensures that any information sent or received from a smart contract adheres to predetermined formats. This standardisation eliminates ambiguity, ensuring that smart contracts and applications can predictably interpret the exchanged data.

Figure 6.3 depicts the Ethereum ABI ontology. The presented ontology provides a complete structural overview of the components of the ABI, focussing on its main classes and attributes.

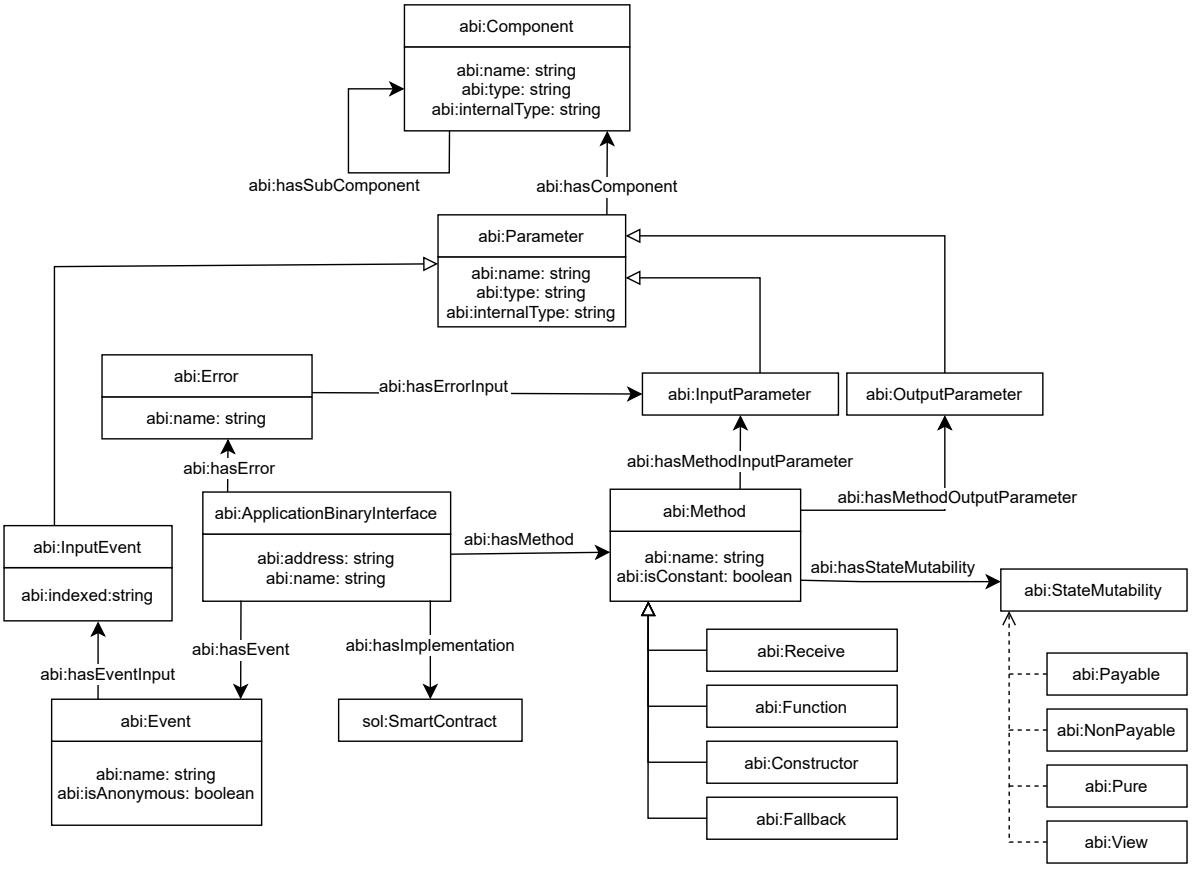

Figure 6.3: Ethereum ABI ontology

The core element of the ontology is the abi:ApplicationBinaryInterface class. This core class is linked to a multiple of functionalities that represent the capabilities of smart contracts. As a property, it has the address that represents the hash of the contract deployed, denoted by abi:address, and is associated with an actual implementation in the Solidity programming language through sol:contract class. This main class is related to other classes that define the

main elements of an ABI:

- Methods in the ABI are represented by the abi:Method class, capturing the specific functions that can be invoked within a smart contract. Each method has a name and is associated with certain properties, such as whether it is constant or its state mutability. The input and output parameters of these methods are highlighted by abi:InputParameter and abi:OutputParameter, respectively. This distinction ensures clarity about what data a method requires and what it returns after execution. Methods can be functions, constructors, fallbacks, or receives.
- Events, represented by abi:Event, store data in the Ethereum block. Events have specific properties (the event name and a Boolean flag for anonymity). These events may also have associated input parameters, illustrated by the abi:hasEventInput property.
- The ABI also handles errors through abi:Error, providing a mechanism for transmitting error messages or conditions back to the caller.
- Components are represented by abi:Component. These components are the representation of hashmaps (a data structure that is used to store and retrieve keys-based values). A component may encompass subcomponents, represented by the abi:hasSubComponent property.

6.5 The Solidity ontology

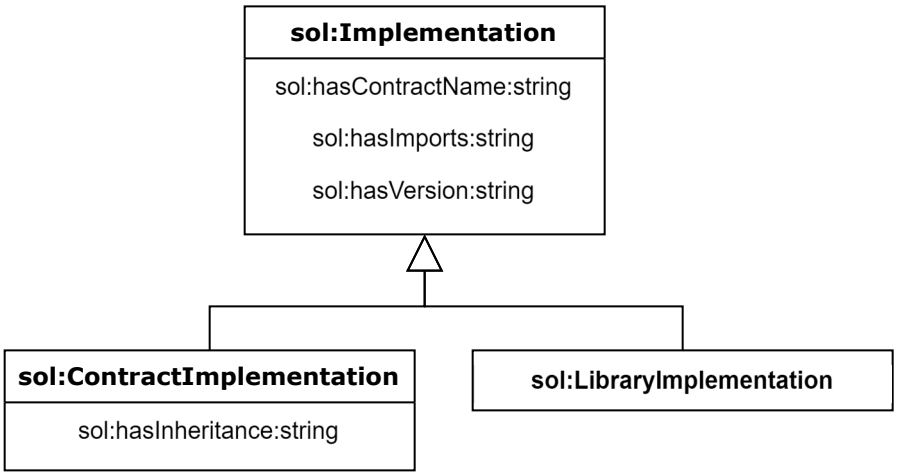

Figure 6.4: Smart contract implementations

6.5.1 Overview of a Solidity smart contract

As mentioned before, the Solidity documentation[6] has been studied to implement the ontology (version 0.8.0). Once the requirements of this version have been captured, an ontology has been developed to support all the language functions and characteristics.

[6] https://solidity.readthedocs.io/

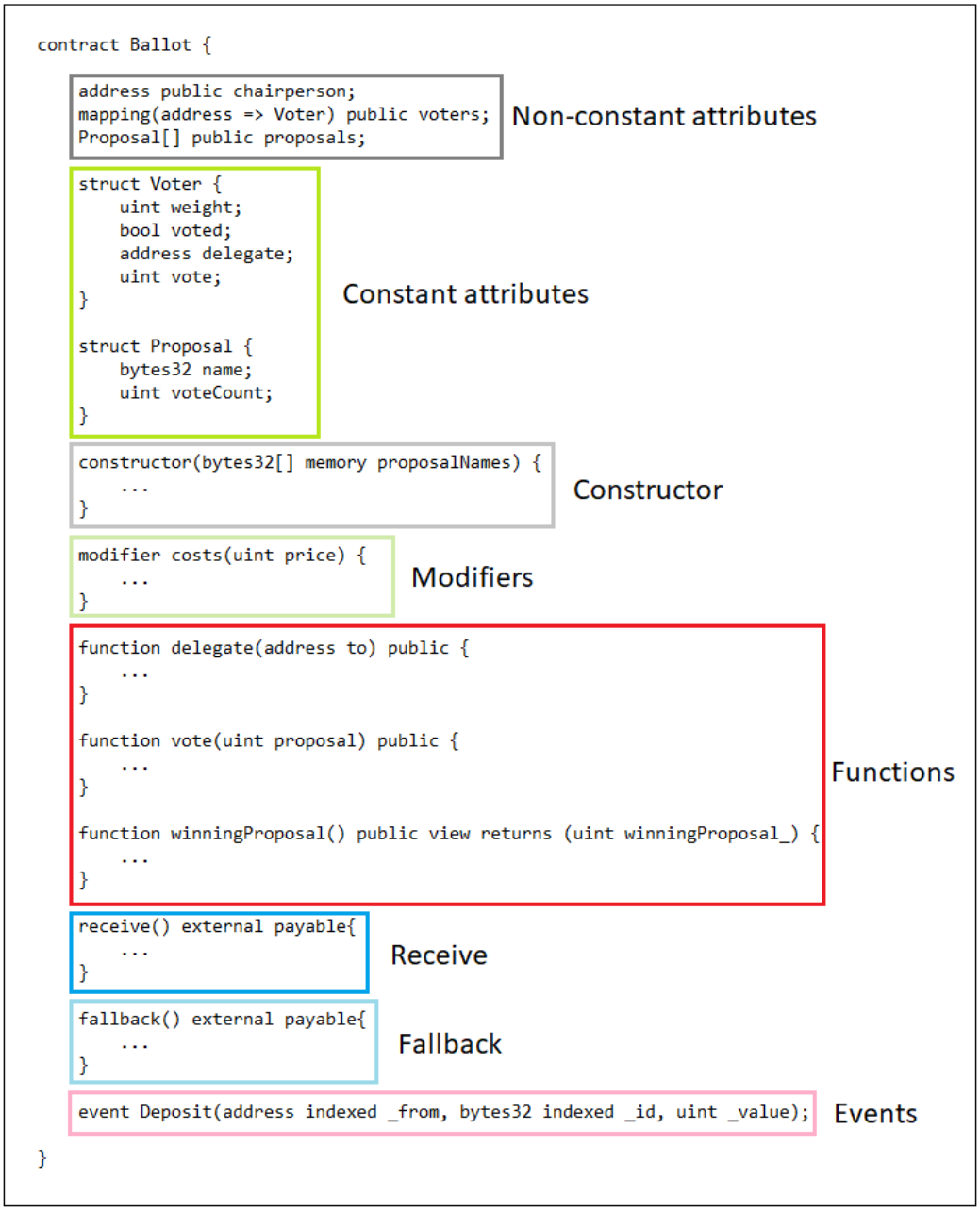

Figure 6.5: Solidity contract example

As depicted in Figure 6.4, there are two types of smart contract implementations: contracts, which are a basic building block of a smart contract, and libraries, which are similar to contracts, but their purpose is that they are deployed only once at a specific address (an Ethereum account) with its code being reused by various contracts.

Figure 6.5 depicts a smart contract example with its structure. Most elements of a Solidity contract are similar to those in the Java language; however, Solidity is an object-oriented programming language designed for developing smart contracts and, thus, contains certain elements that are characteristic of the language, such as certain new attributes (address or

struct), function types (fallback, receive, and modifier), or events, explained in the following sections.

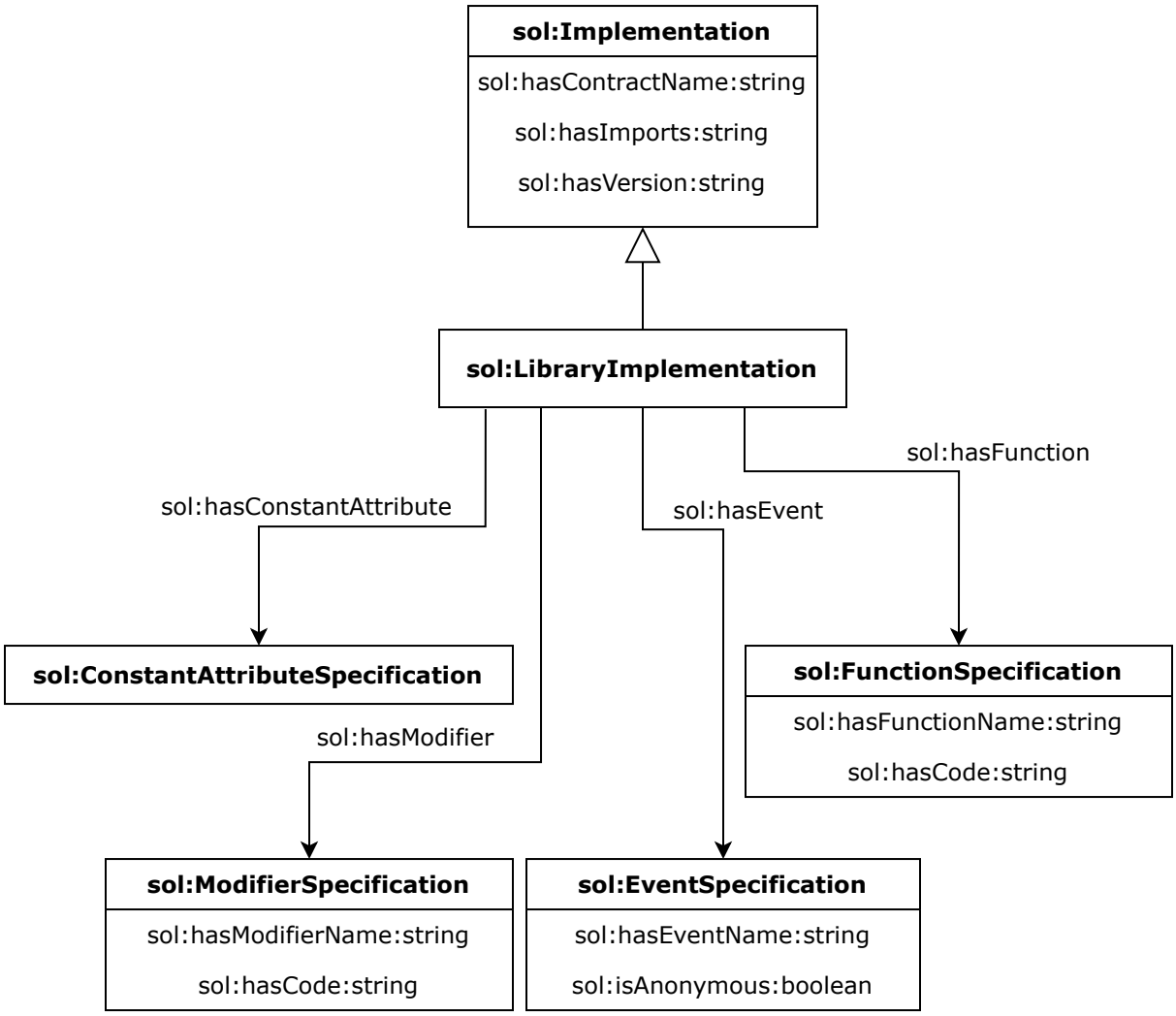

Figure 6.6: Solidity library implementation

6.5.2 Implementation of a smart contract

One of the types of implementations that Solidity supports are contracts (a computer program that includes attributes, complete functions, events and all the code that performs the logic to achieve the contract objective).

On the one hand, libraries are one type of contract. They are deployed only once at a specific address, and other contracts in the Ethereum network can reuse their properties and methods. The elements of a library are depicted in Figure 6.6.

On the other hand, the elements to represent contract implementations are depicted in Figure 6.7. The contracts can inherit from other contracts and have more classes than the library contracts, making them more complete. In a contract implementation, a library

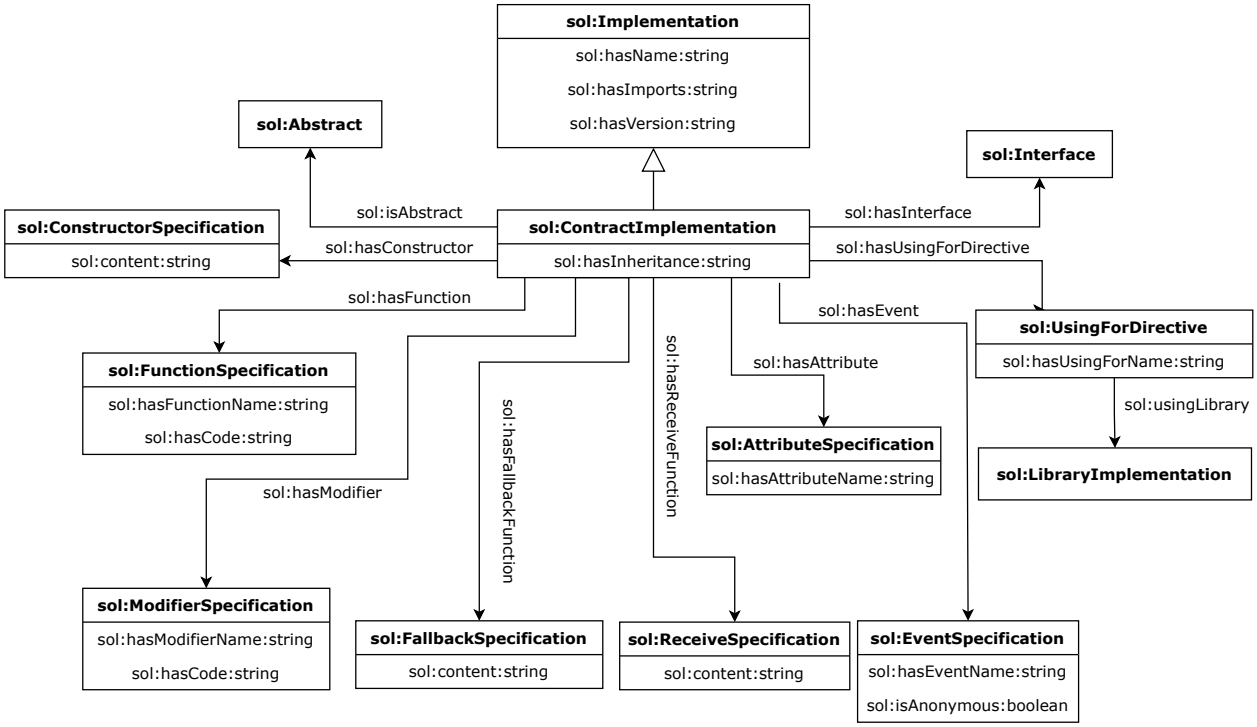

Figure 6.7: Solidity contract implementation

implementation can be used to attach library functions (from the "Library" class) to any type in the context of the contract implementation.

These contracts can be simple contracts themselves or can be the following types of contracts:

- **Interface**. A computer program that defines only the functions without code, in a way to improve the interaction between users and smart contracts, providing better accessibility, self-documentation and removing code duplication, depicted in Figure 6.8.
- **Abstract**. A contract that has all the elements of the Contract but can have unimplemented functions (functions without code), depicted in Figure 6.8.

6.5.3 Specification of contract attributes

An attribute (or variable) is part of a contract used to describe a type of data assigned to an object and, sometimes, give it a value.

In our ontology, attributes are separated into two specifications as depicted in Figure 6.9. According to the Solidity documentation[7] attributes can be classified in two ways. On the one hand, **non-constant attributes** can have a value and change it, have a visibility, can be immutable, and encompass arrays and maps. Non-constant attributes can be marked immutable, which causes them to be read-only.

On the other hand, **constant attributes** cannot have a value nor visibility. This category

[7] https://docs.soliditylang.org/en/v0.4.21/types.html

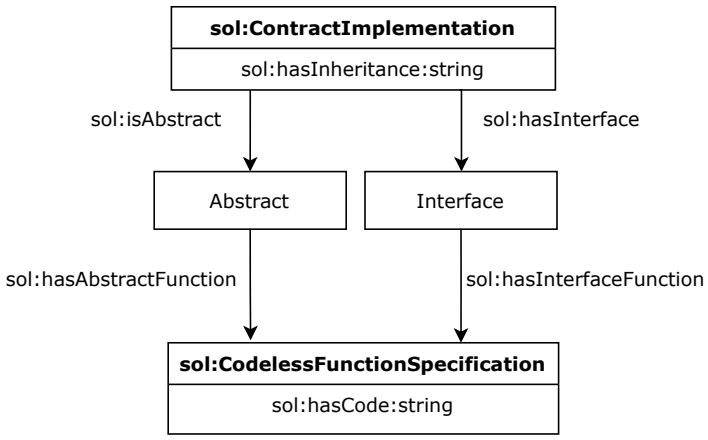

Figure 6.8: Solidity interface and abstract contract

includes enums (containing a pre-defined list of constant values) and structs (containing a group of elements with different attributes).

Non-constant attributes may have a visibility parameter. The visibility of attributes determines who can call them within the contract. In Solidity, there are four visibility types, the well-known **public** (the attribute can be called from all potential parties; unless otherwise specified, all attributes are public by default) and **private** (attributes can only be called by the main contract itself) and two Solidity-specific visibilities: **internal** (attributes can be called by the main contract itself, plus any derived contracts) and **external** (attributes can only be called from a third party; they cannot be called from the main contract itself or any contracts derived from it).

Moreover, all the attributes have a class associated indicating their properties, as depicted by Figure 6.10. First, **non-constant type** which can be contracts, array type, elementary types or memory types. Second, **constant type** which can be enum or struct types. Third, **array type** which will have a dimension with an index (that will mark the position of the dimension and a length (that will mark the value of the size); arrays can be of any type. Finally, **map key type**, which are abstract data types composed of a collection of key_value pairs, such that each possible key appears at most once in the collection; meanwhile, key must be a map key type (which contains contracts, enums, elementary types and memory types), and value can be any type. Some of these types inherit from a parent class called "Type" (map key type, non-constant type and constant type). Thus, an array can have any of these elements.

Certain types are common to other programming languages; however, Solidity contains unique elements, such as address, struct, enum, fixed and ufixed attributes. These elements can be encompassed in classes. In our ontology, we define the well-known type **elementary type** which includes address (holds a 20-byte value, equivalent to the size of an Ethereum address), boolean (the possible values are the constants true and false) and string (represents UTF-8 character strings). An example of an address attribute declaration is presented by listing 6.2 and the ontology representation of this code is depicted by listing 6.3.

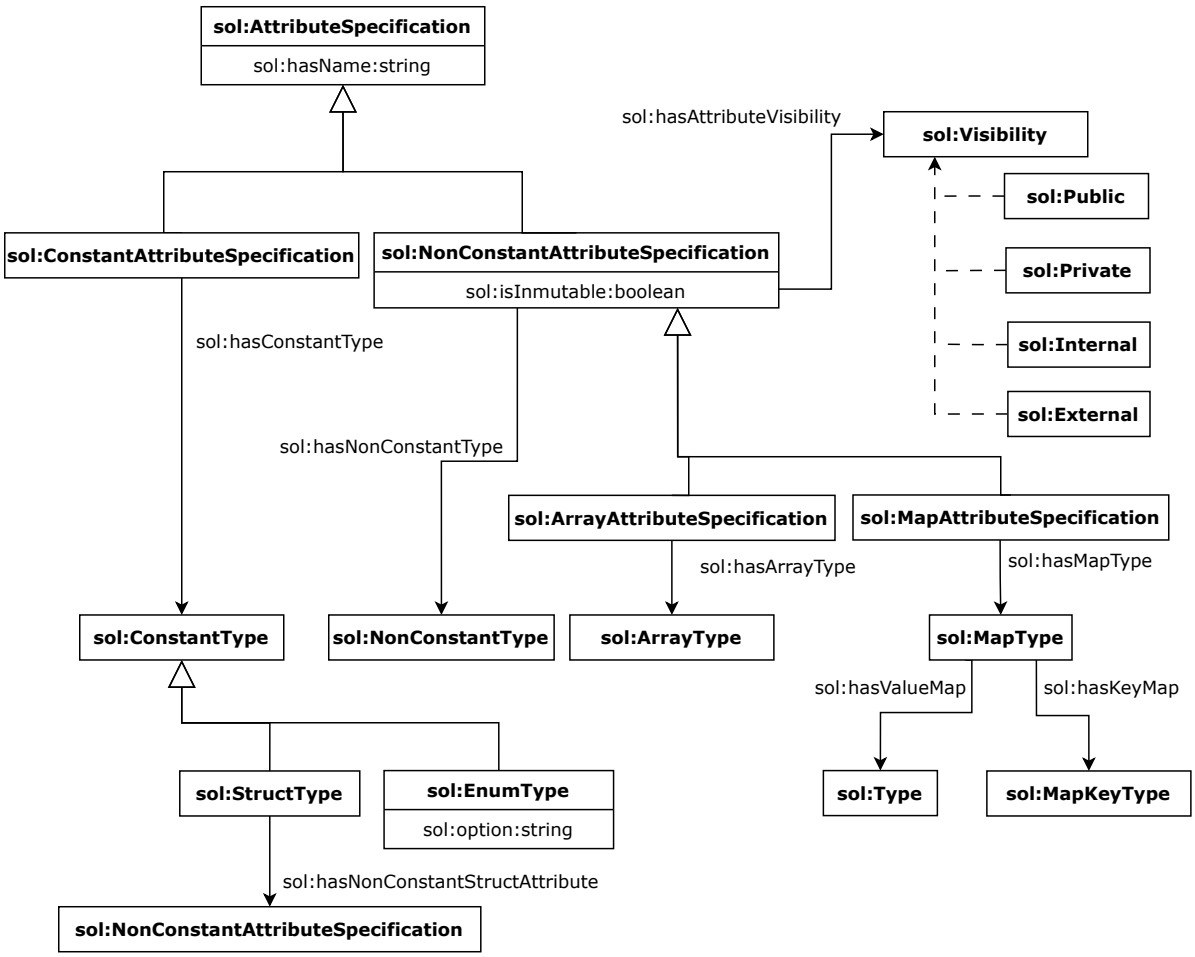

Figure 6.9: Attribute specification

```
address public chairperson;
```

Listing 6.2: Address attribute in Solidity

```
AddressChairP a sol:NonConstantAttributeSpecification;
    sol:hasAttributeVisibility sol:Public;
    sol:hasName "chairperson"^^xsd:string;
    sol:hasNonConstantType sol:Address.
```

Listing 6.3: Address attribute representation in the ontology

Another well-known type is the **memory type**, which can have single or dual memory. Single memory types are byte (a dynamically-sized byte array), int (signed integers of various sizes, represented by a number that can be written without a fractional component) and uint (unsigned integers of various sizes). Dual memory types are fixedMxN (signed fixed-point number of various sizes; M represents the number of bits taken by the type and N represents how many decimal points are available) and ufixedMxN (unsigned fixed-point number of various sizes; M represents the number of bits taken by the type and N represents how many decimal points are available). An example of a uint attribute declaration is depicted by

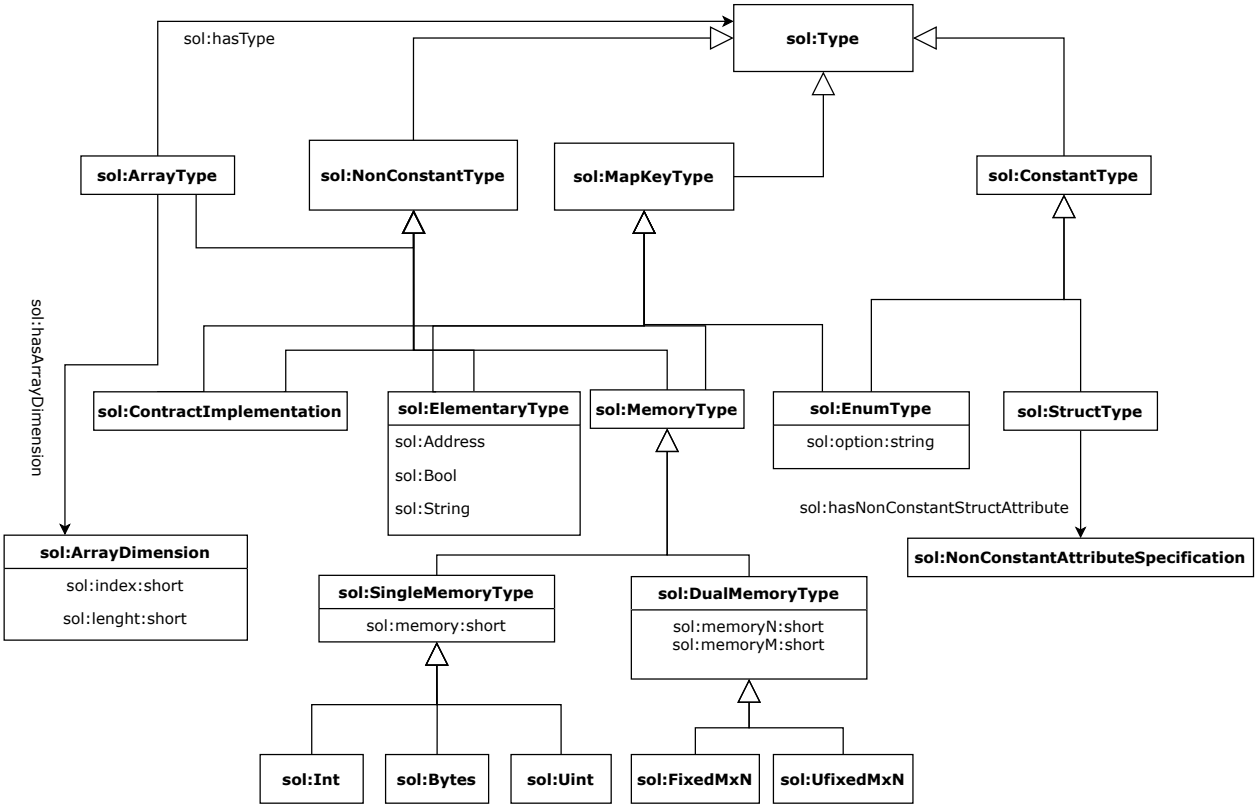

Figure 6.10: Solidity types

listing 6.4 and the ontology representation of this code is depicted by listing 6.5.

```
int public varInt;
```

Listing 6.4: Integer attribute in Solidity

```
Int256 a sol:NonConstantAttributeSpecification;
    sol:hasAttributeVisibility sol:Public;
    sol:hasName "varInt"^^xsd:string;
    sol:hasNonConstantType[
        a sol:Int;
        sol:memory "256"^^xsd:short.
    ].
```

Listing 6.5: Integer attribute representation in the ontology

Moreover, a way to create a user-defined type in Solidity is using the **enum type**. They are explicitly convertible to and from all integer types, but implicit conversion is not allowed. An example of an enum attribute declaration is depicted by listing 6.6 and the ontology representation of this code is depicted by listing 6.7.

```
enum varEnum { Hello, Bye }
```

Listing 6.6: Enum attribute in Solidity

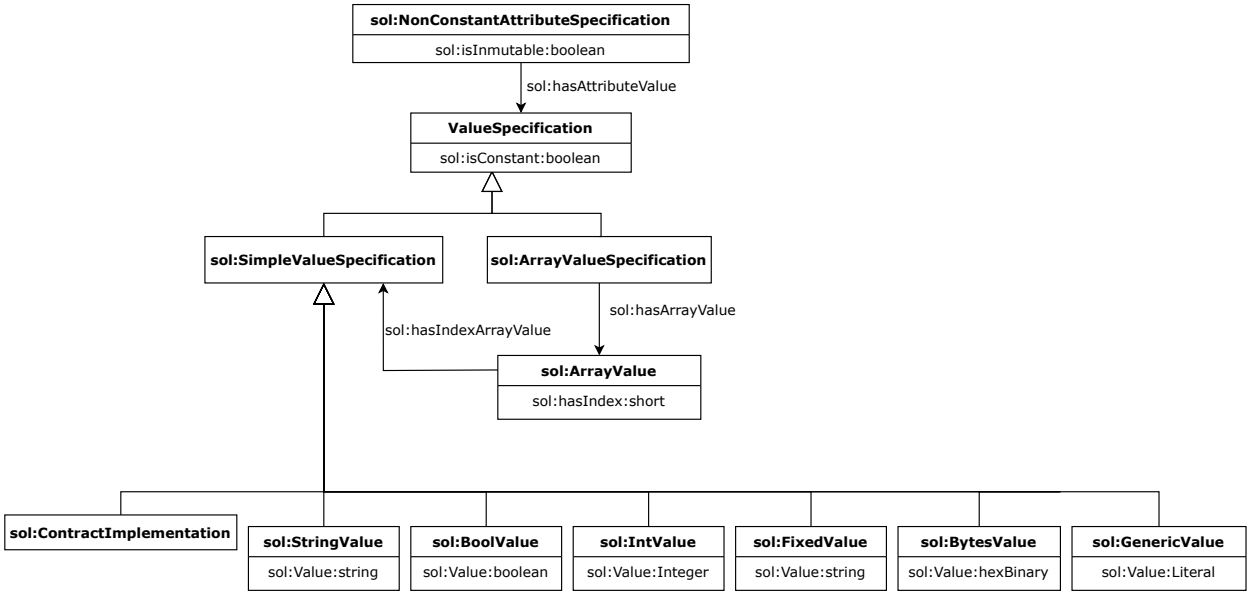

Figure 6.11: Type value specification

```
EnumEx a sol:ConstantAttributeSpecification;
    sol:hasName "varEnum"^^xsd:string;
    sol:hasConstantType[
        a sol:EnumType;
        sol:option "Hello"^^xsd:string;
        sol:option "Bye"^^xsd:string.
    ].
```

Listing 6.7: Enum attribute representation in the ontology

Furthermore, if a type is related to the **contract implementation** class, it indicates that the attribute type is another contract. An example of contract attribute declaration is depicted by listing 6.8 and the ontology representation of this code is depicted by listing 6.9.

```
ContractA public varContract;
```

Listing 6.8: Contract attribute in Solidity

```
ContractEx a sol:NonConstantAttributeSpecification;
    sol:hasVisibility sol:Public;
    sol:hasName "varContract"^^xsd:string;
    sol:hasNonConstantType sol:ContractA.
```

Listing 6.9: Contract attribute representation in the ontology

Finally, one Solidity specific type is the **struct type**, which is a custom type in Solidity. It enables to group several attributes of multiple types. Struct types can be used inside mappings and arrays, and they can themselves contain mappings and arrays. An example of a struct attribute declaration is depicted by listing 6.10 and the ontology representation of this code is depicted by listing 6.11.

```
struct Voter {
    uint age;
    bool voted;
    string name;
}
```

Listing 6.10: Struct attribute in Solidity

```
StructVoter a sol:ConstantAttributeSpecification;
    sol:hasName "Voter"^^xsd:string;
    sol:hasConstantType[
        a sol:StructType:
        sol:hasNonConstantStructAttribute[
            sol:hasName "age"^^xsd:string;
            sol:hasNonConstantType[
                a sol:Uint;
                sol:memory "256"^^xsd:short.
            ].
        ];[
            sol:hasName "voted"^^xsd:string;
            sol:hasNonConstantType sol:Bool.
        ];[
            sol:hasName "name"^^xsd:string;
            sol:hasNonConstantType sol:String.
        ].
    ].
```

Listing 6.11: Struct attribute representation in the ontology

6.5.4 Specification of attribute values

Non-constant attributes, with the exception of mapping attributes, may have values assigned to them. While simple types can only have one value assigned to them, as shown in Listing 6.12, array types can have multiple values assigned to them, with a numeric value indicating the position of the value in the array. An example of an attribute is depicted by listing 6.8 and the ontology representation of this attribute is depicted by listing 6.9.

```
string public varString = "Hello";
```

Listing 6.12: String attribute with value in Solidity

```
StringEx a sol:NonConstantAttributeSpecification;
    sol:hasAttributeVisibility sol:Public;
    sol:hasName "varString"^^xsd:string;
    sol:hasNonConstantType sol:String;
    sol:hasAttributeValue[
        sol:isConstant false^^xsd:boolean;
```

```
        a sol:StringValue;
        sol:Value "Hello"^^xsd:string.
    ].
```

Listing 6.13: String attribute representation with value in the ontology

6.5.5 Specification of constructors

Constructors in Solidity are similar to Java ones. When a contract is created, its constructor (a function declared with the constructor keyword) is executed once. This constructor is optional and, in contrast to Java, only one constructor is allowed, which means overloading is not supported. The Solidity constructor specification is depicted by Figure 6.12.

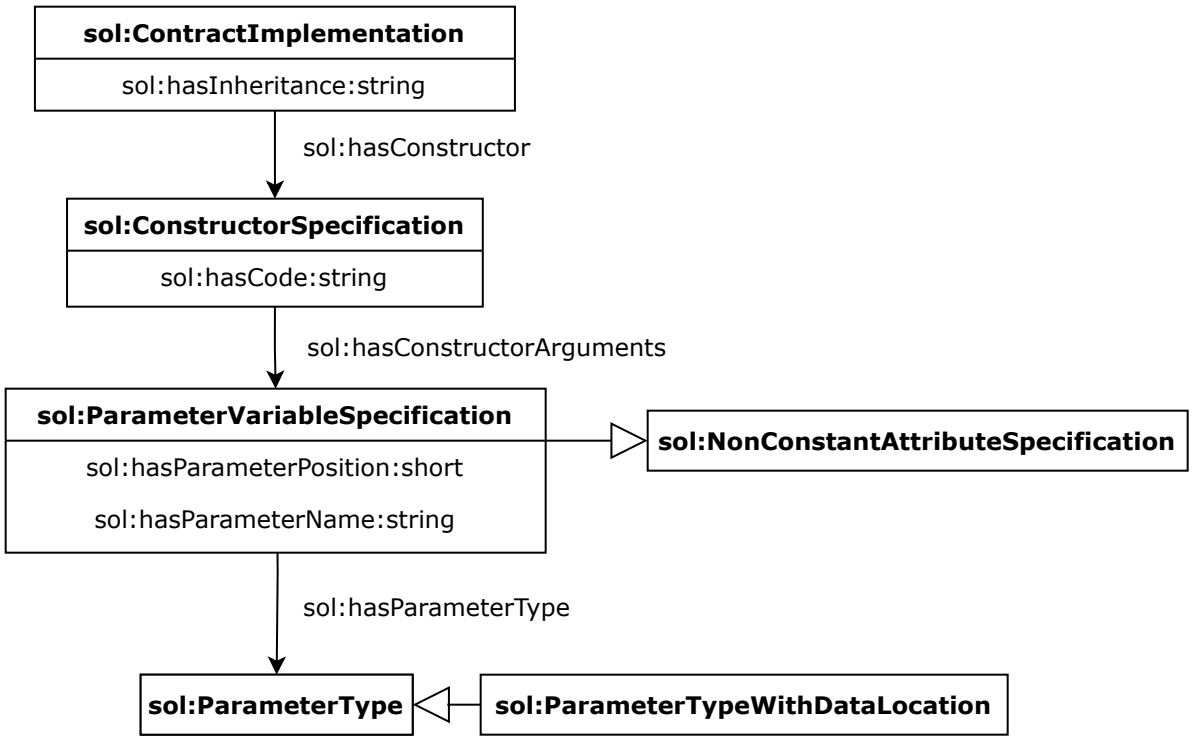

Figure 6.12: Constructor specification

Constructors have parameters (input attributes), and they are declared in the same way as attributes. The parameter types can be Type or Array Type, as depicted by Figure 6.13.

However, struct, byte, and array types can have different types of behaviour, such as **calldata**, a read-only byte-addressable space where the data parameter of a transaction or call is held; **memory**, a byte-array, which holds the data in it until the execution of the function. The memory of an array starts with zero-size and can be expanded in 32-byte chunks by simply accessing or storing memory at indices greater than its current size and, finally, **storage** which is used due to the fact that every contract has a persistent memory that is called storage. Storage is a key-value store where keys and values are both 32 bytes.

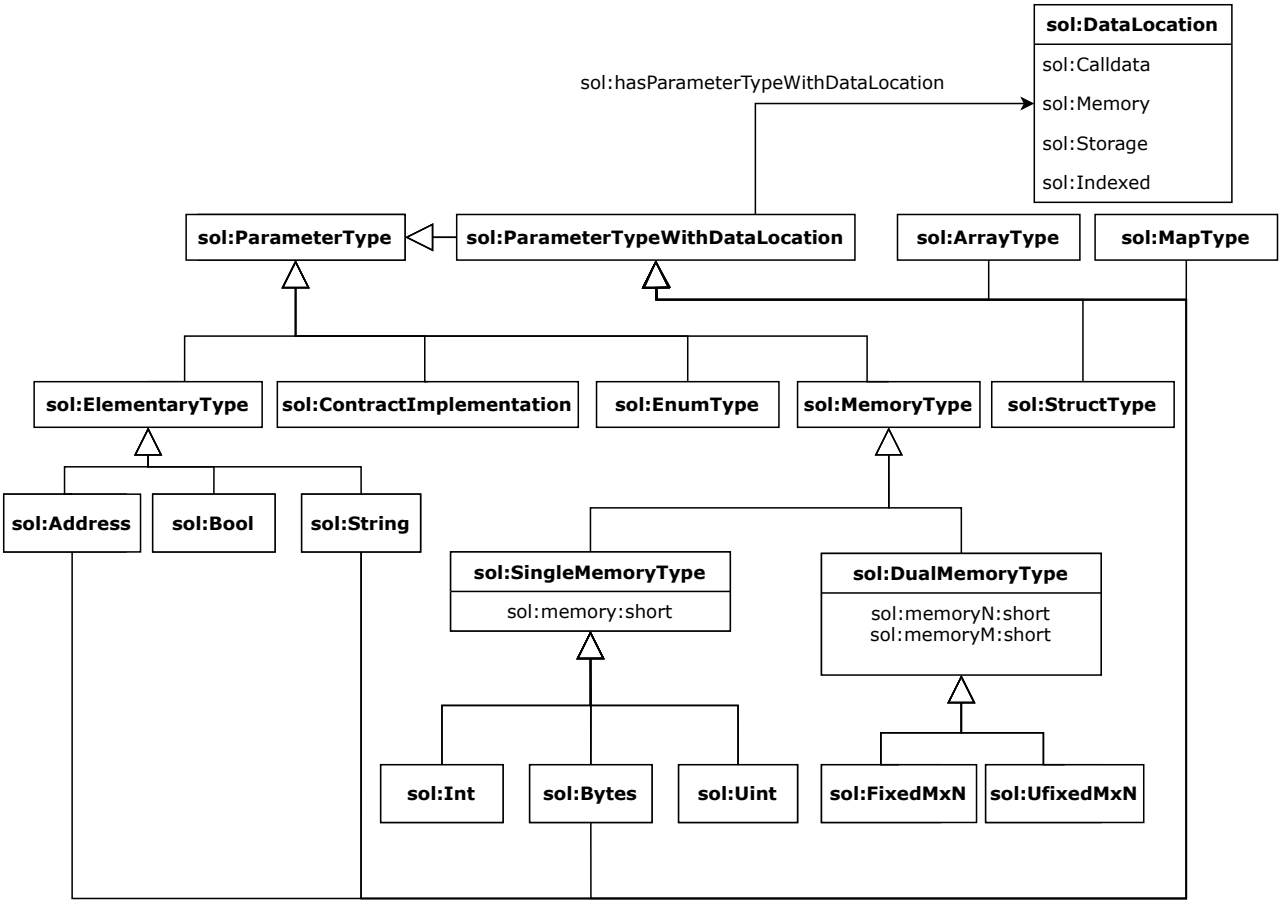

Figure 6.13: Solidity parameter types specification

An example of a constructor is depicted by listing 6.15 and the ontology representation of this constructor is depicted by listing 6.14.

```
constructor(bytes32 memory proposalNames) {
    [...]
}
```

Listing 6.14: Specification of a constructor in Solidity

```
ConstructorCon a sol:ContructorSpecification;
    sol:hasCode "..."^^xsd:string;
    sol:hasConstructorArguments[
        sol:hasParameterPosition "1"^^xsd:short;
        sol:hasParameterType[
            sol:hasParameterName "proposalNames"^^xsd:string;
            a sol:Bytes;
            sol:memory "32"^^xsd:short;
            sol:hasParameterTypeWithDataLocation sol:Memory.
        ].
```

```
11      ].
```

Listing 6.15: Constructor representation in the ontology

6.5.6 Specification of functions

A function in a programming language is a program fragment with the logic to perform a defined task. Optionally, functions take typed parameters as input and may also return an arbitrary number of values as output. Ontology representation of a function is depicted by Figure 6.14.

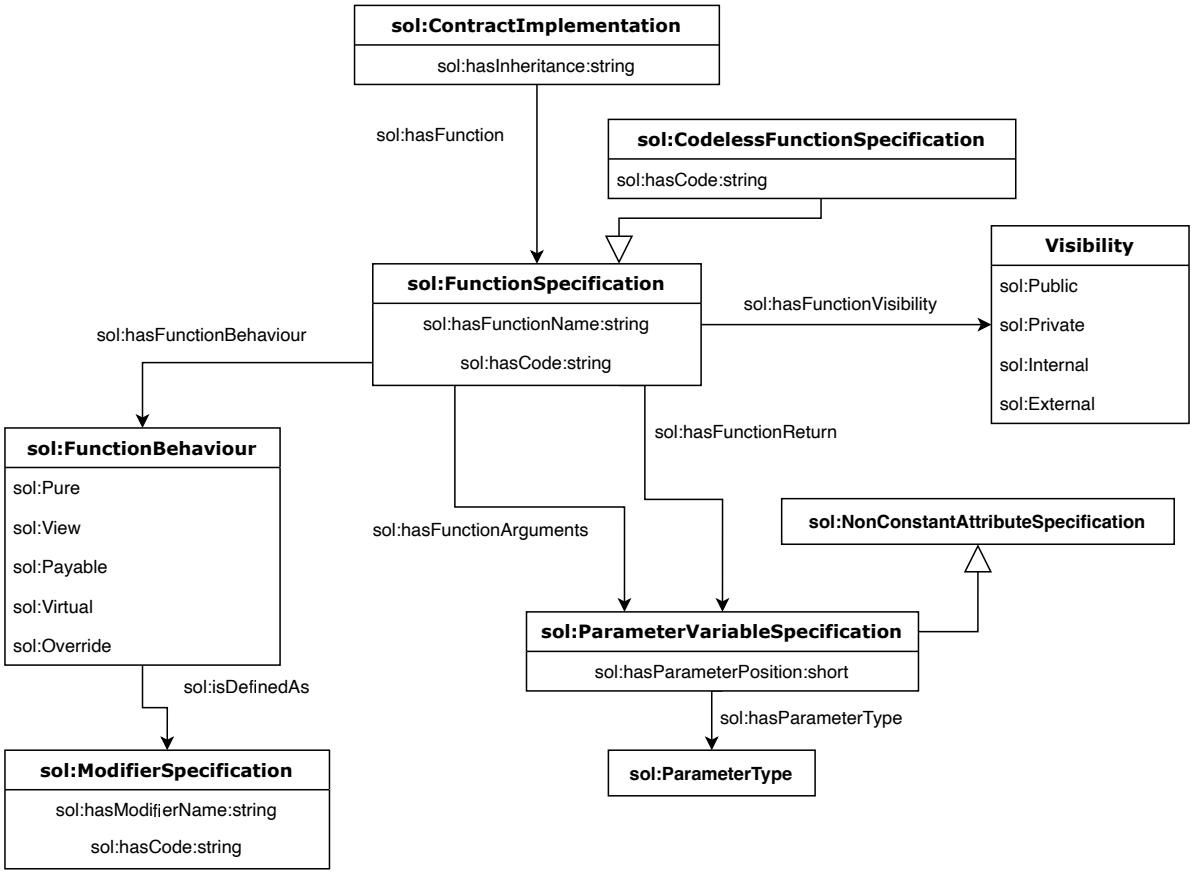

Figure 6.14: Function specification

Common to Java, functions have a name, a content and can have input and output arguments. However, in Solidity, these functions can have a modifier to set their behaviour in a declarative way. One of these behaviours is **pure**, which is used in order to avoid reading or modifying the state. Secondly, **view** to avoid modifying the state. Thirdly, the **payable** behaviour allows a function to receive payments while being called. Fourthly, the behaviour **virtual** allows an inheriting contract to override its behaviour. Fifthly, **override** is required when overriding an existing function.

Finally, a function can have the behaviour defined by the user by using **modifier functions**. The logic of this behaviour is defined in the code of the referenced modifier function. An

example of a function is depicted by listing 6.17 and the ontology representation of this function is depicted by listing 6.16.

```
function giveRightToVote(address voter) public {
    [...]
}
```

Listing 6.16: Specification of a function in Solidity

```
FunctionEx a sol:FunctionSpecification;
    sol:hasFunctionName "giveRightToVote"^^xsd:string;
    sol:hasCode "..."^^xsd:string;
    sol:hasFunctionVisibility sol:Public;
    sol:hasFunctionArguments[
        sol:hasParameterName "voter"^^xsd:string;
        sol:hasParameterPosition "1"^^xsd:short;
        sol:hasParameterType sol:Address.
    ].
```

Listing 6.17: Function representation in the ontology

6.5.7 Specification of modifiers

Modifiers (Figure 6.15) can be used to change the behaviour of functions in a declarative way. Modifiers are unique functions of the Solidity language and enable adding custom behaviours in the contract. On the one hand, the **virtual** keyword allows to change the behaviour of a function and must be used on the overridden modifier; on the other hand, the **override** keyword is similar to virtual, but it is used in case of multiple inheritances.

Modifiers can override certain data types with the override behaviour (struct, enum and contracts), as depicted by Figure 6.16. An example of a modifier is depicted by listing 6.18 and the ontology representation of this modifier is depicted by listing 6.19.

```
modifier costs(uint price) {
    [...]
}
```

Listing 6.18: Specification of a modifier in Solidity

```
ModifierCost a sol:ModifierSpecification;
    sol:hasModifierName "costs"^^xsd:string;
    sol:hasCode "..."^^xsd:string;
    sol:hasModifierArguments[
        sol:hasParameterPosition "1"^^xsd:short;
        sol:hasParameterName "price"^^xsd:string;
        sol:hasParameterType[
            a sol:Uint;
            sol:memory "256"^^xsd:short;
        ].
```

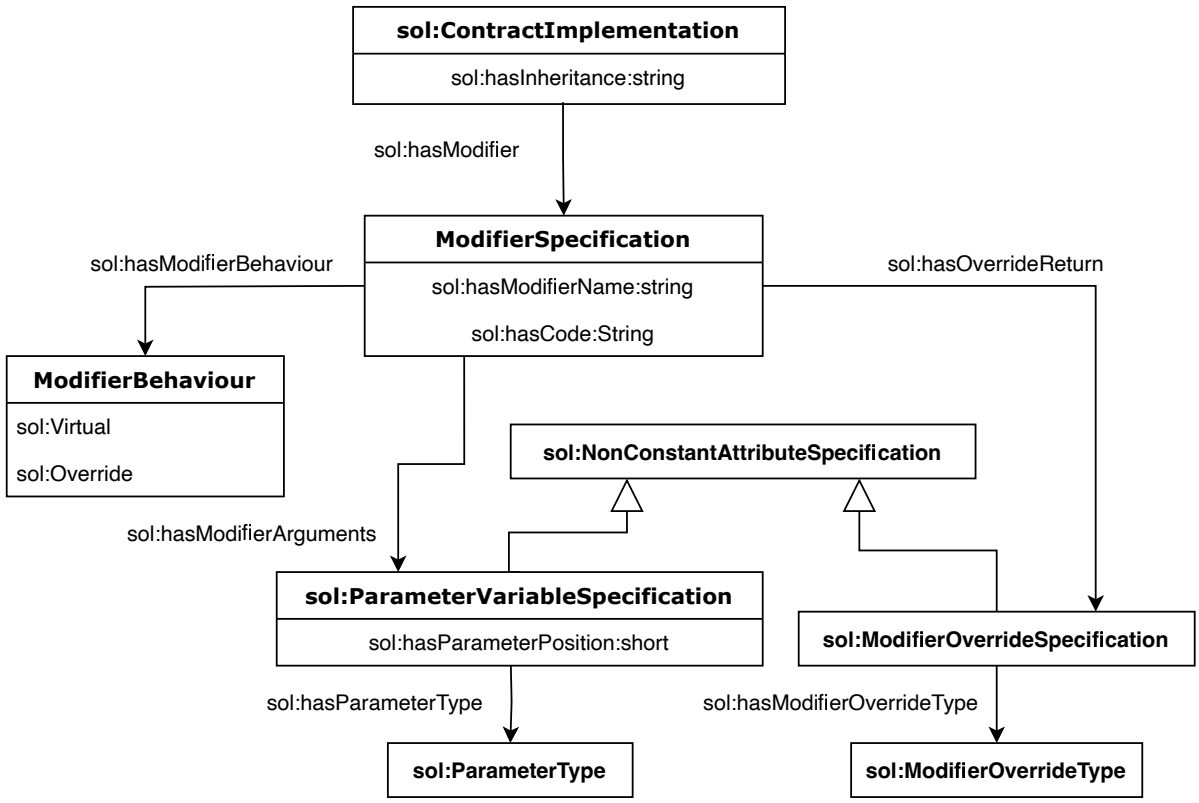

Figure 6.15: Modifier specification

```
11      ].
```

Listing 6.19: Modifier representation in the ontology

6.5.8 Specification of fallback and receive functions

A contract can have only one fallback and one receive function. On the one hand, the fallback function cannot have arguments, cannot return anything and must have external visibility. It is called when a non-existent function is called on the contract. This function always receives data, but to also receive Ether, the function must be payable. The fallback function ontology is depicted by Figure 6.17. An example of a receive is depicted by listing 6.20 and the ontology representation of this receive is depicted by listing 6.21.

```
1 receive() external payable {
2     emit Received(msg.sender, msg.value);
3 }
```

Listing 6.20: Specification of a receive in Solidity

```
1 receiveEx a sol:ReceiveSpecification;
2     sol:hasCode "emit Received[...]"^^xsd:string;
3     sol:hasReceiveBehaviour sol:Payable;
```

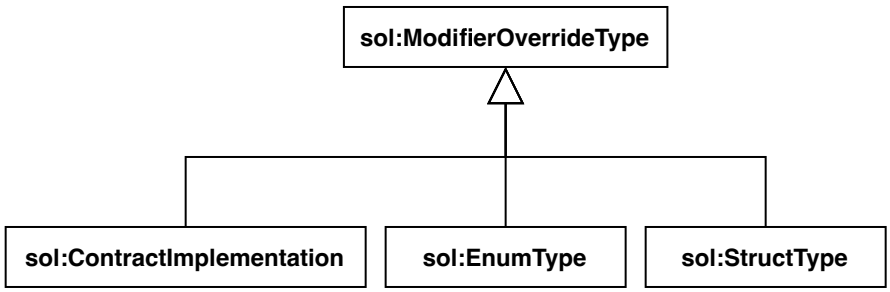

Figure 6.16: Modifier returning parameters specification

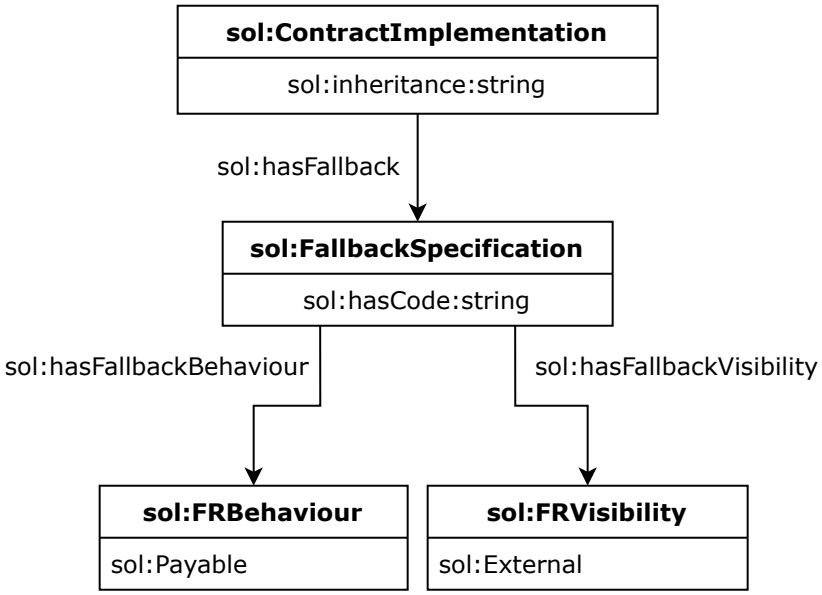

Figure 6.17: Fallback specification

```
4     sol:hasReceiveVisibility sol:External.
```

Listing 6.21: Receive representation in the ontology

On the other hand, the receive function is used as a fallback function in a contract to receive Ether. The receive function ontology is depicted by Figure 6.18. An example of a fallback is depicted by listing 6.22 and the ontology representation of this fallback is depicted by listing 6.23.

```
1 fallback() external { }
```

Listing 6.22: Specification of a fallback in Solidity

```
1 fallbackEx a sol:FallbackSpecification;
2     sol:hasFallbackType sol:Payable;
3     sol:hasVisibility sol:External.
```

Listing 6.23: Fallback representation in the ontology

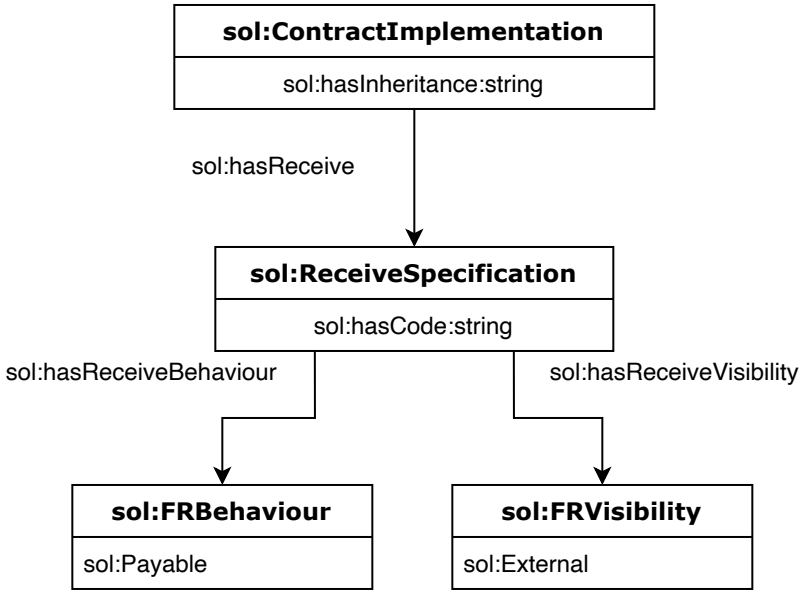

Figure 6.18: Receive specification

6.5.9 Specification of events

Events are specifically designed for smart contract languages. Events are the way to notify about the actions performed by the called function; when an event is emitted, it stores the arguments passed in transaction logs. An anonymous event will be hidden unless the original contract is available. These logs are stored into the blockchain and are accessible using the address of the contract till the contract is present on the blockchain. A generated event is not accessible from contracts within the chain, not even the one which has created and emitted them.

An example of an event is depicted by listing 6.24 and the ontology representation of this event is depicted by listing 6.25.

```
event Received(address, uint);
```

Listing 6.24: Specification of a event in Solidity

```
eventEx a sol:EventSpecification;
    sol:hasEventName "Received"^^xsd:string;
    sol:isAnonymous false^^xsd:boolean;
    sol:hasEventArguments[
        sol:hasParameterPosition "1"^^xsd:short;
        sol:hasParameterType sol:Address.
    ];[
        sol:hasParameterPosition "2"^^xsd:short;
        sol:hasParameterType[
            a sol:Uint;
            sol:memory "256"^^xsd:short;
        ].
```

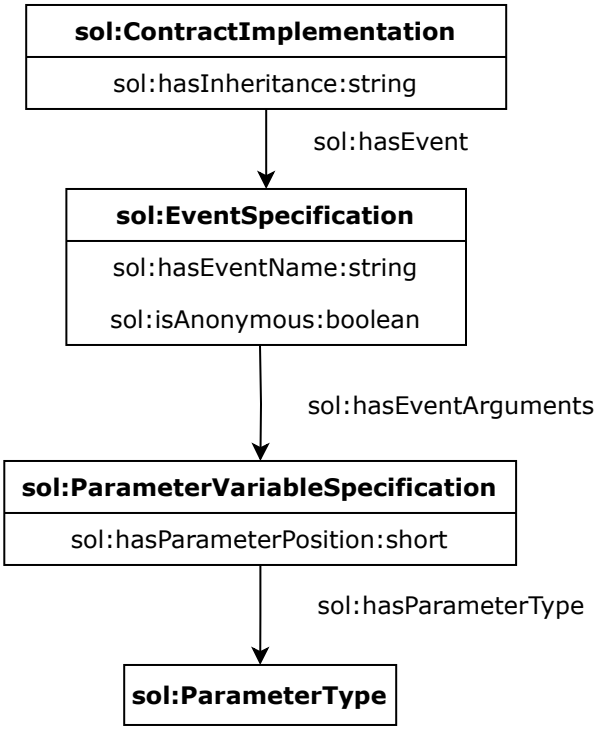

Figure 6.19: Solidity event specification

```
13      ] .
```

Listing 6.25: Event representation in the ontology

6.6 Model validation

The Ethereum block ontology covers all the block data from 2,608 existing blocks and a context was built for aligning the data to the ontology[8]. Moreover, a SHACL shape was generated using Astrea (this tool produces the SHACL shape that can be infered from one, or more, ontologies) [34] to validate the data generated by the Java tool[9].

For the ABI ontology, 8,283 ABI contracts from the 2,000 blocks previously downloaded in the Ethereum public chain have been downloaded. This ontology also covers all the ABI data and a context[10] and a SHACL[11] was generated to validate the data generated.

To verify that our ontological requirements are met in the Solidity ontology, the first step is to check that our ontology covers all the elements of a contract by studying the Solidity language and comparing the elements covered by our ontology with a dataset of contracts deployed in the Ethereum network. Then, we check that the ontology is valid by means of

[8] https://raw.githubusercontent.com/oeg-upm/Ethereum-ontology/main/context/context.json
[9] https://raw.githubusercontent.com/oeg-upm/Ethereum-ontology/main/Shapes/shapes.ttl
[10] https://raw.githubusercontent.com/oeg-upm/Solidity-ABI-ontology/main/context/context.json
[11] https://raw.githubusercontent.com/oeg-upm/Solidity-ABI-ontology/main/shapes/shapes.ttl

a reasoner and a pitfall analyser. The first step was to download contracts deployed in the main Ethereum chain from Etherscan (a block explorer and analytics platform for Ethereum), analyse the code of each downloaded contract (parsing the contract into JSON in order to make it easier to analyse) and validated our ontology by means of a script, checking that all aspects were covered or, on the contrary, if our ontology covers features of the Solidity language that are not covered by the parsed text.

Finally, all the ontologies use OOPS! [132], a tool to detect some of the most common pitfalls appearing within ontology developments and the Hermit reasoner [56] to check that it is valid.

6.6.1 Validating the ontology with deployed contracts

To validate our ontology, 3,001 unique public smart contract files previously validated by the Etherscan[12] repository have been downloaded. In order to analyse these contracts, a Python tool was used to parse contracts to JSON[13].

These smart contract files can contain one or more contracts (which can be contract, library, interface or abstract type). Contracts have been analysed against our ontology to ensure that the ontology covers them or whether our ontology covers features not supported by the parser, as depicted by Figure 6.20.

Analysing contract types

The 3,001 smart contract files contain 11,707 contracts (77,63%), 2,170 interfaces (14,39%) and 1,204 libraries (7,98%). These types are covered in our ontology; however, the parser used to convert the contract into JSON does not consider whether a contract is abstract or not, so this is an advantage for our ontology. The parser includes other common features with our ontology, such as inheritances, the contract name or the version of Solidity used.

Analysing non-constant attribute types

Non-constant attributes include arrays, mappings, other contracts and simple attributes. From the contracts analysed, 12,569 simple attributes, 2,964 mappings, 965 user-defined types (due to the fact that the parser does not distinguish between contracts, structs or enums, and all of them are categorised as "UserDefinedType", it can be any of the three; in our ontology, there is a differentiation between these three types) and 119 arrays have been located. The small number of arrays is due to the large amount of gas they consume, as it is more optimal to use mappings than arrays to store data [97].

As for the types used, contract types are not represented correctly because they have customised names. In simple types, the types listed in table 6.1 below have been used.

For mapping types, there are two values: key types and value types. On the one hand, key types only support elementary and contracts types. Analysing the contracts, it can be observed in table 6.2 that the most used type is address, while the rest are rarely used.

[12] https://etherscan.io/contractsVerified
[13] https://github.com/ConsenSys/python-solidity-parser

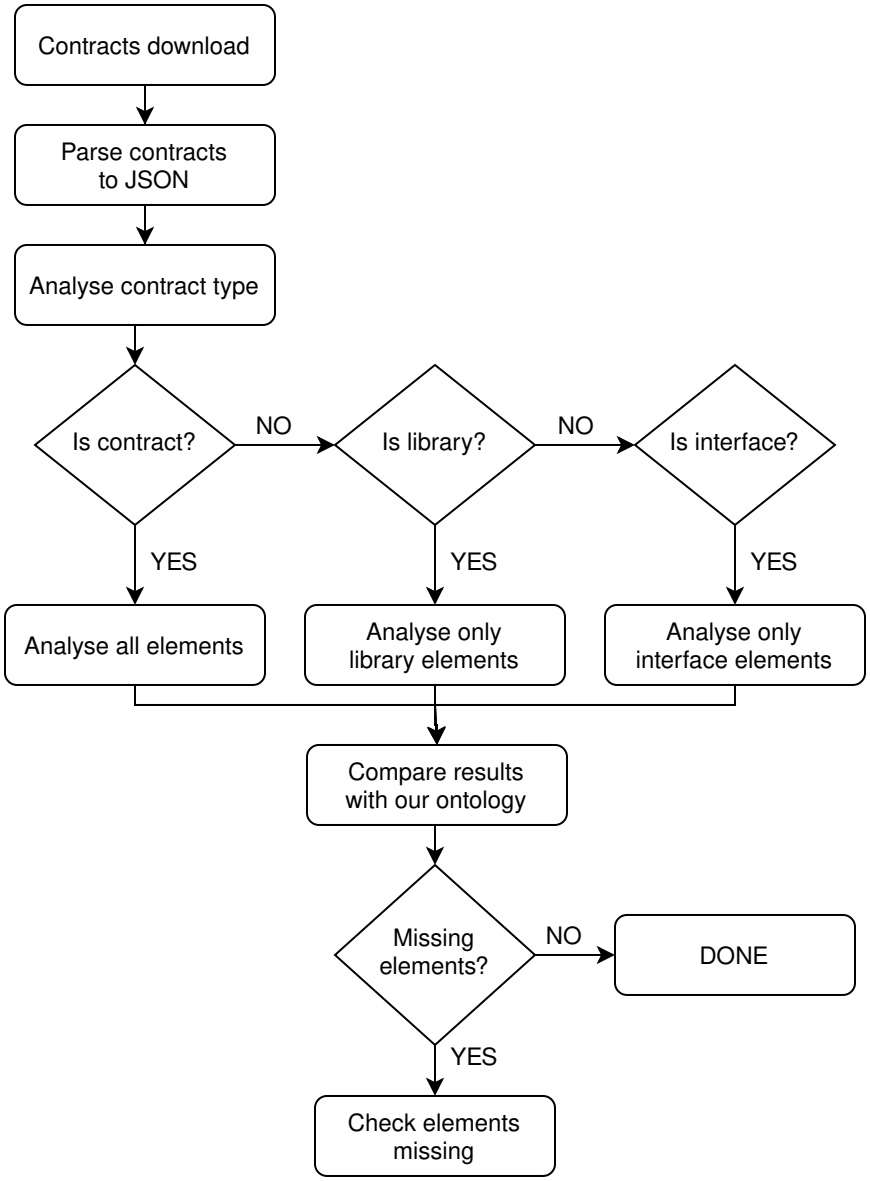

Figure 6.20: Diagram of the ontology validation process

On the other hand, value types in mapping can be user-defined types (appearing 30 times), arrays (66), simple (1,732), and other mappings (appears 1,136 times, each one with the corresponding key and value types).

The parser shows in the variables the four types of visibility (public, private, external or internal), if a variable is constant or not but, however, the parser does not indicate if a variable is immutable.

Analysing constant attribute types

Non-constant attributes include structs and enums. On the one hand, in the structures of the structs, attributes such as user-defined types (appearing 336 times), arrays (193), mappings

Elementary type	Number of appearances
address	4,003
string	3,017
uint256	1,935
uint	1,311
uint8	1,184
bool	489
bytes32	328
bytes4	116
bytes	76
uint32	25
int128	22
int256	16
uint64	13
int	11
uint128	9
uint80	6
uint16	4
int8	3
bytes16	1

Table 6.1: Elementary types used

Map key types	Number of appearances
address	2,474
bytes32	348
uint256	62
string	40
bytes4	20
uint64	4
uint32	1

Table 6.2: Elementary types used for key mapping

(215), and simple (2,685) are used. On the other hand, regarding the enum attributes, the parameters are all of custom type.

Analysing functions

There are 74,518 functions in the total number of contracts. However, the parser does not distinguish between normal, receive and fallback functions. The parser, like our ontology, shows the visibility of functions, mutability, and input and output parameters. However, our ontology contains a content type where the function code is provided.

Analysing modifiers

There are 3,448 modifiers in the total number of contracts. In the parser, the modifiers include the code, which in our ontology is a field called "content". However, the parser in the case of modifiers allows the code to be parsed. As a result, our ontology has less granularity because, in functions, the code is in a single string.

Analysing receive and fallback

In the parser, receive and fallback types are not distinguished from functions. However, due to the limitations of Receive and Fallback functions in a contract, they have been separated in our ontology.

Conclusion of the analysis

In light of the results, comparing the ontology with existing parser tools, we can conclude that the granularity of the ontology is extensive; however, it contains a shortcoming, the representation of the code. This could be added in later versions or in a new ontology representing the function code.

The main differences between the parser and the ontology can be found in table 6.3. While the parser does not distinguish between contracts, structs and enums, defining them as UserDefine, our ontology distinguishes each of these types separately. Furthermore, both the parser and the ontology partially support the code of the functions, as the parser has the code with the highest granularity in the modifiers, not appearing in the rest of the functions. In the ontology, however, the code of any function can be obtained, but it is stored in a string field. Finally, the receive and fallback functions in the parser are not distinguished from other functions, whereas in our ontology, they are.

	Parser	**Ontology**
Supports complex types	Partially	Full support
Supports code content	Partially	Partially
Distinguish receive and fallback functions	No	Yes

Table 6.3: Main differences between parser and ontology

6.6.2 Validating the ontology with other tools

On the one hand, Oops! returns three messages about potential pitfalls. However, as explained below, these messages are not defects in the ontologies.

The first result refers to the disjoint elements of the ontology, presenting four pitfalls. Despite the results indicating these pitfalls, the classes are connected to the ontology with a "skos:closeMatch" relationship, which is used to link two concepts that are sufficiently similar to interchange information between them.

The second result indicates the missing domain or range in properties, obtaining 8 cases. These 8 cases have no domain and range because there is no interest in having them in these

elements of the ontology due to the fact that these relations and objects can be used in different classes.

Finally, the last result presents the inverse relationship not explicitly declared. The ontology does not use inverse relations, so this result can be ignored.

On the other hand, the HermiT reasoner provided a positive result by not reporting errors on the ontology, so the ontology has no inconsistencies.

6.7 Discussion

This chapter proposes the creation of an ontological model for the Ethereum blockchain, providing a representation of the Ethereum ecosystem, including the Application Binary Interface (ABI), and the Solidity programming language. By developing specific ontologies for each of these elements, this work not only addresses the representation of the properties and characteristics of the Ethereum ecosystem, but also enables the implementation of tools provided by the semantic web, promoting interoperability, and facilitating their interaction and analysis.

The main contribution of this chapter has been to demonstrate how the integration of the semantic web with blockchain technology is feasible. The ontology developed for the Solidity programming language and for block and transaction representation in Ethereum has enabled a richer and semantically enriched representation of the data stored on the blockchain. This integration enables more advanced queries over the data stored on the blockchain. Furthermore, if more blockchain technologies are represented in the triple store, this integration facilitates better interoperability between different systems and applications.

Furthermore, the ABI defines as functions different methods that are not functions themselves, such as constructors or modifiers; in contrast, aligning the code of Solidity to our ontology provides more granularity, so the results are precise when searching for a particular result. However, this granularity of contracts code can be increased if the ontology were to be extended with the code of the functions, detailing the internal logic of the functions, and thus allowing for more extensive queries. By aligning the Solidity code with the developed ontology, a level of detail beyond that of traditional ABI is achieved, providing a better understanding of smart contract operations.

Ontology validation with deployed smart contracts and other tools provides a methodology to ensure that the ontology represents all components and logic of the Ethereum blockchain. When comparing the contracts deployed by other available tools that parse the grammar with the ontology, it is observed that the ontology possesses greater granularity. This validation process is useful to ensure the accuracy and usability of the ontology.

Chapter 7

Towards a knowledge graph of the Ethereum blockchain

> The scientific man does not aim at an immediate result. He does not expect that his advanced ideas will be readily taken up. His work is like that of the planter - for the future. His duty is to lay the foundation for those who are to come, and point the way.
>
> *Nikola Tesla*

7.1 Introduction

Building on the Ethereum ontologies, which were the focus of the previous chapter, this chapter introduces and details the construction of a knowledge graph for the Ethereum blockchain. Therefore, this chapter presents a method to provide semantic and interoperable data from the Ethereum blockchain. The generated knowledge graph provides a semantic layer to the blockchain, enabling the advantages provided by the semantic web. Having a knowledge graph for a blockchain enables several benefits, such as querying the contracts using an RDF query language, providing richer data by linking the RDF smart contract instances to other datasets, or aligning the knowledge graph ontology to others, allowing semantic interoperability.

In this chapter, the knowledge graph is built on a set of blocks and on the ABIs and smart contracts that belong to these blocks. The knowledge graph is available for the public[1], along with an SPARQL endpoint; it includes a corpus of 2,608 blocks, 8,283 ABIs, and 94,364 smart contracts. In addition, licenses for the open publication of this corpus are analysed, but in this chapter all the contracts are used.

[1] https://kg.dlt.linkeddata.es/

The presented knowledge graph has been built following a method that generates RDF, expressed in N-Triples, from Ethereum data. This knowledge graph provides, using the ontologies presented in chapter 6, blocks modelled according to the Block ontology; ABI contracts, modelled according to the ABI ontology, and Solidity smart contracts, modelled according to the Solidity ontology [27]. The method was implemented in a software called Sancus[2].

Finally, to evaluate this method queries were performed on the final knowledge graph to assess whether the structure of the network allows effective and efficient information retrieval. In addition, triple store metrics such as the number of triples and the space occupied by each triple store are evaluated.

In summary, the aim of this chapter is to create a method (and the consequent implementation) to convert Ethereum blocks and the data (ABI and Solidity contracts) into a knowledge graph, creating a representation of a part of the semantic blockchain based on Ethereum.

The chapter is organised as follows. Section 7.2 introduces the method for generating knowledge graphs and the architecture of the tool for doing this methodology. Section 7.3 explains how the method has been implemented. After that, section 7.5 presents the potential of the knowledge graph. Finally, section 7.6 exposes the conclusions of this chapter.

7.2 Method to generate a knowledge graph from the Ethereum blockchain

The scope of this chapter is to define a method to generate a knowledge graph from a corpus of validated blocks, ABIs, and Solidity smart contracts, and the main goal is the standardisation of the process of converting blockchain into a format that is interoperable and compatible with semantic web technologies, thus facilitating data integration, query, and analysis. The method followed involves the following conceptual steps:

- **Data extraction:** To accomplish this method, a specific range of blocks from the blockchain will be selected first. Within this range, all smart contracts that have been deployed will be identified. Public or private APIs will be used to extract data from blocks and smart contracts within the defined range. These APIs can be blockchain explorers that allow accessing the information of the blocks and transactions and smart contract specific APIs that provide details about the ABI and source code in Solidity.
- **Data transformation:** Second, a data transformation will be performed, mapping the extracted data to a conceptual model that reflects the entities and properties of the blockchain and assigning the RDF types and properties to the extracted elements, ensuring that each entity and relationship are represented in a semantically rich way, using W3C standards, such as N-Triples serialisation (providing a structure that allows data to be represented as a set of triples). During this step, the ontologies developed in Chapter 6 will be used.

[2] https://dlt.linkeddata.es/

- **Data storage:** Data structured in the N-Triple serialisation are stored in a triple store. A knowledge graph is constructed that represents the knowledge contained in the blocks and contracts of the blockchain in the form of an interconnected graph. Triple stores require a considerable amount of space compared to other databases [133], and after generating each triple store, a mechanism provided by the triple store is used to compress the triple store.

7.2.1 Description of the method

Figure 7.1 describes a detailed method for converting blockchain data into a knowledge graph involving a series of iterative and conditional steps. The steps are as follows:

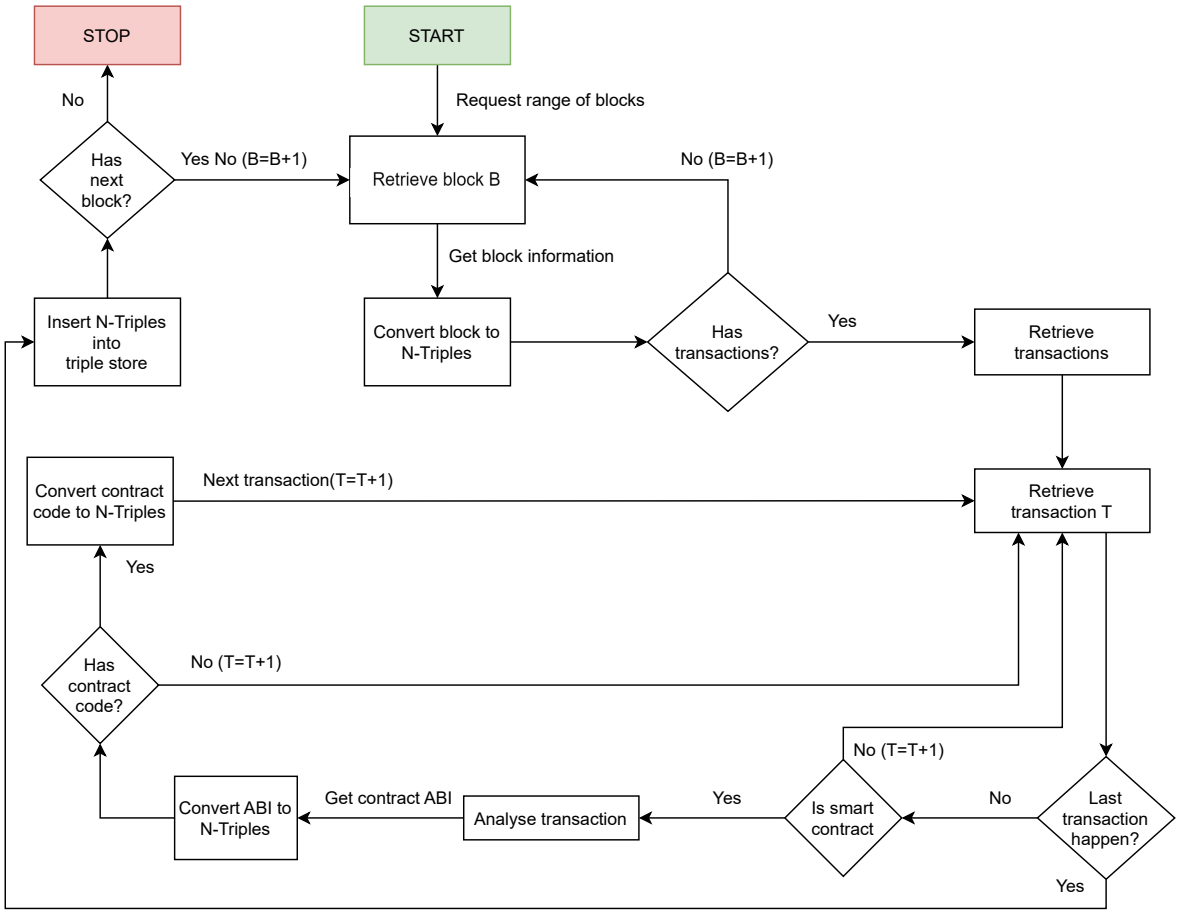

Figure 7.1: Methodology steps diagram

1. The method stars with a request with a range of blocks, and sequentially retrieves each block within the specified range, starting with the first block given.
2. Detailed information is extracted from the retrieved block.
3. The block information is converted to the N-Triples serialisation.
4. Check if the block has transactions.

   (a) If there are no transactions, get the next block (return to step 2).

   (b) If the block has transactions, get the list of the transactions to check, starting with the first transaction present in the given block.

5. Check if the last transaction has been analysed.

   (a) If there are no more transactions to analyse, go to step 10.

   (b) If there are more transactions, check if the transaction is a smart contract.

6. Check if the transaction is a smart contract.

   (a) If the transaction is not a smart contract, get the next transaction (return to step 5).

   (b) If the transaction is a smart contract, analyse the transaction.

7. The ABI code is retrieved and converted to the N-Triples serialisation.

8. Check if the smart contract has a code available.

   (a) If there is no smart contract code, return to step 5.

   (b) If the smart contract has a code available, convert the contracts to N-Triples.

9. Check if there are more transactions in the current block to process (return to step 5).

10. Insert N-triples into the triple store.

11. Check if there are more blocks to analyse.

   (a) If there are more blocks to analyse, complete, analyse next block and go to step 2.

   (b) If there are no more blocks to analyse, complete.

7.3 Implementation of the method

To implement the previous method following the planned steps, the software Sancus[3] has been developed. The architecture of Sancus is depicted in figure 7.2. The first mechanism is to implement the methodology to convert the Ethereum blockchain to a knowledge graph consisting of setting a series of blocks and getting one block by one using the *block retriever* provided by Sancus to semantise the blocks with the *block converter* (section 7.4.1). In the second mechanism, the contracts are retrieved from Etherscan[4], an Ethereum block and smart contract explorer, using the *ABI retriever* provided by Sancus to semantise the ABI contracts with the *ABI converter* (section 7.4.2). The last mechanism is to retrieve the source code of the contracts provided by Etherscan by using the *Solidity retriever* provided by Sancus to semantise the smart contracts code with the *Solidity converter* (section 7.4.3). Finally, all the N-Triples generated are stored in the triple store. The triple store server used is Fuseki[5].

[3] https://github.com/oeg-upm/Sancus
[4] https://etherscan.io
[5] https://jena.apache.org/documentation/fuseki2/

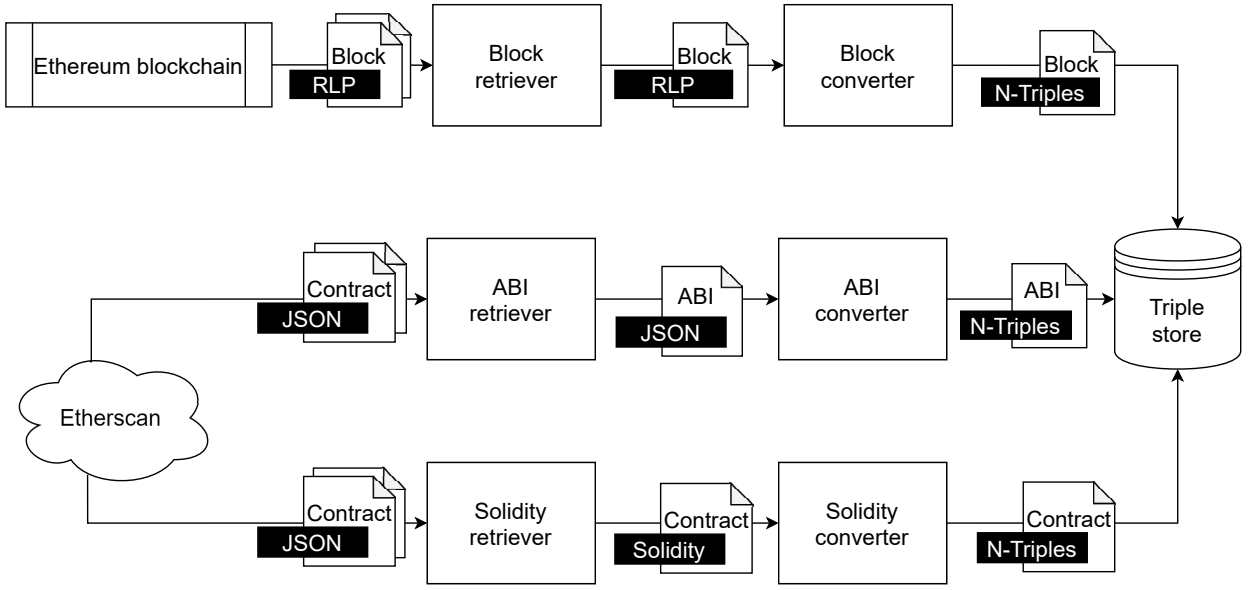

Figure 7.2: Sancus arquitecture

To recover contracts, as depicted in figure 7.3, once the block retriever has the block data, a script filters all transactions inside a block; then, the transactions are sent to the ABI retriever and the Solidity retriever, and Etherscan provides the ABI contract and the Solidity contract code. Etherscan sends a JSON file containing the ABI and the contract code, so this request can only be made once.

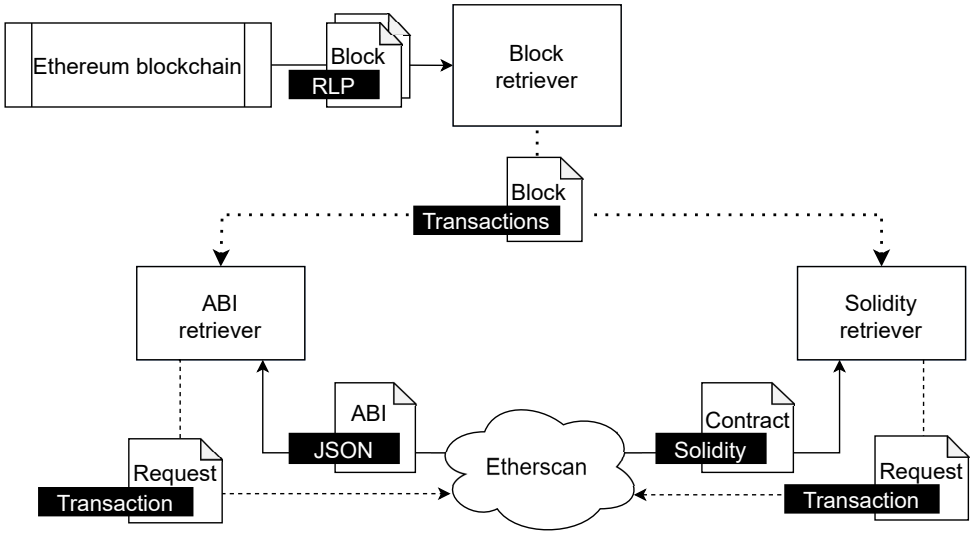

Figure 7.3: Iteration to obtain block and smart contract information

A template of the JSON contract received by Etherscan its detailed by listing 7.1. From this JSON, the fields that are useful for the method are the "ABI" field, which provides the ABI code of the contract, the "SourceCode" field, which provides the source code, and the "LicenseType" field, which provides the license used by the contract.

```
{
   "status":"1",
   "message":"OK",
   "result":[
      {
         "SourceCode":"Solidity_Code",
         "ABI":"ABI_code",
         "ContractName":"Contract_name",
         "CompilerVersion":"version_number",
         "OptimizationUsed":"0",
         "Runs":"200",
         "ConstructorArguments":"",
         "EVMVersion":"Default",
         "Library":"",
         "LicenseType":"License_type",
         "Proxy":"0",
         "Implementation":"",
         "SwarmSource":"url"
      }
   ]
}
```

Listing 7.1: JSON template retrieved from Etherscan

7.4 The Ethereum knowledge graph

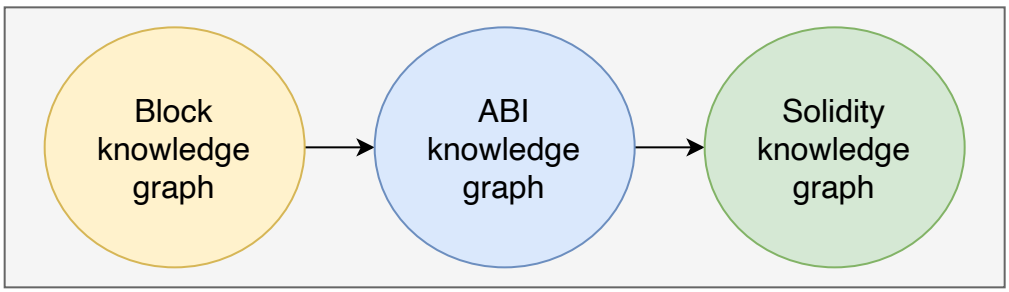

Figure 7.4: Overview of the Ethereum knowledge graph

The N-Triples generated by Sancus are stored in a triple store. The triple store contains three knowledge graphs, as depicted in figure 7.4 and is publicly available[6]. The block knowledge graph stores the block data, and is related with the ABI knowledge graph by the ABI address. The ABI knowledge graph stores the ABI data, and is related with the Smart Contract knowledge graph by the Solidity address. The Solidity knowledge graph stores all the Solidity smart contract data.

[6] https://kg.dlt.linkeddata.es

The number of triples in the blocks is high, mainly due to the number of transactions in each block. While the number of triples stored in Solidity is about 15 million, the number of triples in ABI is only about 3 million. This is due to the granularity of the contracts stored in Solidity, aligning to our ontology, as opposed to the contracts stored in ABI. In the triple store, as shown in table 7.1, the generated source code contracts contain more data than the ABI contracts, expanding the richness of the data and the different possibilities that can be performed with it.

	Triples in blocks	**Triples in ABI**	**Triples in Solidity**
Number of triples	8,434,726	3,167,373	15,555,663

Table 7.1: Triples comparison between the different triple store.

7.4.1 Block module implementation

To implement the methodology, we send to the *block endpoint* a range of blocks and an external Ethereum blockchain endpoint to retrieve the data of the blocks (Sancus does not provide the chain itself, it must be a synchronised chain, either the Ethereum chain locally or using third party services). With this data, the **block module** first collects the Ethereum block data. This block data uses a format called Recursive Length Prefix (RLP) and needs to be converted to the N-Triples serialisation in order to store the data. Each block is stored in a different graph, and the URI is built in this way: *"http://kg.dlt.linkeddata.es/block/**blockNumber**"*, where **blockNumber** is the number of the block. Annex 1 presents a sample of an Ethereum block. The main properties of the Ethereum block (such as the gas used, the hash of the block or the block number) are represented, as well as the transactions (and the main properties of the transaction, at least the properties of the transaction that are stored in the block) and the withdrawals.

1. **Block retriever.** Sancus extracts the blocks, one by one, from the Ethereum blockchain.
2. **Block converter** After receiving the block, Sancus extracts the block data, encoded in RLP, to transform the RLP data into N-Triples fully aligned with the block ontology, using the Web3j library to parse the RLP data.

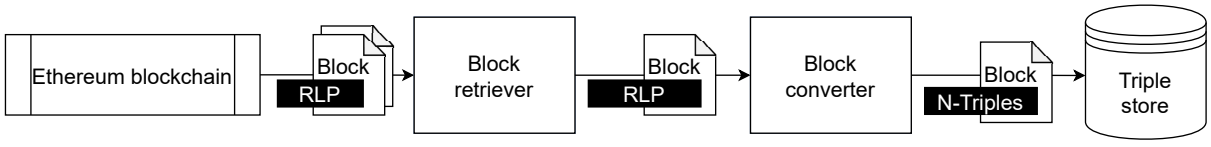

Figure 7.5: Block module workflow in knowledge graphs

Licenses

Regarding licenses of the data stored in the Ethereum blocks, information on Ethereum blocks is considered public data, given the open and decentralised nature of the blockchain.

Data storage

The size of storing 2,608 blocks with all their information in a triple store is, before compacting the triple store, 48,4 gigabytes, and after compacting the triple store is 3,62 gigabytes.

Sample queries

Two samples queries were run; the first one is to retrieve all the triples in the triple store. The second one is to check which contracts had an ABI code (to distinguish normal transactions from smart contracts). The query types and query times can be found in table 7.2.

Query type	Execution time

`select * where {`
`graph ?g {`
`?s ?p ?o .`
`}`
`}`
	>900 seconds
	(no results found)

`select ?s where {`
`graph ?g {`
`?s <https://w3id.org/def/Ethereum/hasABI>?o .`
`}`
`}`
	6,82 seconds
	(8,283 results found)

Table 7.2: Processing time of each query regarding the blocks

In the first query, due to the number of triples to be retrieved, the SPARQL query exceeds the timeout to execute the query, so in all other cases, due to the large number of triples, executing this query would have the same result. The second query verifies that there are 8,283 ABI contracts available for query.

7.4.2 ABI module implementation

The *Block module* sends a list of hashes (belonging to the block transactions) to the *ABI module*. This module processes the list one by one, starting with the ABI endpoint, which needs the hash of the transaction and an external API service that provides the ABI code in JSON format, such as Etherscan (the ABI code is not stored in the blockchain itself). With these data, the **ABI semantiser** collects the ABI contract from an external site. Therefore, it is necessary to analyse the JSON and parse it to the N-Triples serialisation according to the context of the ABI ontology. The block module is depicted in detail in figure 7.5. Each ABI contract is stored on a different graph, and the URI is built in this way: *"http://kg.dlt.linkeddata.es/ABI/**address**"*, where the **address** is the address of the smart contract. Annex 2 presents a sample of an ABI contract.

1. **ABI retriever.** Sancus extracts the ABI from the corresponding Solidity contract. The ABI code is already in JSON, so it does not need to be transformed into another format.

2. **ABI converter** Once the ABI code is received, it is transformed into N-Triples fully

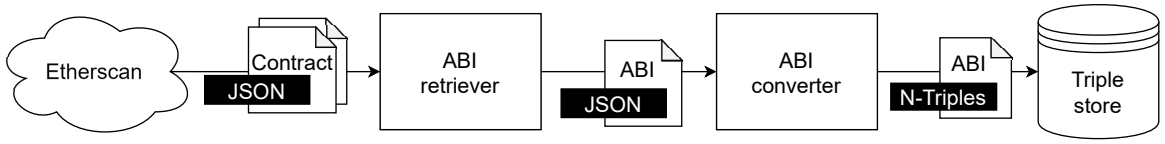

Figure 7.6: ABI module workflow in Sancus

aligned with the ABI ontology.

Licenses

Regarding licenses, the information stored in ABIs is essentially an interface that enables interaction between two programme modules, such as smart contracts and applications on the Ethereum network, describing how data is encoded and decoded for communication. Ethereum provides tools for working with the Ethereum ABI; it is stated that the code is distributed under the MIT license and is calculated based on the contract. This implies that the ABI itself, as a description of the interface for interacting with smart contracts, can be freely generated and used once it is publicly available, although no license for the ABI itself is explicitly mentioned. However, since users provide a license when uploading to Etherscan, these licenses have been used when filtering results. Table 7.3 presents the different licenses found and the number of contracts using each license (or without any license).

ABI license Type	**Number of contracts**
None	4,567
MIT	2,541
GNU GPLv3	382
GNU LGPLv3	306
Apache-2.0	218
Unlicense	160
GNU AGPLv3	59
GNU GPLv2	23
BSD-3-Clause	14
BSL 1.1	7
GNU LGPLv2.1	2
MPL-2.0	2
BSD-2-Clause	1
OSL-3.0	1

Table 7.3: Number of licenses in the ABI smart contract sample

Data storage

The size of storing 8,283 ABIs with all their information in a triple store is, before compacting the triple store, 19,2 gigabytes, and after compacting the triple store 1,24 gigabytes.

Sample queries

Two sample queries were run; the first is to retrieve all ABIs that have the source code

available. One of the main problems of the blockchain is the duplicity of the smart contracts; then, the second query is to retrieve all the contracts with the same contract name, which can be useful to check if the contract is already deployed in the network. In table 7.4 can be found the different types of queries and the time with the results obtained.

Query type	Execution time

PREFIX abi: <https://w3id.org/def/SolidityABI/#>
select distinct * where {
graph ?g {
?s abi:hasImplementation ?o .
}
}
	0.138 seconds
	(8,180 results found)

PREFIX abi: <https://w3id.org/def/SolidityABI/#>
SELECT ?name WHERE{
{SELECT ?s (COUNT(?o) AS ?methodCount) WHERE{
GRAPH ?g1 {
?s abi:hasMethod ?o .
}
} GROUP BY ?s
} FILTER (?methodCount > 50)
} GRAPH ?g2 {
?s abi:name ?name .
}
}GROUP BY ?name
HAVING (COUNT(DISTINCT ?s) > 1)
	0.155 seconds
	(40 results found)

Table 7.4: Processing time of each query regarding the ABI

In the first query, of the 8283 ABI contracts that exist, 8180 have an associated implementation, so not all contracts that have ABI have an associated public code uploaded. The second query shows the contracts that have more than 50 methods and their name, in order to find contracts that are large and therefore would consume a lot of gas, and to be able to check later if it is the same contract.

7.4.3 Solidity module implementation

The *ABI module* sends a list of hashes (belonging to the block transactions) to the *Solidity module*. The last module is the Solidity module, depicted in detail in figure 7.7. Sancus retrieves verified Solidity smart contracts from the Ethereum blockchain through an external site. To preserve the structure of Ethereum, the Solidity smart contracts are grouped by the original hash and are stored in a different graph. The URI is built as follows: *"http://kg.dlt.linkeddata.es/Solidity/**address**"*, where the **address** is the address of the contract, and one address can contain one or more contracts. Annex 3 presents a sample of a Solidity smart contract.

1. **Solidity retriever.** Sancus extracts the source code from the correspondent Solidity

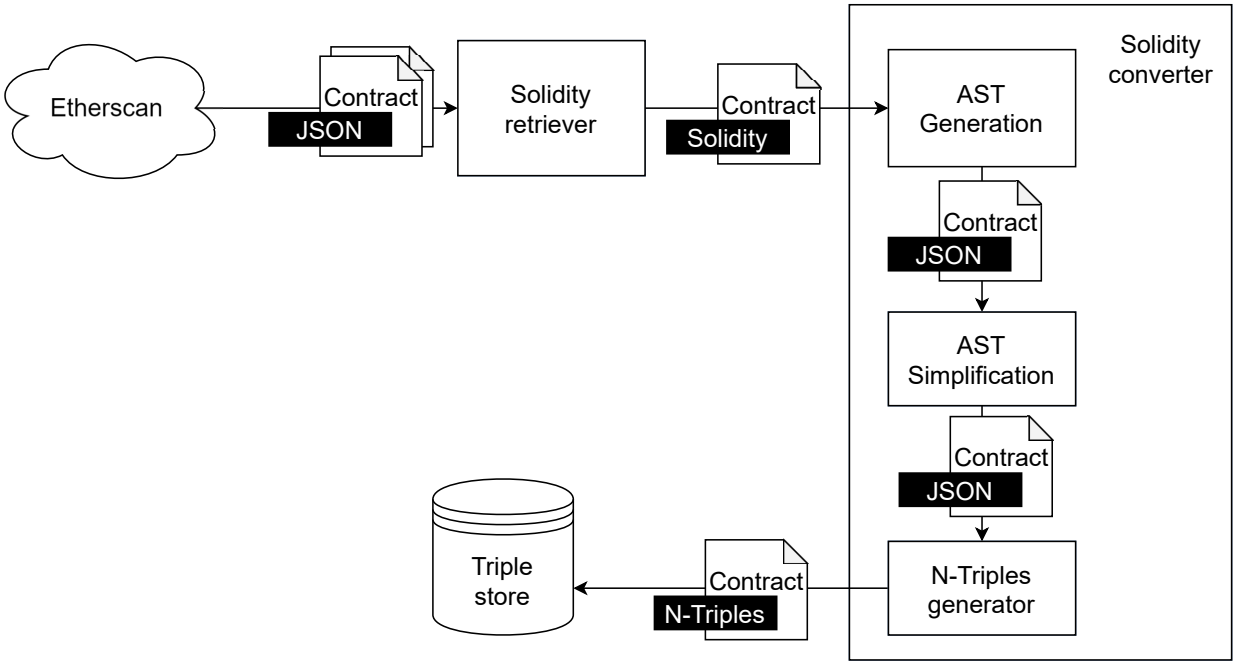

Figure 7.7: Solidity module in Sancus

contract.

2. **Abstract syntax tree (AST) generation.** Once the Solidity source code is received, given the Solidity grammar, Sancus processes the code and generates an abstract syntax tree (AST). This AST is generated in JSON, which will facilitate its later processing.

3. **AST simplification.** Once the JSON AST is obtained, the JSON is processed. The original AST is packed with information and contains all the syntactic elements of the source code. The simplification stage consists of synthesising this complex tree until it is reduced to the representative elements of a Solidity contract, eliminating superfluous nodes and retaining only those that are essential to represent the structure of the Solidity code. This AST simplification returns a clean JSON that is fully aligned with the Solidity ontology.

4. **N-Triples generator.** To transform the JSON Solidity ontology into the final N-Triples, context information is needed. The context element maps terms to identifiers, allowing the transformation from JSON to RDF, so the Solidity ontology will be used to describe the context. In this context, the Solidity contract in the N-Triples serialisation has already been generated.

Licenses

Regarding licencing, smart contracts written in Solidity and stored in Etherscan are subject to licensing terms that define how they can be used, shared, or modified. The license under which a smart contract code is distributed in Solidity can vary and is usually specified at the top of the contract source code. This source code may include an open license, such as the MIT one, which allows anyone to use, modify, and redistribute the code under the

same terms, or it may be more restrictive, depending on the developer's choice. As with the ABI code, once the source code of a smart contract in Solidity is publicly available on platforms such as Etherscan, other developers can interact with it under the terms of its license, although detailed license specifications need to be consulted in the contract source code to fully understand the applicable rights and restrictions. In table 7.5 the different types of licenses can be found.

Solidity license Type	**Number of contracts**
None	58,317
MIT	26,753
GNU GPLv3	3,533
Apache-2.0	1,725
Unlicense	1,348
GNU LGPLv3	1,331
GNU AGPLv3	813
GNU GPLv2	233
BSD-3-Clause	147
BSL 1.1	79
MPL-2.0	63
GNU LGPLv2.1	10
BSD-2-Clause	7
OSL-3.0	5

Table 7.5: Number of licenses in the smart contract code sample

Data storage

The size of storing 94,364 smart contracts with all their information in a triple store is, before compacting the triple store, 87,9 gigabytes, and after compacting the triple store 5,19 gigabytes.

Sample queries

Two sample queries were run. The first one was to retrieve all the smart contracts that had the source code available. The second one is to check all ABI contracts with more than 50 functions. One of the main problems of the blockchain is the duplicity of the smart contracts; then, the third experiment is to retrieve all the contracts with the same contract name, which can be useful to check if the contract is already deployed in the network. In table 7.6 the different types of queries and the time with the results obtained can be found.

In the first query, specific contracts are searched. This is because ABI contracts only refer to the main contract, and not to the other contracts that may import that main contract. The results obtained in the second query provide less results than the equivalent query of table 7.6. This is because the Solidity triple store has more granularity than the ABI triple store data, and thus the Solidity triple store provides more detailed results.

Query type	Execution time

PREFIX sol: <https://w3id.org/def/Solidity#>
select distinct ?s ?contractName where {
graph ?g {
?s sol:contractName ?contractName .
filter(contains(lcase(?contractName), "person"))
}
}
	0.093 seconds
	(2 results found)

PREFIX sol: <https://w3id.org/def/Solidity#>
select ?name where {
{ select ?name where {
{ select ?name (count(?name) as ?countName) where{
{ select ?s where {
{ select ?s (count(?o) as ?numMethods) where {
graph ?g {
?s sol:hasImplementationFunction?o .
} group by ?s having (count(?o) > 50)
}
}
} graph ?g {
?s sol:contractName ?name .
}
}
} group by ?name
} filter (?countName > 1)
}
	0.503 seconds
	(21 results found)

Table 7.6: Processing time of each query regarding the Solidity smart contracts

7.5 Potential of the knowledge graph

With a fast query, it can be seen that there are 103 contracts without a public code, so these 103 contracts cannot be verified and, therefore, could be malicious in order to cause economic harm to users. In addition, although this chapter has used all available contracts, more than half of them have no licence and, therefore, cannot be published.

By including the knowledge graph blocks, ABIs and Solidity smart contracts code, complex queries can be performed; for example, starting with listing 7.2, a query can be performed regarding the number of contracts deployed per hash. In this case, the query searches for the hashes with the highest number of deployed contracts.

```
1 PREFIX eth: <https://w3id.org/def/Ethereum/>
2
3 SELECT ?creator (COUNT(DISTINCT ?abi) AS ?numContractsDeployed) WHERE {
4     SERVICE <http://kg.dlt.linkeddata.es/Blocks/query> {
```

```
        GRAPH ?g {
            ?block eth:hasTransaction ?transaction .
            ?transaction eth:from ?creator .
            ?transaction eth:hasABI ?abi .
        }
    }
} GROUP BY ?creator
ORDER BY DESC(?numContractsDeployed)
LIMIT 10
```

Listing 7.2: SPARQL query

The results obtained by running this query are shown in table 7.7. In this table, it is possible to see how a single address has deployed 213 contracts (in this case, it only takes into account ABI contracts and not all contracts). This brings up the next point, which is to find out which contracts this user has deployed.

Hash of the creator	Number of contracts deployed
0x75e89d5979e4f6fba9f97c104c2f0afb3f1dcb88	213
0x0d0707963952f2fba59dd06f2b425ace40b492fe	199
0x28c6c06298d514db089934071355e5743bf21d60	180
0xf16e9b0d03470827a95cdfd0cb8a8a3b46969b91	178
0xdfd5293d8e347dfe59e90efd55b2956a1343963d	161
0x21a31ee1afc51d94c2efccaa2092ad1028285549	157
0x6a116b531eb60d4f5e2ab31fbcaec5c962865f87	74
0x46340b20830761efd32832a74d7169b29feb9758	72
0xf89d7b9c864f589bbf53a82105107622b35eaa40	67
0x6cc5f688a315f3dc28a7781717a9a798a59fda7b	64

Table 7.7: Number of contracts deployed by hash

Then, based on the last query and the results, a new query can be performed. This SPARQL query uses the address with the highest number of contracts and selects the most frequent contract for that user. The SPARQL query is shown in the listing 7.3.

```
PREFIX eth: <https://w3id.org/def/Ethereum/>
PREFIX solidity: <https://w3id.org/def/SolidityABI/>

SELECT ?contractName (COUNT(DISTINCT ?abiGraph) AS ?numDeployments)
   WHERE {
    SERVICE <http://kg.dlt.linkeddata.es/Blocks/query> {
        GRAPH ?g {
            ?block eth:hasTransaction ?transaction .
            ?transaction eth:from "0
               x75e89d5979e4f6fba9f97c104c2f0afb3f1dcb88" .
            ?transaction eth:hasABI ?abiGraph .
        }
    }
```

```
    SERVICE <http://kg.dlt.linkeddata.es/ABI/query> {
        GRAPH ?abiGraph {
            ?uuid solidity:name ?contractName .
            FILTER (!isBlank(?uuid))
        }
    }
} GROUP BY ?contractName
ORDER BY DESC(?numDeployments)
```

Listing 7.3: SPARQL query

The results obtained in table 7.8 show that out of 213 contracts, at least 7 times the "Token" contract has been deployed and 6 times the "AdminUpgradeabilityProxy" contract. These contracts have the same name and could be contracts already deployed on the blockchain, so we can check the licensing differences that contracts deployed with these names have.

Hash of the creator	**Number of contracts deployed**
Token	7
AdminUpgradeabilityProxy	6
ERC20	3
TransparentUpgradeableProxy	3
ERC1967Proxy	2
MiniMeToken	2
StandardERC20	2
StandardToken	2
AGBToken	1

Table 7.8: Processing time of each query

In the third SPARQL query, a query can be made to check the licence types that exist with the contracts that have the two most deployed names by that user, as shown in listings 7.4 and 7.5.

```
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX solidity: <https://w3id.org/def/Solidity#>

SELECT ?licenseType (COUNT(?licenseType) AS ?numLicenses) WHERE {
    SERVICE <http://kg.dlt.linkeddata.es/Solidity/query> {
        ?contract solidity:name "Token" .
        ?contract dc:license ?licenseType .
    }
} GROUP BY ?licenseType
ORDER BY DESC(?numLicenses)
```

Listing 7.4: SPARQL query

```
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX solidity: <https://w3id.org/def/Solidity#>
```

```
3
4  SELECT ?licenseType (COUNT(?licenseType) AS ?numLicenses) WHERE {
5      SERVICE <http://kg.dlt.linkeddata.es/Solidity/query> {
6          ?contract solidity:name "AdminUpgradeabilityProxy" .
7          ?contract dc:license ?licenseType .
8      }
9  } GROUP BY ?licenseType
10 ORDER BY DESC(?numLicenses)
```

Listing 7.5: SPARQL query

The results obtained in table 7.9 show that different contracts with the same name have different licenses. Since SPARQL queries cannot easily compare triples, a script has been used to compare the source code of the contracts with the name "Token". The results show that at least 66 have the same code without any modification, and the rest of the contracts have minor changes in the code (removal of comments on functions or comments on licences).

Type of license	**Number of appearances**
None	49
MIT	21
Unlicense	2
BSD-3-Clause	1
GNU GPLv3	1

Table 7.9: Different licenses and number of appearances for the contract "Token"

Finally, the results obtained in table 7.10 show similar results to those in table 7.9 with licenses. In this way, a query has been made that would not be possible via the blockchain itself.

Type of license	**Number of appearances**
MIT	47
None	42
Apache-2.0	1
GNU LGPLv3	1

Table 7.10: Different licenses and number of appearances for the contract "AdminUpgradeabilityProxy"

With the analysis of queries performed in section 7.4 and these queries regarding licences, the following conclusions can be made, which could not be possible to extract from the blockchain directly:

- Most of the contracts deployed do not specify a licence. This could indicate that developers are either not concerned about intellectual property or reuse of their code or are not aware of the importance of explicitly stating a licence.

- The MIT licence is the most common explicit licence among the contracts analysed. This licence is known to be permissive, allowing almost any use of the software (including commercial use and modification), as long as the copyright notice and licence are retained on any copy of the software or substantial part of it. Its popularity may reflect developers' preference for sharing their work in an open and permissive manner, facilitating adoption and collaboration.
- There is much less preference for the other licences in the contracts. Unlicensed is a declaration of dedication to the public domain, whereas the other licences are permissive with slightly more restrictive requirements than MIT.
- The high incidence of contracts without an explicit licence and the popularity of permissive licences such as the MIT suggest an ecosystem that favours open innovation and collaboration. However, the absence of a licence can lead to legal uncertainties about the use and distribution of code, potentially inhibiting collaboration or use in projects that require legal clarity.

In conclusion, the distribution of licences suggests a possible lack of awareness or concern about the implications of software licensing. Educating developers about the importance of choosing an appropriate licence could help protect their work, encourage ethical and legal use of software, and facilitate collaboration and reuse of code in a more structured and secure way.

7.6 Discussion

In this chapter, a methodology has been proposed and evaluated to represent a significant portion of the Ethereum blockchain as a knowledge graph, using the ontologies detailed in Chapter 6. This representation is based on the premise that ontologies, by providing a defined and coherent structure, are essential to facilitate interoperability between different blockchain implementations and to improve the ability to analyse and query the data stored in these decentralised technologies.

The main contribution of this chapter has been to demonstrate how the integration of the semantic web with blockchain technology is feasible. The ontology developed for the Solidity programming language and for block and transaction representation in Ethereum has enabled a richer and semantically enriched representation of the data stored in the blockchain. This integration enables more accurate queries of the data stored on the blockchain. In addition, if more blockchain technologies are represented in the triple store, this integration facilitates better interoperability between different systems and applications.

Furthermore, the ABI defines functions as different methods that are not functions themselves, such as constructors or modifiers; in contrast, aligning Solidity's code to our ontology provides more granularity, so the results are precise when searching for a particular result. In addition, it provides greater granularity and greater detail since the ABI does not store, for example, the code of the functions, but our triple store does. However, this granularity of contract code can be increased if the ontology were to be extended with the code of the functions, detailing the internal logic of the functions, and thus allowing for more extensive queries.

Chapter 8

Conclusions and future work

> Every new beginning comes from some other beginning's end.
>
> *Seneca*

This thesis explores the integration of blockchain technologies with the semantic web, exploring how this combination can enhance the decentralisation of the web and improve data management. This thesis explores the synergies between the blockchain and the semantic web to overcome the limitations of the current Internet and promote a new era of decentralisation and data security, highlighting how the blockchain can benefit from the rich data structuring offered by the semantic web and, in turn, how the latter can take advantage of the immutability and decentralisation of the blockchain.

The thesis is based on the premise that, although the blockchain offers a robust platform for secure and decentralised transactions, its potential in the management of complex and structured data is still in early phases. On the other hand, the semantic web, with its ability to structure and link data, has limitations in terms of security and decentralisation.

Therefore, the combination of blockchain and Ethereum promotes an early ecosystem for the development of decentralised applications. In conclusion, this thesis opens up several avenues of research, including optimising semantic data storage in blockchain, developing tools to improve interoperability, and exploring new practical applications.

8.1 Conclusions

As stated in Chapter 3, the overall objective of this thesis is to advance the current state of the art in the combination of the semantic web and blockchain. Due to the fact that blockchain is a recent technology, the combination of these fields is an emerging area, so it is still being studied which is the best way to combine both technologies. Depending on the problem, different approaches can be proposed, which is why this thesis starts by defining the synergies provided by the combination of both technologies and proposes different scenarios for different use cases. The use of these scenarios will depend on the objective to be achieved,

as they will depend on what semantic data is to be stored, or even the use of a particular scenario will be limited to certain blockchain technologies.

Once these scenarios have been developed, the deployment and analysis of different scenarios aimed at storing semantic data on the public blockchain have been carried out. The results obtained are that currently storing semantic data in public blockchains is highly expensive; however, using a virtualiser for RDF and smart contracts helps alleviate this burden.

As for the ontologies developed, ontologies are the basis for achieving semantic interoperability between different blockchain technologies, but standards need to be defined.

The knowledge graph has allowed us to perform queries that would not be possible through the normal use of the blockchain, i.e., contract searches cannot be performed, as Ethereum does not store ABI or Solidity code. This leads us to the fact that building the knowledge graph requires third party services to retrieve these ABI and Solidity codes, so it can be concluded that public blockchains (more specifically, Ethereum), although a way to combine blockchain and the semantic web, are only useful in terms of data analysis.

The objectives of this thesis are fulfilled by the main contributions of the thesis:

- **C1. Identification of synergies between the semantic web and blockchain.** The identification of synergies has yielded a large number of benefits and scenarios, validating hypothesis 1. By analysing the advantages and possibilities that emerge from the combination of semantic web and blockchain technologies, we can conclude the following for each scenario proposed in Chapter 4:
  - **Scenario 1. Blockchain with semantic metadata.** This scenario is useful in research and data analysis contexts. However, this approach needs to be developed and may not be optimal in development environments due to the lack of benefits.
  - **Scenario 2. Blockchain with semantic data.** This scenario facilitates data access and querying and could be useful for research, data analysis, and development. However, the main limitation is the increase in transaction and storage costs, as semantic data often require more space than traditional data formats. Therefore, a specific blockchain should be developed for this scenario.
  - **Scenario 3. Blockchain with virtual semantic data and metadata.** This approach combines the benefits of semantic data accessibility with more efficient storage management. This scenario is the most suitable for storing RDF data in a public blockchain.
  - **Scenario 4. Blockchain referencing another blockchain with semantic data.** This scenario has not been analysed in this thesis. However, this scenario is characterised by challenges in terms of security, trust, and governance between interconnected blockchains, as well as potential latency and performance issues when accessing cross-chain data.
  - **Scenario 5. Semantic blockchain.** This scenario is ideal to fully merge the blockchain and the semantic web, and the representation seen in Chapter 7 is an approximation of this scenario, however, this scenario is hypothetical and needs to

be developed.

- **C2. Feasibility of storing RDF in blockchain.** The feasibility of storing RDF data on blockchain through transactions or smart contracts was evaluated. In conclusion, storing RDF via transactions directly on the blockchain is highly inefficient in terms of cost, while storing information via smart contracts and subsequently using RDF virtualisation techniques is the most efficient way to store information to be read as RDF. This is due to the nature of Ethereum, since it was created for banking transactions and is orientated to cryptocurrencies. Due to the low cost of storage in terms of gas through smart contracts, hypothesis 2 is validated.
- **C3. Development of Ethereum ontologies.** The main features and properties of Ethereum blocks and transactions (such as the ABI generated when a smart contract is deployed) and the Solidity language were identified. These ontologies lay the foundation to enable interoperability between different blockchain technologies and different smart contract languages. Because these ontologies represent the entire blockchain and the Solidity smart contract characteristics, they can be aligned with other ontologies, allowing interoperability and accurate queries, validating hypothesis 3.
- **C4. Representation of the semantic blockchain.** A knowledge graph was designed and developed to visualise and understand how a semantic blockchain would work. In conclusion, it can be concluded that this knowledge graph is an excerpt of what a semantic blockchain based on Ethereum could work, based on the Ethereum ecosystem, representing the structure and transactions of the Ethereum blockchain in a semantic format, which facilitates interoperability, querying, analysis, and information extraction, validating hypothesis 4.

The final conclusion of this thesis is that **the integration of the semantic web and the public blockchain is useful at the research level**. This combination refers to storing RDF data on public blockchains or creating triple-store representing the blockchain is useful at the research level to analyse blockchain data with information external to the blockchain (such as smart contract code).

Additionally, an analysis of blockchain with external data sources can be performed using the triple store. With the triple store, an analysis of the blockchain data can be performed beyond the information stored in the blockchain itself, allowing a detailed analysis of how the blockchain data interact and relate to information from external data. This thesis analysed the licenses of smart contracts deployed on the Ethereum blockchain, revealing a chaotic use of smart contract licenses. Given the public nature of Ethereum, the correct use of licenses is vital to the right management and associated responsibilities.

8.2 Future work

Although all the objectives of this thesis have been met, there are many improvements that could be made to achieve better results. In this section, for each of the aspects of the research carried out in this thesis, some of the open issues that deserve further research are provided.

First, the proposed synergies in this thesis are heavily based on blockchain technology. Specifically, a public blockchain offers a level of decentralisation that a private blockchain cannot, illustrating a dependency on the blockchain's nature. Additionally, as blockchain is a recent technology and constantly evolving, there are opportunities for new approaches in which blockchain and semantic web can be combined, highlighting a dynamic field ripe for further exploration.

The focus of this work has been on storing RDF on the blockchain and addressing current gaps. However, only a few serialisations have been tested and there has not been full utilisation of the potential that semantic web tools offer. Furthermore, the cost associated with storing RDF on the public Ethereum blockchain is noted to be too expensive and exploration of other blockchain technologies has been limited. This thesis has focused solely on the Ethereum blockchain, and hence the development of necessary ontologies has been restricted to this technology.

Moreover, the scope of this thesis is confined to the Solidity smart contract language, limiting the ontologies built to this programming language alone. The representation of the Ethereum blockchain in a knowledge graph has been partial, not exploring other public or private blockchains, which could significantly consume storage space if the entire blockchain were represented. Additionally, the construction and analysis of the knowledge graph have relied heavily on third-party tools due to Ethereum's lack of native smart contract code storage, necessitating external collection.

Given these limitations, our future research will aim to address these issues. We plan to explore the synergies provided by different types of public and private blockchain to understand their unique benefits and limitations. Continuing research on the evolving blockchain technology and semantic web is essential to identify and evaluate new forms of integration that may arise with technological advancements. Expanding the research to include more serialisations and exploring the full use of semantic web tools will enrich blockchain data management and digital rights.

Another key future work involves investigating alternatives to reducing the costs of blockchain storage, such as second-layer solutions or other more cost-efficient blockchain technologies. Developing ontologies for blockchain technologies beyond Ethereum will enhance interoperability and integration with various blockchain platforms. Extending ontology development to include other smart contract languages will cover various applications and use cases across blockchains.

Exploring the representation of alternative blockchains in the knowledge graph will provide a comprehensive view of different blockchain technologies. Furthermore, researching and developing semantic tools to build and analyse blockchain knowledge graphs will reduce dependence on third-party tools and tailor solutions to specific project needs.

Finally, an area of future research closest to this work is the exploration of other blockchain platforms, such as the private blockchain, which enable smart contracts that support CRUD operations and can be written in Java, which could open up new possibilities for integrating semantic web libraries directly into smart contracts. In addition, some private blockchains use CouchDB as a database, which allows us to store RDF data in JSON-LD 1.1 format within

the blockchain itself. Further exploration of private blockchains and their integration with semantic web technologies could be an interesting area of future research.

Bibliography

[1] Abuhassan, I., AlMashaykhi, A.M.: Domain ontology for programming languages. Journal of Computations & Modelling **2**(4), 75–91 (2012)

[2] Aldweesh, A., Alharby, M., Van Moorsel, A.: Performance benchmarking for Ethereum opcodes. In: 2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA). pp. 1–2. IEEE (2018)

[3] Alsamani, A., Beckmann, A.: Combining blockchain and semantic technologies: A survey. In: 2022 IEEE 1st Global Emerging Technology Blockchain Forum: Blockchain & Beyond (iGETblockchain). pp. 1–6. IEEE (2022)

[4] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the thirteenth EuroSys conference. pp. 1–15 (2018)

[5] Atzeni, M., Atzori, M.: CodeOntology: RDF-ization of source code. In: International Semantic Web Conference. pp. 20–28. Springer (2017)

[6] Banerjee, A., Joshi, K.: Link before you share: Managing privacy policies through blockchain. In: 2017 IEEE International Conference on Big Data (Big Data). pp. 4438–4447. IEEE (2017)

[7] Bansod, S., Ragha, L.: Blockchain technology: Applications and research challenges. In: 2020 International Conference for Emerging Technology (INCET). pp. 1–6. IEEE (2020)

[8] Baqa, H., Truong, N.B., Crespi, N., Lee, G.M., Le Gall, F.: Semantic Smart Contracts for Blockchain-based Services in the Internet of Things. In: 2019 IEEE 18th International Symposium on Network Computing and Applications (NCA). pp. 1–5. IEEE (2019)

[9] Bartoletti, M., Pompianu, L.: An analysis of Bitcoin OP_RETURN metadata. In: International Conference on Financial Cryptography and Data Security. pp. 218–230. Springer (2017)

[10] Bartoletti, M., Pompianu, L.: An empirical analysis of smart contracts: platforms, applications, and design patterns. In: International conference on financial cryptography and data security. pp. 494–509. Springer (2017)

[11] Beris, T., Koubarakis, M.: Modeling and preserving greek government decisions using semantic web technologies and permissionless blockchains. In: European Semantic Web

Conference. pp. 81–96. Springer (2018)

[12] Bermès, E.: Enabling your catalogue for the semantic web. Catalogue **2**, 117–142 (2013)

[13] Berners-Lee, T., Fielding, R., Masinter, L.: RFC 3986: Uniform Resource Identifier (URI): Generic syntax (2005)

[14] Berners-Lee, T., Hendler, J., Lassila, O., et al.: The semantic web. Scientific american **284**(5), 28–37 (2001)

[15] Berners-Lee, T.J.: Information management: A proposal. Tech. rep. (1989)

[16] Besançon, L., Da Silva, C.F., Ghodous, P., Gelas, J.P.: A Blockchain Ontology for DApps Development. IEEE Access **10**, 49905–49933 (2022)

[17] Bistarelli, S., Mazzante, G., Micheletti, M., Mostarda, L., Sestili, D., Tiezzi, F.: Ethereum smart contracts: Analysis and statistics of their source code and opcodes. Internet of Things **11**, 100198 (2020)

[18] Bizer, C.: Interlinking scientific data on a global scale. Data Science Journal **12**, GRDI6–GRDI12 (2013)

[19] Bizer, C., Heath, T., Berners-Lee, T.: Linked data: The story so far. In: Semantic services, interoperability and web applications: emerging concepts, pp. 205–227. IGI Global (2011)

[20] Blaney, J.: Introduction to the principles of linked open data. The Programming Historian (2017)

[21] Brand, S., et al.: Whole earth catalog. Point Foundation (1968)

[22] Brickley, D., Guha, R.V., McBride, B.: RDF Schema 1.1. W3C recommendation **25**, 2004–2014 (2014)

[23] Buil-Aranda, C., Arenas, M., Corcho, O.: Semantics and optimization of the SPARQL 1.1 federation extension. In: Extended Semantic Web Conference. pp. 1–15. Springer (2011)

[24] Buterin, V.: Ethereum: A next-generation cryptocurrency and decentralized application platform. Bitcoin Magazine **23** (2014)

[25] Buterin, V., et al.: A next-generation Smart Contract and decentralized application platform Ethereum. White Paper. Ethereum Project White Paper (2014)

[26] Cano-Benito, J., Cimmino, A., García-Castro, R.: Towards blockchain and semantic web. In: Business Information Processing workshop. Springer (2019)

[27] Cano-Benito, J., Cimmino, A., García-Castro, R.: Toward the ontological modeling of smart contracts: A solidity use case. IEEE Access **9**, 140156–140172 (2021)

[28] Chatterjee, K., Goharshady, A.K., Goharshady, E.K.: The treewidth of smart contracts. In: Proceedings of the 34th ACM/sigapp symposium on applied computing. pp. 400–408 (2019)

[29] Chaum, D.L.: Computer Systems established, maintained and trusted by mutually suspicious groups. Electronics Research Laboratory, University of California (1979)

[30] Chawla, T., Singh, G., Pilli, E.S., Govil, M.C.: Storage, partitioning, indexing and retrieval in Big RDF frameworks: A survey. Computer Science Review **38**, 100309 (2020)

[31] Chen, H., Hu, N., Qi, G., Wang, H., Bi, Z., Li, J., Yang, F.: Openkg chain: A blockchain infrastructure for open knowledge graphs. Data Intelligence **3**(2), 205–227 (2021)

[32] Choudhury, O., Rudolph, N., Sylla, I., Fairoza, N., Das, A.: Auto-generation of smart contracts from domain-specific ontologies and semantic rules. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). pp. 963–970. IEEE (2018)

[33] Cimmino, A., Oravec, V., Serena, F., Kostelnik, P., Poveda-Villalón, M., Tryferidis, A., García-Castro, R., Vanya, S., Tzovaras, D., Grimm, C.: VICINITY: IoT semantic interoperability based on the web of things. In: 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS). pp. 241–247. IEEE (2019)

[34] Cimmino, A., Fernández-Izquierdo, A., García-Castro, R.: Astrea: automatic generation of shacl shapes from ontologies. In: European Semantic Web Conference. pp. 497–513. Springer (2020)

[35] Cimmino Arriaga, A., García Castro, R.: Helio: A framework for implementing the life cycle of knowledge graphs. Semantic Web **15**, 1–27 (01 2023). https://doi.org/10.3233/SW-233224

[36] Clark, K.L., McCabe, F.G.: Ontology oriented programming in Go! Applied Intelligence **24**(3), 189–204 (2006)

[37] Cohen, B.: The rise of alternative currencies in post-capitalism. Journal of Management Studies **54**(5), 739–746 (2017)

[38] Dallyn, S., Frenzel, F.: The Challenge of Building a Scalable Postcapitalist Commons: The Limits of FairCoin as a Commons-Based Cryptocurrency. Antipode **53**(3), 859–883 (2021)

[39] De Kruijff, J., Weigand, H.: Towards a blockchain ontology. The Netherlands, pdfs. semanticscholar. org/0782/c5badb4f407ee0964d07eda9f74a92de3298. pdf [dostęp 22.07. 2018] (2017)

[40] Demir, M., Turetken, O., Ferwom, A.: Blockchain and iot for delivery assurance on supply chain (BIDAS). In: 2019 IEEE International Conference on Big Data (Big Data). pp. 5213–5222. IEEE (2019)

[41] Di Sorbo, A., Laudanna, S., Vacca, A., Visaggio, C.A., Canfora, G.: Profiling gas consumption in Solidity smart contracts. Journal of Systems and Software **186**, 111193 (2022)

[42] Diatta, B., Basse, A., Ouya, S.: PasOnto: Ontology for Learning Pascal Programming Language. In: 2019 IEEE Global Engineering Education Conference (EDUCON). pp. 749–754. IEEE (2019)

[43] Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In: LDOW (2014)

[44] Dolata, U., Schrape, J.F.: Collectivity and power on the internet: A sociological perspective. Springer (2018)

[45] Dürst, M., Suignard, M.: Internationalized resource identifiers (IRIs). Tech. rep. (2005)

[46] Dwivedi, V., Norta, A., Wulf, A., Leiding, B., Saxena, S., Udokwu, C.: A formal specification smart-contract language for legally binding decentralized autonomous organizations. IEEE Access **9**, 76069–76082 (2021)

[47] Dwivedi, V., Pattanaik, V., Deval, V., Dixit, A., Norta, A., Draheim, D.: Legally enforceable smart-contract languages: A systematic literature review. ACM Computing Surveys (CSUR) **54**(5), 1–34 (2021)

[48] English, M., Auer, S., Domingue, J.: Block chain technologies & the semantic web: A framework for symbiotic development. In: Computer Science Conference for University of Bonn Students, J. Lehmann, H. Thakkar, L. Halilaj, and R. Asmat, Eds. pp. 47–61 (2016)

[49] Faísca, J.G., Rogado, J.Q.: Decentralized semantic identity. In: Proceedings of the 12th International Conference on Semantic Systems. pp. 177–180 (2016)

[50] Faye, D.C., Cure, O., Blin, G.: A survey of RDF storage approaches. Revue Africaine de Recherche en Informatique et Mathématiques Appliquées **15** (2012)

[51] Fenu, G., Marchesi, L., Marchesi, M., Tonelli, R.: The ICO phenomenon and its relationships with ethereum smart contract environment. In: 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE). pp. 26–32. IEEE (2018)

[52] Foschini, L., Gavagna, A., Martuscelli, G., Montanari, R.: Hyperledger fabric blockchain: Chaincode performance analysis. In: ICC 2020-2020 IEEE International Conference on Communications (ICC). pp. 1–6. IEEE (2020)

[53] Gangemi, A., Presutti, V.: The bourne identity of a web resource. In: Proceedings of Identity Reference and the Web Workshop (IRW) at the WWW Conference (2006)

[54] García, R., Gil, R.: Social media copyright management using semantic web and blockchain. In: Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services. pp. 339–343 (2019)

[55] Garcia-Teruel, R.M.: Legal challenges and opportunities of blockchain technology in the real estate sector. Journal of Property, Planning and Environmental Law **12**(2), 129–145 (2020)

[56] Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: Hermit: an owl 2 reasoner.

Journal of Automated Reasoning **53**(3), 245–269 (2014)

[57] Goldman, N.M.: Ontology-oriented programming: static typing for the inconsistent programmer. In: International Semantic Web Conference. pp. 850–865. Springer (2003)

[58] Gómez-Carmona, O., Buján-Carballal, D., Casado-Mansilla, D., López-de Ipiña, D., Cano-Benito, J., Cimmino, A., Poveda-Villalón, M., García-Castro, R., Almela-Miralles, J., Apostolidis, D., et al.: Mind the gap: The AURORAL ecosystem for the digital transformation of smart communities and rural areas. Technology in Society **74**, 102304 (2023)

[59] Graux, D., Sejdiu, G., Jabeen, H., Lehmann, J., Sui, D., Muhs, D., Pfeffer, J.: Profiting from kitties on ethereum: Leveraging blockchain RDF data with SANSA. SEMANTiCS Conference (2018)

[60] Haber, S., Stornetta, W.S.: How to time-stamp a digital document. Springer (1991)

[61] Halpin, H.: The semantic web: The origins of artificial intelligence redux. In: Third international workshop on the history and philosophy of logic, mathematics, and computation (HPLMC-04 2005). Citeseer (2004)

[62] Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 query language. W3C recommendation **21**(10), 778 (2013)

[63] Harz, D., Knottenbelt, W.: Towards safer smart contracts: A survey of languages and verification methods. arXiv preprint arXiv:1809.09805 (2018)

[64] Hausenblas, M., Slany, W., Ayers, D.: A Performance and Scalability Metric for Virtual RDF Graphs. In: SFSW (2007)

[65] Hayes, J., Gutierrez, C.: Bipartite graphs as intermediate model for RDF. In: The Semantic Web–ISWC 2004: Third International Semantic Web Conference, Hiroshima, Japan, November 7-11, 2004. Proceedings 3. pp. 47–61. Springer (2004)

[66] He, N., Wu, L., Wang, H., Guo, Y., Jiang, X.: Characterizing code clones in the Ethereum smart contract ecosystem. In: International Conference on Financial Cryptography and Data Security. pp. 654–675. Springer (2020)

[67] Hegedűs, P.: Towards analyzing the complexity landscape of solidity based ethereum smart contracts. Technologies **7**(1), 6 (2019)

[68] Hoffman, M.R., Ibáñez, L., Fryer, H., Simperl, E.: Smart papers: Dynamic publications on the blockchain. In: European Semantic Web Conference. pp. 304–318. Springer (2018)

[69] Hong, D., Gu, Q., Whitehouse, K.: High-dimensional time series clustering via cross-predictability. In: Artificial Intelligence and Statistics. pp. 642–651. PMLR (2017)

[70] Hong, S.: P2p networking based internet of things (iot) sensor node authentication by blockchain. Peer-to-Peer Networking and Applications **13**(2), 579–589 (2020)

[71] Horrocks, I., Parsia, B., Patel-Schneider, P., Hendler, J.: Semantic web architecture:

Stack or two towers? In: International Workshop on Principles and Practice of Semantic Web Reasoning. pp. 37–41. Springer (2005)

[72] Huang, J., Abadi, D.J., Ren, K.: Scalable SPARQL querying of large RDF graphs. Proceedings of the VLDB Endowment **4**(11), 1123–1134 (2011)

[73] Ibánez, L., Fryer, H., Simperl, E.: Attaching semantic metadata to cryptocurrency transactions (2017)

[74] Iman, L., Davenport, J.: Approximations of the critical region of the Friedman statistic. Communications in Statics pp. 571–595 (1980)

[75] Jurgelaitis, M., Butkienė, R., et al.: Solidity code generation from uml state machines in model-driven smart contract development. IEEE Access **10**, 33465–33481 (2022)

[76] Kaleem, M., Mavridou, A., Laszka, A.: Vyper: A security comparison with solidity based on common vulnerabilities. In: 2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS). pp. 107–111. IEEE (2020)

[77] Kapoor, N., Bahl, N.: Comparative study of forward and backward chaining in artificial intelligence. International Journal Of Engineering And Computer Science **5**(4) (2016)

[78] Kaygin, E., Zengin, Y., Topcuoglu, E., Ozkes, S.: The evaluation of block chain technology within the scope of ripple and banking activities. Journal of Central Banking Theory and Practice **10**(3), 153–167 (2021)

[79] Khoshnampour, M., Nosrati, M.: An overview of E-commerce. World Applied Programming **1**(2), 94–99 (2011)

[80] Kim, H.M., Laskowski, M., Nan, N.: A first step in the co-evolution of blockchain and ontologies: towards engineering an ontology of governance at the blockchain protocol level. arXiv preprint arXiv:1801.02027 (2018)

[81] Kim, H.M., Laskowski, M.: Toward an ontology-driven blockchain design for supply-chain provenance. Intelligent Systems in Accounting, Finance and Management **25**(1), 18–27 (2018)

[82] Kitchenham, B.: Procedures for performing systematic reviews. Keele, UK, Keele University **33**(2004), 1–26 (2004)

[83] Kouneli, A., Solomou, G., Pierrakeas, C., Kameas, A.: Modeling the knowledge domain of the java programming language as an ontology. In: International Conference on Web-Based Learning. pp. 152–159. Springer (2012)

[84] de Kruijff, J., Weigand, H.: Understanding the blockchain using enterprise ontology. In: International Conference on Advanced Information Systems Engineering. pp. 29–43. Springer (2017)

[85] de Kruijff, J., Weigand, H.: Ontologies for commitment-based smart contracts. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". pp. 383–398. Springer (2017)

[86] Kshetri, N., Voas, J.: Blockchain in developing countries. It Professional **20**(2), 11–14 (2018)

[87] Kudumakis, P., Wilmering, T., Sandler, M., Rodríguez-Doncel, V., Boch, L., Delgado, J.: The challenge: from MPEG intellectual property rights ontologies to smart contracts and blockchains [standards in a nutshell]. IEEE Signal Processing Magazine **37**(2), 89–95 (2020)

[88] Kushwaha, S.S., Joshi, S., Singh, D., Kaur, M., Lee, H.N.: Systematic review of security vulnerabilities in Ethereum blockchain smart contract. IEEE Access **10**, 6605–6621 (2022)

[89] Larimer, D.: Transactions as proof-of-stake. Nov-2013 **909** (2013)

[90] Le-Tuan, A., Hingu, D., Hauswirth, M., Le-Phuoc, D.: Incorporating Blockchain into RDF Store at the Lightweight Edge Devices. In: International Conference on Semantic Systems. pp. 369–375. Springer (2019)

[91] Lee, J.J., Madhuranath, S.B.: Blockchain fundamentals and classifications-a pathway to the moon. Issues in Information Systems **23**(4) (2022)

[92] Lee, M.C., Ye, D.Y., Wang, T.I.: Java learning object ontology. In: 5th IEEE International Conference on Advanced Learning Technologies (ICALT'05). pp. 538–542. IEEE (2005)

[93] Lefrançois, M., Zimmermann, A., Bakerally, N.: A SPARQL extension for generating RDF from heterogeneous formats. In: European Semantic Web Conference. pp. 35–50. Springer (2017)

[94] Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: Twenty-ninth AAAI conference on artificial intelligence (2015)

[95] Malsa, N., Vyas, V., Gautam, J.: Blockchain platforms and interpreting the effects of bitcoin pricing on cryptocurrencies. In: Soft Computing: Theories and Applications: Proceedings of SoCTA 2020, Volume 1. pp. 137–147. Springer (2022)

[96] Maly, R.J., Mischke, J., Kurtansky, P., Stiller, B.: Comparison of centralized (client-server) and decentralized (peer-to-peer) networking. Semester thesis, ETH Zurich, Zurich, Switzerland pp. 1–12 (2003)

[97] Marchesi, L., Marchesi, M., Destefanis, G., Barabino, G., Tigano, D.: Design patterns for gas optimization in ethereum. In: 2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE). pp. 9–15. IEEE (2020)

[98] McGeady, S.: The digital reformation: Total freedom, risk, and responsibility'(1996). Harvard Journal of Law and Technology **10**, 137–at (1996)

[99] McGuinness, D.L., Van Harmelen, F., et al.: OWL web ontology language overview. W3C recommendation **10**(10), 2004 (2004)

[100] Mendes, P.N., Mühleisen, H., Bizer, C.: Sieve: linked data quality assessment and

fusion. In: Proceedings of the 2012 joint EDBT/ICDT workshops. pp. 116–123 (2012)

[101] Mingxiao, D., Xiaofeng, M., Zhe, Z., Xiangwei, W., Qijun, C.: A review on consensus algorithm of blockchain. In: 2017 IEEE international conference on systems, man, and cybernetics (SMC). pp. 2567–2572. IEEE (2017)

[102] Modoni, G.E., Sacco, M., Terkaj, W.: A survey of RDF store solutions. In: 2014 International Conference on Engineering, Technology and Innovation (ICE). pp. 1–7. IEEE (2014)

[103] Mohanta, B.K., Panda, S.S., Jena, D.: An overview of smart contract and use cases in blockchain technology. In: 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT). pp. 1–4. IEEE (2018)

[104] Murphy, H.M., McBean, E.A., Farahbakhsh, K.: Appropriate technology–a comprehensive approach for water and sanitation in the developing world. Technology in Society **31**(2), 158–167 (2009)

[105] Naaz, R., Saxena, A.K.: A methodology for selecting smart contract in blockchain-based applications. In: Advances in Cognitive Science and Communications: Selected Articles from the 5th International Conference on Communications and Cyber-Physical Engineering (ICCCE 2022), Hyderabad, India. pp. 1–9. Springer (2023)

[106] Naim, B.A., Klas, W.: Knowledge graph-enhanced blockchains by integrating a graph-data service-layer. In: 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS). pp. 420–427. IEEE (2019)

[107] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Tech. rep., Manubot (2019)

[108] Nakamoto, S., et al.: Bitcoin: A peer-to-peer electronic cash system (2008)

[109] Negroponte, N., Harrington, R., McKay, S.R., Christian, W.: Being digital. Computers in Physics **11**(3), 261–262 (1997)

[110] Nejdl, W., Wolpers, M., Capelle, C., Wissensverarbeitung, R., et al.: The RDF schema specification revisited. In Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung 2000 (2000)

[111] Nentwig, M., Hartung, M., Ngonga Ngomo, A.C., Rahm, E.: A survey of current link discovery frameworks. Semantic Web **8**(3), 419–436 (2017)

[112] Nguyen, C.T., Hoang, D.T., Nguyen, D.N., Niyato, D., Nguyen, H.T., Dutkiewicz, E.: Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities. IEEE access **7**, 85727–85745 (2019)

[113] Nilsson, M., Palmér, M., Naeve, A.: Semantic web metadata for e-learning. Some Ontology (2001)

[114] Nizamuddin, N., Abugabah, A.: Blockchain for automotive: An insight towards the ipfs blockchain-based auto insurance sector. International Journal of Electrical and Computer Engineering (IJECE) **11** (2021)

[115] Norta, A., Ma, L., Duan, Y., Rull, A., Kõlvart, M., Taveter, K.: Econtractual choreography-language properties towards cross-organizational business collaboration. Journal of internet services and applications **6**(1), 1–23 (2015)

[116] Nugent, T., Upton, D., Cimpoesu, M.: Improving data transparency in clinical trials using blockchain smart contracts. F1000Research **5** (2016)

[117] Ouf, S.: A Proposed Architecture for Pharmaceutical Supply Chain Based Semantic Blockchain. International Journal of Intelligent Engineering and Systems **14**(3), 31–42 (2021)

[118] O'reilly, T.: What is web 2.0 (2005)

[119] Panarello, A., Tapas, N., Merlino, G., Longo, F., Puliafito, A.: Blockchain and iot integration: A systematic survey. Sensors **18**(8), 2575 (2018)

[120] Patwardhan, B., Tillu, G.: Universal health coverage and ayush systems. Journal of Ayurveda and Integrative Medicine **9**(1), 1 (2018)

[121] Paul, P., Aithal, P., Saavedra, R., Ghosh, S.: Blockchain Technology and its Types—A Short Review. International Journal of Applied Science and Engineering (IJASE) **9**(2), 189–200 (2021)

[122] Pazarbasioglu, C., Mora, A.G., Uttamchandani, M., Natarajan, H., Feyen, E., Saal, M.: Digital financial services. World Bank **54** (2020)

[123] Petri, I., Barati, M., Rezgui, Y., Rana, O.F.: Blockchain for energy sharing and trading in distributed prosumer communities. Computers in Industry **123**, 103282 (2020)

[124] Pfeffer, J., Beregszazi, A., Detrio, C., Junge, H., Chow, J., Oancea, M., Pietrzak, M., Khatchadourian, S., Bertolo, S.: EthOn-an Ethereum ontology (2016)

[125] Pierrakeas, C., Solomou, G., Kameas, A.: An ontology-based approach in learning programming languages. In: 2012 16th Panhellenic Conference on Informatics. pp. 393–398. IEEE (2012)

[126] Pierro, G.A.: Smart-graph: Graphical representations for smart contract on the ethereum blockchain. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 708–714. IEEE (2021)

[127] Pillai, S.G., Soon, L.K., Haw, S.C.: Comparing DBpedia, Wikidata, and YAGO for web information retrieval. In: Intelligent and Interactive Computing: Proceedings of IIC 2018. pp. 525–535. Springer (2019)

[128] Pogran, K.T., Reed, D.P.: The MIT laboratory for computer science network. COM78 pp. 20–22 (1968)

[129] Popov, S., Lu, Q.: IOTA: feeless and free. IEEE Blockchain Technical Briefs (2019)

[130] Postman, N.: Building a bridge to the 18th century: How the past can improve our future. Vintage (2011)

[131] Poveda-Villalón, M., Fernández-Izquierdo, A., Fernández-López, M., García-Castro, R.:

LOT: An industrial oriented ontology engineering framework. Engineering Applications of Artificial Intelligence **111**, 104755 (2022)

[132] Poveda-Villalón, M., Gómez-Pérez, A., Suárez-Figueroa, M.C.: OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation. International Journal on Semantic Web and Information Systems (IJSWIS) **10**(2), 7–34 (2014)

[133] Prokudin, A., Denisov, M., Sychev, O.: Disk Space Consumption by Triple Storage Systems. In: Novel & Intelligent Digital Systems Conferences. pp. 266–275. Springer (2022)

[134] Qiu, T., Zhang, R., Gao, Y.: Ripple vs. swift: Transforming cross border remittance using blockchain technology. Procedia computer science **147**, 428–434 (2019)

[135] Reddy, E., Lawack, V.: An overview of the regulatory developments in south africa regarding the use of cryptocurrencies. SA Mercantile Law Journal **31**(1), 1–28 (2019)

[136] Ritzer, G., Dean, P., Jurgenson, N.: The coming of age of the prosumer. American behavioral scientist **56**(4), 379–398 (2012)

[137] Rohloff, K., Dean, M., Emmons, I., Ryder, D., Sumner, J.: An evaluation of triple-store technologies for large data stores. In: On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops: OTM Confederated International Workshops and Posters, AWeSOMe, CAMS, OTM Academy Doctoral Consortium, MONET, OnToContent, ORM, PerSys, PPN, RDDS, SSWS, and SWWS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part II. pp. 1105–1114. Springer (2007)

[138] Ruta, M., Scioscia, F., Ieva, S., Capurso, G., Di Sciascio, E.: Supply chain object discovery with semantic-enhanced blockchain. In: Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems. pp. 1–2 (2017)

[139] Ruta, M., Scioscia, F., Ieva, S., Capurso, G., Loseto, G., Gramegna, F., Pinto, A., Di Sciascio, E.: Semantic-enhanced blockchain technology for smart cities and communities. In: 3rd Italian Conference on ICT for Smart Cities & Communities (I-CiTies 2017) (2017)

[140] Ruta, M., Scioscia, F., Ieva, S., Capurso, G., Di Sciascio, E.: Semantic blockchain to improve scalability in the internet of things. Open Journal of Internet Of Things (OJIOT) **3**(1), 46–61 (2017)

[141] Ruta, M., Scioscia, F., Ieva, S., Capurso, G., Pinto, A., Di Sciascio, E.: A Blockchain Infrastructure for the Semantic Web of Things. In: SEBD (2018)

[142] Sajja, G.S., Rane, K.P., Phasinam, K., Kassanuk, T., Okoronkwo, E., Prabhu, P.: Towards applicability of blockchain in agriculture sector. Materials Today: Proceedings **80**, 3705–3708 (2023)

[143] Sanka, A.I., Cheung, R.C.: A systematic review of blockchain scalability: Issues, solutions, analysis and future research. Journal of Network and Computer Applications **195**, 103232 (2021)

[144] Sauermann, L., Cyganiak, R., Völkel, M.: Cool URIs for the semantic web (2007)

[145] Schrape, J.F.: The promise of technological decentralization. A brief reconstruction. Society **56**, 31–37 (2019)

[146] Sergey, I., Kumar, A., Hobor, A.: Scilla: a smart contract intermediate-level language. arXiv preprint arXiv:1801.00687 (2018)

[147] Sharma, N., Shamkuwar, M., Singh, I.: The history, present and future with iot. In: Internet of Things and Big Data Analytics for Smart Generation, pp. 27–51. Springer (2019)

[148] Sheeba, T., Begum, S.H., Bernard, M.J.: Semantic web to e-Learning content. International Journal of Advanced Research in Computer Science and Software Engineering **2**(10.2012) (2012)

[149] Shkembi, K., Kochovski, P., Papaioannou, T.G., Barelle, C., Stankovski, V.: Semantic web and blockchain technologies: Convergence, challenges and research trends. Journal of Web Semantics p. 100809 (2023)

[150] Sicilia, M., Visvizi, A.: Blockchain and OECD data repositories: opportunities and policymaking implications. Library hi tech (2019)

[151] Sidoroff, T., Hyvönen, E.: Semantic e-goverment portals-a case study. In: Proceedings of the ISWC-2005 Workshop Semantic Web Case Studies and Best Practices for eBusiness SWCASE05. vol. 7 (2005)

[152] Sikorski, J.J., Haughton, J., Kraft, M.: Blockchain technology in the chemical industry: Machine-to-machine electricity market. Applied Energy **195**, 234–246 (2017)

[153] Skotnica, M., Pergl, R.: Das contract-a visual domain specific language for modeling blockchain smart contracts. In: Enterprise Engineering Working Conference. pp. 149–166. Springer (2019)

[154] Sopek, M., Gradzki, P., Kosowski, W., Kuziski, D., Trójczak, R., Trypuz, R.: Graphchain: a distributed database with explicit semantics and chained rdf graphs. In: Companion Proceedings of the The Web Conference 2018. pp. 1171–1178 (2018)

[155] Sosnovsky, S., Gavrilova, T.: Development of educational ontology for C-programming (2006)

[156] Studer, R., Benjamins, V.R., Fensel, D.: Knowledge engineering: Principles and methods. Data & knowledge engineering **25**(1-2), 161–197 (1998)

[157] Taherdoost, H.: A critical review of blockchain acceptance models—blockchain technology adoption frameworks and applications. Computers **11**(2), 24 (2022)

[158] Tang, N.: Big rdf data cleaning. In: 2015 31st IEEE International Conference on Data Engineering Workshops. pp. 77–79. IEEE (2015)

[159] Ticona-Herrera, R., Tekli, J., Chbeir, R., Laborie, S., Dongo, I., Guzman, R.: Toward RDF normalization. In: Conceptual Modeling: 34th International Conference, ER 2015,

Stockholm, Sweden, October 19-22, 2015, Proceedings 34. pp. 261–275. Springer (2015)

[160] Tijan, E., Aksentijević, S., Ivanić, K., Jardas, M.: Blockchain technology implementation in logistics. Sustainability **11**(4), 1185 (2019)

[161] Tomaszuk, D., Hyland-Wood, D.: RDF 1.1: Knowledge representation and data integration language for the Web. Symmetry **12**(1), 84 (2020)

[162] Ugarte, H.: A more pragmatic Web 3.0: Linked Blockchain Data. Bonn, Germany (2017)

[163] Vacca, A., Di Sorbo, A., Visaggio, C.A., Canfora, G.: A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges. Journal of Systems and Software **174**, 110891 (2021)

[164] Vandenbussche, P.Y., Atemezing, G.A., Poveda-Villalón, M., Vatant, B.: Linked Open Vocabularies (LOV): a gateway to reusable semantic vocabularies on the Web. Semantic Web **8**(3), 437–452 (2017)

[165] Vujičić, D., Jagodić, D., Ranđić, S.: Blockchain technology, bitcoin, and Ethereum: A brief overview. In: 2018 17th international symposium infoteh-jahorina (infoteh). pp. 1–6. IEEE (2018)

[166] Wang, J., Wang, S., Guo, J., Du, Y., Cheng, S., Li, X.: A summary of research on blockchain in the field of intellectual property. Procedia computer science **147**, 191–197 (2019)

[167] Wang, X., Ban, T., Chen, L., Usman, M., Guan, Y., Lyu, D., Cheng, J., Chen, H., Leung, C., Miao, C.: Decentralised Knowledge Graph Evolution Via Blockchain. IEEE Transactions on Services Computing (2023)

[168] Wilde, E., Wilde, E.: Universal Resource Identifier (URI). Wilde's WWW: Technical Foundations of the World Wide Web pp. 35–51 (1999)

[169] Willerval, A., Diefenbach, D., Bonifati, A.: qEndpoint: A Wikidata SPARQL endpoint on commodity hardware (04 2023). https://doi.org/10.1145/3543873.3587327

[170] Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**, 1–32 (2014)

[171] Wooldridge, M., Jennings, N.: Intelligent agents: Theory and practice. The knowledge engineering review **10**(2), 115–152 (1995)

[172] Xi, J., Zou, S., Xu, G., Guo, Y., Lu, Y., Xu, J., Zhang, X.: A comprehensive survey on sharding in blockchains. Mobile Information Systems **2021**, 1–22 (2021)

[173] Yao, Y., Kshirsagar, M., Vaidya, G., Ducrée, J., Ryan, C.: Convergence of blockchain, autonomous agents, and knowledge graph to share electronic health records. Frontiers in Blockchain **4**, 661238 (2021)

[174] Zheng, G., Gao, L., Huang, L., Guan, J., Zheng, G., Gao, L., Huang, L., Guan, J.: Application binary interface (ABI). Ethereum Smart Contract Development in Solidity

pp. 139–158 (2021)

[175] Zheng, Z., Xie, S., Dai, H.N., Chen, X., Wang, H.: Blockchain challenges and opportunities: A survey. International journal of web and grid services **14**(4), 352–375 (2018)

[176] Zheng, Z., Xie, S., Dai, H.N., Wang, H.: Blockchain challenges and opportunities: A survey. Work Pap.–2016 (2016)

Annex 1. Sample block in TriG serialisation

```
<http://kg.dlt.linkeddata.es/block/17541191> {
    <uuid:d934bd40-d9d4-4bc1-a10f-33bef78e5812>
            <https://w3id.org/def/Ethereum/baseFeePerGas>
                    12719435246;
            <https://w3id.org/def/Ethereum/difficulty>
                    0;
            <https://w3id.org/def/Ethereum/extraData>
                    "0x7273796e632d6275696c6465722e78797a";
            <https://w3id.org/def/Ethereum/gasLimit>
                    30000000;
            <https://w3id.org/def/Ethereum/gasUsed>
                    15483034;
            <https://w3id.org/def/Ethereum/hash>
                    "0x875edd8a55eb0...";
            <https://w3id.org/def/Ethereum/logsBloom>
                    "0xfee94e53696cb7883ffe...";
            <https://w3id.org/def/Ethereum/miner>
                    "0x95222290dd7278aa3ddd389cc1e1d165cc4bafe5";
            <https://w3id.org/def/Ethereum/mixHash>
                    "0x643f399d5837c40a...";
            <https://w3id.org/def/Ethereum/nonce>
                    0;
            <https://w3id.org/def/Ethereum/number>
                    17540304;
            <https://w3id.org/def/Ethereum/parentHash>
                    "0xfb6a939a41ca05...";
            <https://w3id.org/def/Ethereum/sha3Uncles>
                    "0x1dcc4de8dec75d...";
            <https://w3id.org/def/Ethereum/size>
                    187674;
            <https://w3id.org/def/Ethereum/stateRoot>
                    "0xad58fd2bea07116...";
            <https://w3id.org/def/Ethereum/timestamp>
                    1687499327;
            <https://w3id.org/def/Ethereum/totalDifficulty>
                    "58750003716598352816469" .
            <https://w3id.org/def/Ethereum/hasTransaction>
                    [ <https://w3id.org/def/Ethereum/blockHash>
                              "0x01d20934fa35...";
```

```
                        <https://w3id.org/def/Ethereum/blockNumber>
                                17541191;
                        <https://w3id.org/def/Ethereum/chainId>
                                1;
                        <https://w3id.org/def/Ethereum/from>
                                "0x381e840f4ebe...";
                        <https://w3id.org/def/Ethereum/gas>
                                75000;
                        <https://w3id.org/def/Ethereum/gasPrice>
                                15219435246;
                        <https://w3id.org/def/Ethereum/hasABI>
                                <http://kg.dlt.linkeddata.es/ABI/0xd1d2eb
                                   ...>;
                        <https://w3id.org/def/Ethereum/hash>
                                "0x3a042717db1d0f402...";
                        <https://w3id.org/def/Ethereum/input>
                                "0xa9059cbb000000000000000000000000b2e...";
                        <https://w3id.org/def/Ethereum/maxFeePerGas>
                                26998826826;
                        <https://w3id.org/def/Ethereum/maxPriorityFeePerGas>
                                2500000000;
                        <https://w3id.org/def/Ethereum/nonce>
                                116054;
                        <https://w3id.org/def/Ethereum/r>
                                "0x6db2e03626f...";
                        <https://w3id.org/def/Ethereum/s>
                                "0x2649dbb797b...";
                        <https://w3id.org/def/Ethereum/to>
                                "0xd1d2eb1b1e90...";
                        <https://w3id.org/def/Ethereum/transactionIndex>
                                36;
                        <https://w3id.org/def/Ethereum/type>
                                "0x2";
                        <https://w3id.org/def/Ethereum/v>
                                1;
                        <https://w3id.org/def/Ethereum/value>
                                0
                    ];
            <https://w3id.org/def/Ethereum/hasWithdrawal>
                    [ <https://w3id.org/def/Ethereum/address>
                                "0x2f777e9f26aa1...";
                        <https://w3id.org/def/Ethereum/amount>
                                13994070;
                        <https://w3id.org/def/Ethereum/index>
                                "0x7b631a"
                    ];
}
```

Listing 1: Sample block in TriG serialisation

Annex 2. Sample ABI in TriG serialisation

```
<http://kg.dlt.linkeddata.es/ABI/0\
   x916c6af08bf922eaf80c05975886c0a421c78a35> {
    <uuid:78522ee1-d022-46f5-9ca5-8d28ea9eb009>
            <https://w3id.org/def/SolidityABI/address>
                    "0x916c6af08bf922eaf80c05975886c0a421c78a35";
            <https://w3id.org/def/SolidityABI/hasEvent>
                    [ <https://w3id.org/def/SolidityABI/hasEventInput>
                            [ <https://w3id.org/def/SolidityABI/indexed>
                                    true;
                              <https://w3id.org/def/SolidityABI/
                                 internalType>
                                    "address";
                              <https://w3id.org/def/SolidityABI/name>
                                    "approved";
                              <https://w3id.org/def/SolidityABI/type>
                                    "address"
                            ];
                      <https://w3id.org/def/SolidityABI/hasEventInput>
                            [ <https://w3id.org/def/SolidityABI/indexed>
                                    true;
                              <https://w3id.org/def/SolidityABI/
                                 internalType>
                                    "uint256";
                              <https://w3id.org/def/SolidityABI/name>
                                    "tokenId";
                              <https://w3id.org/def/SolidityABI/type>
                                    "uint256"
                            ];
                      <https://w3id.org/def/SolidityABI/hasEventInput>
                            [ <https://w3id.org/def/SolidityABI/indexed>
                                    true;
                              <https://w3id.org/def/SolidityABI/
                                 internalType>
                                    "address";
                              <https://w3id.org/def/SolidityABI/name>
                                    "owner";
                              <https://w3id.org/def/SolidityABI/type>
                                    "address"
                            ];
```

```
36                          <https://w3id.org/def/SolidityABI/isAnonymous>
37                                  false;
38                          <https://w3id.org/def/SolidityABI/name>
39                                  "Approval"
40                      ];
```

Listing 2: Sample ABI in TriG serialisation

Annex 3. Sample Solidity smart contract code in TriG serialisation

```
<http://kg.dlt.linkeddata.es/Solidity/0
   xf74d5b5a806bf8904337c1738c1583279ba0e27c> {
    <uuid:31dcbca7-cdec-4dbf-a53a-7d4f6f266f2a-LockToken>
            a       <https://w3id.org/def/Solidity#Contract>;
            <https://w3id.org/def/Solidity#contractName>
                    "LockToken";
            <https://w3id.org/def/Solidity#hasContractAttribute>
                    [ a       <https://w3id.org/def/Solidity#
                       NonConstantAttributeSpecification>;
                      <https://w3id.org/def/Solidity#attributeName>
                              "isOpen";
                      <https://w3id.org/def/Solidity#hasAttributeValue>
                              [ a       <https://w3id.org/def/Solidity#
                                 bool> ];
                      <https://w3id.org/def/Solidity#hasAttributeValue>
                              [ <https://w3id.org/def/Solidity#isConstant>
                                      "false"^^<http://www.w3.org/2001/
                                         XMLSchema#literal> ];
                      <https://w3id.org/def/Solidity#
                         hasAttributeVisibility>
                              <https://w3id.org/def/Solidity#Public>
                    ];
            <https://w3id.org/def/Solidity#hasContractAttribute>
                    [ <https://w3id.org/def/Solidity#attributeName>
                              "_whiteList";
                      <https://w3id.org/def/Solidity#hasAttributeValue>
                              [ a       <https://w3id.org/def/Solidity#
                                 MapType> ];
                      <https://w3id.org/def/Solidity#
                         hasAttributeVisibility>
                              <https://w3id.org/def/Solidity#Private>
                    ];
            <https://w3id.org/def/Solidity#hasContractConstructor>
                    [ <https://w3id.org/def/Solidity#constructorCode>
                              "_whiteList[ msg.sender] = true;\n
                                 _whiteList[ address(this)] = true;" ];
            <https://w3id.org/def/Solidity#hasImplementationFunction>
                    [ <https://w3id.org/def/Solidity#functionCode>
                              "isOpen = value;";
```

```
32                      <https://w3id.org/def/Solidity#functionName>
33                              [ a       <https://w3id.org/def/Solidity#
                                   openTrade> ];
34                      <https://w3id.org/def/Solidity#hasFunctionArguments>
35                              [ <https://w3id.org/def/Solidity#
                                   hasParameterName>
36                                        "value";
37                                  <https://w3id.org/def/Solidity#
                                      hasParameterPosition>
38                                        "0"^^<http://www.w3.org/2001/
                                           XMLSchema#short>;
39                                  <https://w3id.org/def/Solidity#
                                      hasParameterTypeWithDataLocation>
40                                        [ a       <https://w3id.org/def/
                                           Solidity#bool> ]
41                              ];
42                      <https://w3id.org/def/Solidity#hasFunctionBehaviour>
43                              [ a       <https://w3id.org/def/Solidity#
                                   Public>;
44                                  <https://w3id.org/def/Solidity#isDefinedAs
                                      >
45                                        [ <https://w3id.org/def/Solidity#
                                           modifierName>
46                                                "onlyOwner" ]
47                              ]
48                  ];
49              <https://w3id.org/def/Solidity#hasImplementationFunction>
50                      [ <https://w3id.org/def/Solidity#functionCode>
51                              "_whiteList[ _users] = _trueFalse;";
52                        <https://w3id.org/def/Solidity#functionName>
53                              [ a       <https://w3id.org/def/Solidity#
                                   includeToWhiteList> ];
54                        <https://w3id.org/def/Solidity#hasFunctionArguments>
55                              [ <https://w3id.org/def/Solidity#
                                   hasParameterName>
56                                        "_trueFalse";
57                                  <https://w3id.org/def/Solidity#
                                      hasParameterPosition>
58                                        "1"^^<http://www.w3.org/2001/
                                           XMLSchema#short>;
59                                  <https://w3id.org/def/Solidity#
                                      hasParameterTypeWithDataLocation>
60                                        [ a       <https://w3id.org/def/
                                           Solidity#bool> ]
61                              ];
62                        <https://w3id.org/def/Solidity#hasFunctionArguments>
63                              [ <https://w3id.org/def/Solidity#
                                   hasParameterName>
64                                        "_users";
65                                  <https://w3id.org/def/Solidity#
                                      hasParameterPosition>
66                                        "0"^^<http://www.w3.org/2001/
                                           XMLSchema#short>;
67                                  <https://w3id.org/def/Solidity#
```

```
                                    hasParameterTypeWithDataLocation>
68                                      [ a       <https://w3id.org/def/
                                           Solidity#address> ]
69                              ];
70                      <https://w3id.org/def/Solidity#hasFunctionBehaviour>
71                              [ a       <https://w3id.org/def/Solidity#
                                   Public>;
72                                <https://w3id.org/def/Solidity#isDefinedAs
                                   >
73                                      [ <https://w3id.org/def/Solidity#
                                           modifierName>
74                                              "onlyOwner" ]
75                              ]
76                  ];
77              <https://w3id.org/def/Solidity#hasImplementationModifier>
78                      [ <https://w3id.org/def/Solidity#modifierCode>
79                              "require(isOpen||_whiteList[ from]||
                                   _whiteList[ to], \" Trading is not Open
                                   \");\n _;";
80                        <https://w3id.org/def/Solidity#modifierName>
81                              "open"
82                  ];
83              <https://w3id.org/def/Solidity#inheritance>
84                      <uuid:31dcbca7-cdec-4dbf-a53a-7d4f6f266f2a-Ownable>;
85              <https://w3id.org/def/Solidity#isAbstract>
86                      false;
87              <https://w3id.org/def/Solidity#version>
88                      "^0.8.0" .
```

Listing 3: Sample Solidity smart contract code in TriG serialisation