Semantic Smart Contracts and Their Integration with Semantic Data Licensing

Zahra Jafari
Supervisor: Prof. Dr. Anna Fensel
Department of Computer Science
Semantic Technology Institute Innsbruck

A thesis submitted for the degree of

Master's Program
Computer Science

2023

Abstract

This thesis addresses the issue of semantic representation of the deployment environment of a smart contract and the integration of semantic licensing from the Data Licenses Clearance Center (DALICC) library. This research is divided into four sections for the following reasons: First, we concentrated on how smart contracts develop on the blockchain and some additional information about this system. Second, we show how blockchain and semantic web technology can work together. It focuses on several methods for indexing and running Ethereum blockchain smart contracts. The third segment, meanwhile, focuses on DALICC as a framework for identifying licensing conflicts and cutting down on the price of rights clearance. In the final piece, we created a DApp to describe a semantic model of these installations and connect semantic licensing with content using smart contracts.

Contents

List of Figures

0.1 Introduction

Distributed ledger on the blockchain has gained popularity recently [24]. The characteristics of blockchain that make it valuable for cryptocurrencies, such as proof of consensus, safe transactions, and transparency, also make it appropriate in many other situations, including supply chain management, healthcare, and social services.

One of the well-known ledgers that can create decentralized applications and manage cryptocurrencies using smart contracts is the Ethereum blockchain [24]. It serves as a platform for programmers to create decentralized applications (DApp), as William cites, by implementing smart contracts in a decentralized paradigm.

A smart contract is a piece of code created in the Solidity programming language, stored on the blockchain, and responsible for executing certain blockchain-based functionality in accordance with the terms of the contract in the most reliable manner [24].

With the widespread adoption of distributed ledger technology like blockchain in a variety of contexts (such as public blockchain, private blockchain, etc.), the requirement for data queries and index entries is becoming more crucial. The integration of data recorded on the blockchain with outside sources is crucial; in other words, connected data is required [24].

The World Wide Web was founded by Tim Berners-Lee, who asserts that "linked data is the semantic web done right, and the web done right" [27]. In addition to enabling access, integration, and query on data sources, the semantic web is a collection of standards that fosters communication between various application domains and produces a variety of insights. Additionally, it can employ ontology to represent Ethereum elements like blocks and transactions to support SPARQL web queries and dataset linking [24].

In this thesis, we address the following issues: The first step is to figure out how to incorporate DALICC library semantic licenses into a smart contract. Data Licenses Clearance Center, or DALICC, is a framework that encourages automated rights clearance in order to detect license conflicts and lower the cost of high transactions associated with human licensing content clearance [19].

The second is that although DALICC licenses are already linked to the blockchain, the records themselves are not semantically represented. As a result, we created a semantic model using the EthOn ontology[1], and mapped it to data that we obtained from de-

[1] http://ethon.consensys.net

ployed smart contracts in RDF format.

In order to solve these concerns, we built a DApp to attach licenses to content, using earlier work on the subject[2] [10]. We then used semantic web approaches to link data taken from Ethereum transactions to relevant EthOn ontology concepts.

To have a deeper understanding of the technology being employed, we discussed blockchain, Ethereum, distributed ledger technology, and some additional relevant subjects in the first chapter. We discussed some research on how the semantic web can be used with blockchain in the following chapter. Additionally, we demonstrated a DApp in the last chapter to illustrate how smart contracts may be integrated with the DALICC semantic licenses library and represented semantically in this deployment.

[2] https://github.com/kilianhnt/dalicc-license-annotator

Chapter 1

Smart Contract and Distributed Ledger Technology

With the development of technology, money transfers take place through networks, and we now need to update database entries that are governed by centralized institutions like banks. All of these procedures call for a third party to carry out the proper business dealings between unidentified parties.
Having third parties can lead to a number of issues, including the control of the entire transaction by a single authority and the invalidation of any transaction that accomplishes its goal. The new concept of a distributed ledger can be used to alleviate all of these problems. With this new technology, third parties are no longer necessary, and all transactions may be completed quickly between parties via a public network.
With the distributed ledger, it is impossible to alter transactions recorded in a ledger or identify the persons involved in these transactions [14].

1.1 Ledger

A ledger is a book or computer that keeps track of financial system transactions. There are two separate ledgers that look like this:
Centralized Ledger includes all recording of transactions involving firm resources, expenses, libraries, etc.
Decentralized Ledger is a database that shares data across the network. It allows transactions to be executed in public. Any participant at each node can have an identical copy of the ledger, which is already shared on the network.
If any change or update occurs on the ledger, each node constructs a new transition

and votes using the consensus algorithm to choose the correct copy of the ledger. Once consensus has been reached, other nodes will be synchronized with the latest version of the ledger [17].

1.2 Distributed Vs. Decentralized

Baran (1964) [2] clarifies the distinction between decentralized and distributed. Decentralized refers to the absence of a central decision-making body. Every participant has the freedom to make their own decisions, and the system collects them all as subsequent behaviors. The process is divided among all participants in the distributed system, but there is no central authority; instead, decisions will be centralized. Decentralized databases are a collection of connected databases that operate autonomously in several locations, which is their fundamental distinction from distributed databases.
A distributed database, according to Ozsu et al. [17], is a group of various, logically connected databases that are dispersed throughout a computer network and provide transparent distribution to all users [17]. According to this description, blockchain technology encompasses both kinds because it functions as a network and appears to users to be a single system. Blockchain is a type of distributed database system [17].

1.3 Distributed Ledger Technology (DLT)

DLT refers to a database that offers participants identical copies of shared data that are updated using a sophisticated consensus mechanism. Costs are decreased, while transparency, traceability, and process speed are all improved.
There are various difficulties associated with this technology, some of which have not yet been solved. Scalability, separability, and data privacy are the three main issues with DLT [22].

How Does DLT Work?

DLT is the result of combining the main three technologies:
- *P2P*: Each participant (node) functions both as a client and a server at the same time, using and contributing resources.
- *Cryptography* is used to authenticate the identity of the participant and the information

between the two parties. Encryption helps in limiting access to information by other parties.
- *Consensus algorithm* allows network participants to come into agreement to add a new node (block) to the ledger [22].

1.4 Blockchain

Blockchain is the most widely used distributed ledger, according to the World Bank Group [16], that stores and distributes data in units called "blocks." Each block's header includes details including a nonce, timestamp, block hash, and a hash pointer to the block before it. As a result, a digital chain is formed by connecting all of these pieces. [16].
Blockchain is described by Luke et al. [13] as a collection of blocks that are connected to one another and encrypted. An identical copy of these records is kept locally on the PCs of network members. When a user requests a transaction, whether it be a transaction, a contract, or other information, the blockchain begins processing.
A P2P network of nodes broadcasts the transaction. Following then, the P2P network's nodes engage in a process known as verification in which they collectively check the transactions using hashes produced by an algorithm. Details of the transaction will be saved in a block after verification is finished. Finally, a new block is permanently and inevitably added to a chain [13]. The *Genesis* block is the first block in the blockchain, and more nodes will be added to the chain once all nodes have reached consensus. The blockchain can expand without concern for information in the blocks being altered thanks to the consensus method. Since the blocks include transactions, the consensus procedure lasts for a set amount of time. When a transaction is added to a blockchain, there is a time lag between when the transaction was initiated and when it was added. Based on the block size, the number of transactions, and the consensus process, this confirmation time varies. There are several consensus mechanisms stated, including:

- Proof of Work (PoW): It is a system that promotes consensus without relying on a centralized authority. This method pits miners against one another to see who can enter their transactions into the blockchain first and earn incentives (such Bitcoin or Ether).

Anyone who completes their assignment earlier can add their block first on the blockchain. Miners are participants in cryptocurrency transactions that connect to the blockchain and carry out tasks, validating transactions to add new blocks by solving a cryptographic challenge [21].

- Proof of Stake (PoS): It is a proof-of-work substitute that uses less CPU (Central Processing Unit) computation during mining. The likelihood of mining the following block in proof of stake relies on node balance.
  However, consensus procedures like proof of work are not necessary for private networks when users are acquainted. This specifically eliminates the requirement for mining and allows us access to a wider range of consensus procedures [4].

- Proof of Authority (PoA): It confirms accounts and allows them to add transactions in blocks. Compared to the previous approaches, this one is more vulnerable to attack because it is much more centralized and has higher transaction speeds [13].

- Practical Byzantine Fault Tolerance (PBFT): Byzantine Generals are some network users who communicate transaction-related information to others incoherently or fraudulently. Blockchain seeks to address this issue.
  This results in the unreliability of blockchain because there is no authority on it to rectify them. The PBFT algorithm leverages the idea of primary and secondary votes to try to reach a consensus in order to resolve this issue. If the primary vote is corrupted, the secondary vote can collectively change to a new primary after automatically evaluating the decisions made by the primary vote [13].

Blockchain is associated with cryptocurrencies like Ethereum, Bitcoin, Litecoin, etc. Gupta (2017) [8] identified five core attributes through which blockchain builds trust:

- Distributed ledger: No single authority has control over the data. It is distributed and updated throughout the network, and any new changes are distributed to all users.

- Organized and adaptable: Because smart contracts can be carried out on the blockchain, the technology may be developed to support company operations.

- Transparent and auditable: Since everyone has access to the same ledger, can confirm transactions, and can determine the owner, there is no need for a third party or other authority.
- Secure, private, and indelible: Blockchain offers these advantages by making use of tools like cryptography and permissions, which make sure that unauthorized users can't access the network. It indicates that participants are who they say they are.
- Consensus: To validate the transition, all nodes on the network must concur, and the blockchain implements this process via a consensus mechanism [8].

Type of Blockchain Blockchain is used to transmit information across a secure network, according to Aithal et al. [18]. Public and private networks were the two main types of blockchain technology. Blockchain technology can alternatively be categorized as consortium blockchain technology or hybrid blockchain technology, according to further investigation [18].
All blockchains, it should be mentioned, are node-based and run on P2P (peer-to-peer) networks. According to Aithal et al. [18], there are three different kinds of blockchain: consortium blockchain, private blockchain, and public blockchain. The hybrid blockchain is an additional type of blockchain that exists [18].

- **Public Blockchain** is the main variety of blockchain that is open and decentralized by nature. Anyone is allowed to join the network and establish consensus in this system. Any miner (participant) on a public blockchain can develop consensus techniques like proof of work and proof of stake to validate transactions with a low validity rate [11].
- **Private Blockchain** is not open and restricted in this way that Only authorized participants have the ability to validate transactions. As a result, it offers improved privacy, enhances scalability, and reduces security concerns. All participants in this network are known, hence there are no mining computations necessary for this blockchain to establish a consensus [11].
- **Consortium Blockchain** is a semi-decentralized blockchain that is used to carry out operations for just one company, such as a bank, etc. A consortium blockchain differs from a private blockchain in that it is managed by multiple parties rather than just one [18].

- **Hybrid Blockchain** is a combination of private and public blockchain. As a result, it combines the advantages of a private blockchain's privacy with the security and openness of a public blockchain.
  The user of this kind of blockchain can regulate who has access to what data on the blockchain. On a private network, a transaction can be validated before the user releases it to the public blockchain. By doing this, only a portion of the records can be made public while the remainder can be kept secret on a network [18].

1.5 Ethereum

Currently, Ethereum is the world's most active public blockchain. It is a different cryptocurrency that is based on the blockchain and is comparable to Bitcoin. The transactions are published by the users on the network, which uses a consensus mechanism to store them in blocks and add them to the blockchain. In Ethereum, "state" refers to the current status of the various accounts that make up the blockchain. In Ethereum, a user's account may either be an unrelated external account or a contract account that receives ongoing blockchain storage. *Ether* serves as the system's virtual money. By establishing a new contract or triggering an already-existing contract, the transaction has the ability to alter the system's state [7].

1.6 How Does Ethereum Work?

In this subsection, we will focus on the Ethereum workflow at a technical level.

1.6.1 Blockchain

The blockchain contains some information that we have utilized in our project. Therefore, we focus on it more, as follows:

- **Block** as a data structure within the blockchain, contains different functions, which include transaction hashes and some other additional information for blockchain technology. following information was described by Gavin Wood et al. [29] and we

[1]https://archive.researchworld.com/three-ways-that-blockchain-can-improve-the-quality-of-market-research/

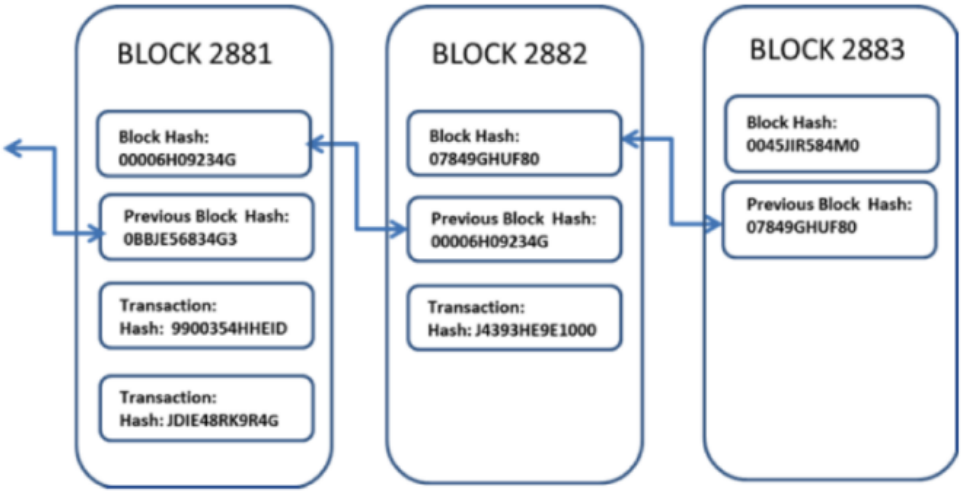


Figure 1.1: Block contents[1]

used it in our project:

- *parentHash* is the hash of the parent block's header.
- *stateRoot* is the hash of the root node of the state after execution of all transactions is applied.
- *transactionRoot* is the hash of the root node of data populated with a transaction in the transactions list inside the block.
- *receiptRoot* is the hash of the root node of the data populated with the receipts of transactions in the block.
- *logsBloom* composed of log information.
- *difficulty* represents the difficulty level of the block.
- *number* is the number of ancestor blocks.
- *gasLimit* represents the current limit of gas in the block.
- *gasUsed* is the amount of gas used for the transaction in the block.
- *timestamp* is the time of reasonable output.
- *extraData* is a byte array containing relevant information in the block.

- *nonce* is the number of several computations that have been done in the block.

- **Mining** is a process of computation on the blockchain to verify and add a block. A new block is added by the miner, and then it is verified by others. Anyone can join the mining pool, however, the likelihood of discovering a valid block relies on the computer's processing capability. A miner may occasionally discover an uncle block. "An uncle block" is a legal block that is eventually surpassed by a faster block. A hash will be added to a valid block, and an Uncle block is rewarded with $\frac{7}{8}$ of the total block value. A valid block may have up to two uncle blocks added to it, and each uncle block earns the valid block's miner an additional $\frac{1}{32}$ worth of ether [28].

1.6.2 Ether

It serves as both Ethereum's fuel and a means of payment. In order to successfully mine (find the solution and add blocks), you need five ethers. It becomes less ether, such as 4.375 ether, and becomes an uncle block if the miner finds a solution but does not act quickly enough. There can only be two uncle blocks per block, and each uncle block is paid $\frac{1}{32}$. If a second miner also discovers the answer, the block cannot be added to the blockchain, and the first miner only gets a reward of a couple of ether [28].

1.6.3 Account

There are two types of accounts in Ethereum:
- *Normal account* is controlled by the private key. The owner of this account can send either a message.
- *Contract* account is controlled by code. It can only fire a transaction in response to other transactions [28]. An account encompasses four fields:

- *Nonce* is the number of transactions sent from this address [29].

- *Balance* is the number of Wei owned by this address [29].

- *StorageRoot* is the hash of the root node of the Merkle Patricia tree, which encodes the content of an account. It should also be noted that the Merkle tree is used for data representation in the block header [29].

- *CodeHash* is the hash associated with this account that would be executed when this account address receives a message call and would no longer be changeable. For later retrieval, all information pertaining to this account is kept in the database under the associated hash code [29].

Hash Function At the National Institute of Standards and Technology (NIST), the SHA3 standardization procedure was finished in August 2015. The Secure Hash Algorithm-3 (SHA-3) family of functions for binary data is described by this standard. The *Kacak* algorithm, also known as NIST and the winner of the SHA3 cryptographic hash algorithm competition, serves as the foundation for each hash function. There are four functions in the SHA3 family, each having a distinct length of 224, 256, 384, or 512 bits. The input to the hash function is referred to as a *message*, and the output is referred to as a *hash value*, where the message's length might vary but the hash value's length is always constant [5].

$$h: M \rightarrow \{0,1\}^n, with\ h(m) = \hat{m}$$

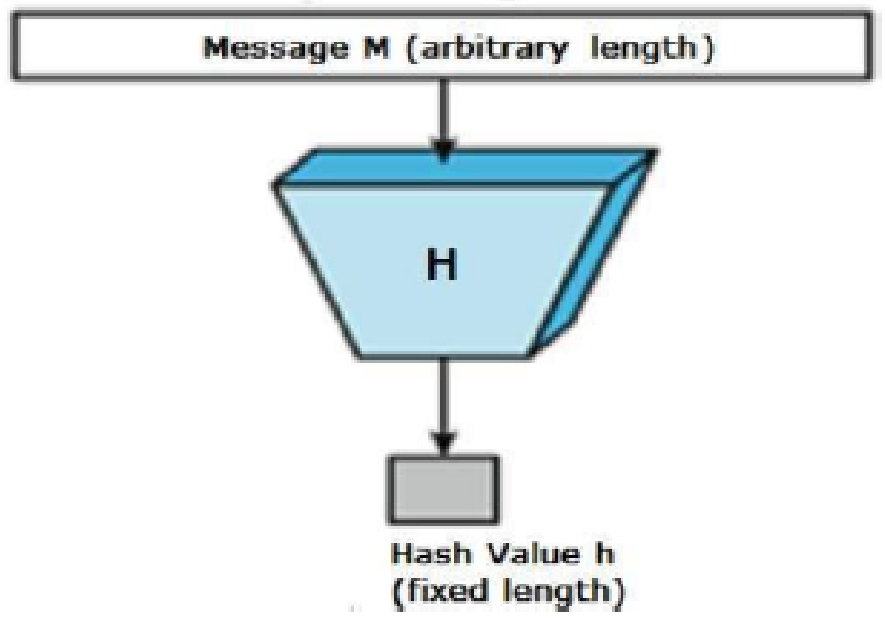


Figure 1.2: Image of hash function [5]

On the Ethereum network, gas is the fuel that must be purchased prior to completing a transaction. The consumed gas will not be refunded if the transaction is rolled back [28].

Transaction A transaction is a cryptography-signed instruction that is carried out by an external actor, which can be a person or another contract, according to Gavin Wood et al. [29]. The transaction describes these fields:

- *Nonce* is the number of transactions sent by the sender.
- *GasPrice* is the number of wei to be paid per unit of gas.
- *GasLimit* is the amount of gas that can be used for transactions.
- *To* is the address to which that contract sends a transaction.
- *Value* is the number of wei that is transferred in the transaction.

Message is sent by a contract and has the same attributes as a transaction, but $gasPrice$ is not included [28].

1.6.4 Contract

It is an Ethereum blockchain account that is managed by code and has its own code. Every time a message is received, the contract's internal code is activated, enabling it to read and write contract storage or send messages.
A contract in Ethereum is an autonomous agent that executes when it receives a message or transaction and has control over its balance and the key/value stored as constant variables. This means that the contract is an autonomous agent that carries out some operations that are programmed to achieve the user's goals.
The contract's long-lasting keys and values are used when the contract begins to function [28].

1.6.5 Smart Contract

Nick Szabo[23] first popularized the phrase "smart contract" in 1994 after realizing that DLT might also be used for them.

The definition of a smart contract is given by Nick Szabo[2]: *Smart contract is a computerized transaction protocol that executes the terms of a contract.* He had an idea for how to design contracts that automatically and effectively enforce the terms between the parties to a transaction.
Each node executes a smart contract as part of the block-building process. When a transaction occurs on the blockchain, a block is created. Each smart contract has its own address, which is an essential component. A node can construct a specific transaction by giving it a contract address and carrying the contract code, which makes it possible for this transaction to execute the contract code at the time of formation. The address won't ever change after that because the contract will become a permanent part of the block. A message should be sent to the contract's address that contains the method and input data whenever a node wants to invoke a method inside the contract.
When a new block is created, the contract will execute as part of that process and then either return value or store data on the blockchain [9].

Solidity is a high-level, Turing-complete language with a syntax resembling JavaScript. The contract is comparable to classes in object-oriented programming languages, which have fields for persistently storing contracts and methods that both internal and external transactions can call. Either a new instance of this contract must be created or a transaction must be made to a known contract address in order to interact with another contract.
Solidity, in theory, offers certain fundamentals for getting at blocks and transaction information, such as *msg.sender* for getting at an account's address or *msg.value* for getting at the amount of *wei* transmitted by the transaction. Additionally, it employs several features, such as *call* and *send* to transfer money to another contract. These functions are used to transfer value and translate it into internal calls to transactions, which cause the contract to run code in addition to transferring value or may fail to run owing to a lack of gas [7].

[2]https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature-/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html

Chapter 2

Blockchain as the Infrastructure of Semantic Web

In more and more places throughout the world, transactions involving numerous parties are represented by distributed ledgers. Without an index, it is challenging to search for specific data in a distributed ledger. As a result, indexing data in the distributed ledger is a prerequisite that gives users the capacity to search across many ledgers, hence boosting the system's functionality and usability [24].

2.1 Distributed Ledgers and Indexing

A distributed ledger built on a blockchain lacks centralized management. Multiple blocks make up a blockchain; the first block is manually made, and subsequent blocks are added by some sort of consensus mechanism amongst nodes.

With the help of an Ethereum smart contract, it is possible to autonomously handle transactions on the blockchain without involving any dubious third parties. The accounts used by Ethereum smart contracts often have the ability to store data, change it, or carry out operations using input and output. Data must be indexed since smart contracts are time-ordered and hold information in blocks. By indexing the smart contract, we are able to explore, evaluate, and utilize the distributed ledger's services, as well as expose them to the outside world for more interactions.

There are various indexing levels for smart contracts: The fundamental level for the following step is the basic level. Data can be stored or accessed here, and it indexes fundamental distributed ledger entities like accounts and blocks. Smart contracts have many

functional interfaces that represent the additional capabilities of systems like Ethereum at the functional level [24].

2.1.1 Why Do We Use Ontology for Blockchain?

Blockchain often uses a cloud computing architecture and is a distributed database that is copied across all nodes. In other words, the database contains data that is dispersed among numerous organizations. In order for data to be properly understood by organizations, there needs to be a standard interpretation. These interpretations are applicable through a formal specification that allows for verification and inference in network-based software and applications.
Here, ontology is crucial in ensuring that various enterprises have a common interpretation of the data in the shared database.
Blockchain as a modeling form uses a different type of ontology [12]:

Informal/Semi-Ontology facilitates search and enhances a better understanding of the business process for developing and applying on the blockchain [12].

Formal Ontology helps the formal specification in automating inference and validating the blockchain's functionality. In other words, formal ontology-based blockchain can help in the creation of smart contracts that run on the blockchain [12].
Ontologies can also be used to store data in the blockchain: On the one hand, it helps people grasp blockchain concepts better. On the other hand, it allows for the interlinking with other linked data to convey formal reasoning and inferences, [12].
By describing the transaction in the context of connected data and enabling the graphical representation of the location of such transactions, vocabulary employed inside an ontology increases the transparency of transactions. Consequently, it also improves users' analytical capacity [12].

2.1.2 Linked Data

According to Tim Berners-Lee et al., "linked data" is the creation of typed linkages between data from various sources using the Web [3]. When information is presented as linked data, it is easy to obtain relevant additional information and link new information

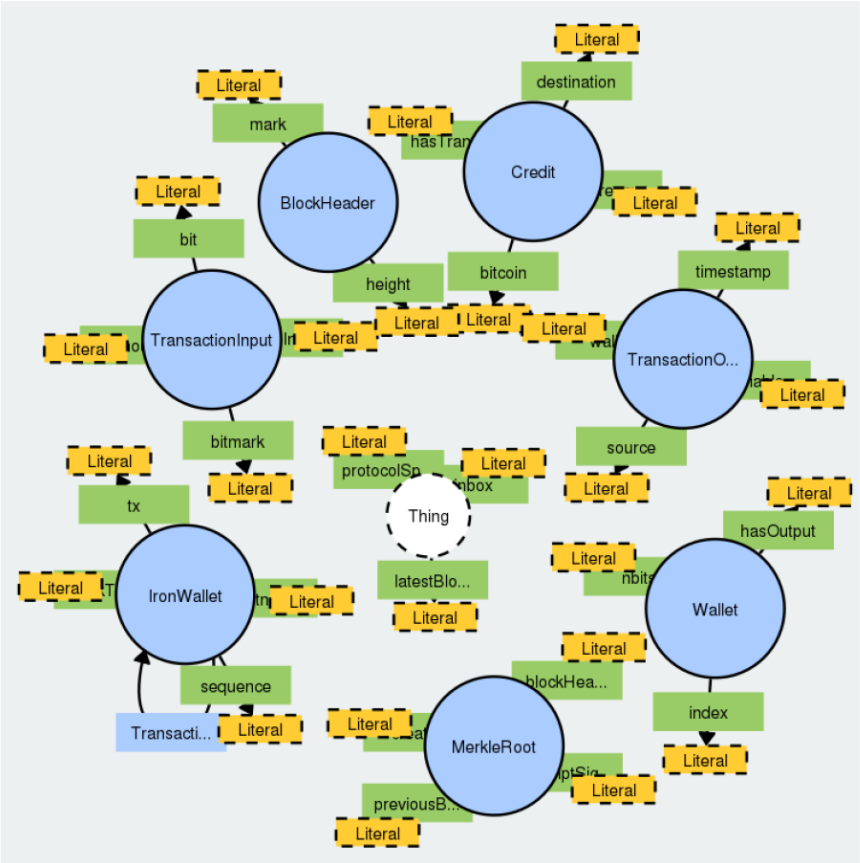


Figure 2.1: Illustration of ontology diagram [6]

to it. *Berners-Lee* described four rules for linked data:

- **URIs** (Uniform Resource Identifier) as names.
- **HTTP** to search for names.
- **SPARQL, RDF** provide related information about what a user is looking for.
- **Link** to other URLs to provide more information [27].

2.1.3 RDF

A group of W3C specifications is known as the Resource Description Framework (RDF). Information is described and modeled using RDF. It is referred to as a triple and describes

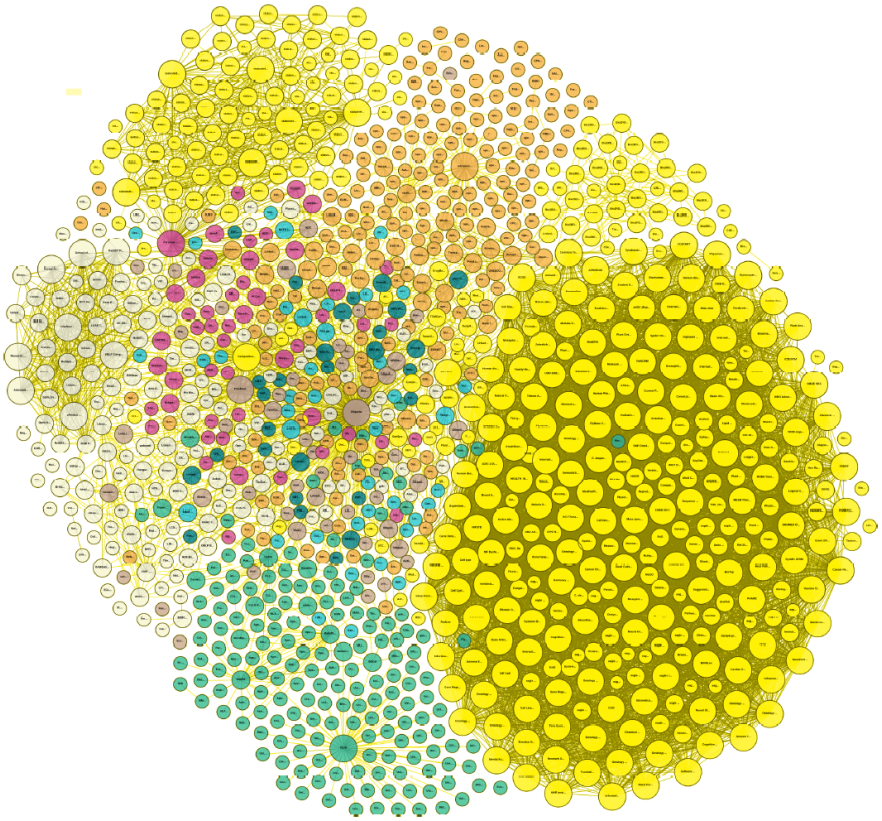


Figure 2.2: Linked data diagram [27]

a topic that predicts an object. i) The topics described by RDF expressions. ii) Mention certain characteristics, traits, or connections when describing a resource. The object's name refers to a property or set of values. [27].

2.1.4 SPARQL

It is a semantic query language for a dataset that enables us to retrieve and alter data stored in RDF format known as triples, according to Wikipedia's definition. One, two, or all of a triple's elements can be queried using SPARQL [27].

2.1.5 OWL (Ontology Web Language)

Ontology Web Language is intended to represent information about objects and their relationships. Because OWL is built on computational logic, its language models can be

used in computer programs to perform operations like negation and intersection [27].

2.1.6 Evolution of World Wide Web

Based on three technologies, the development and interaction of users on the Internet are categorized:

- *Web 1.0*, also known as *web of document*, is the earliest website with the basic capability of linking to other websites.

- *Web 2.0*, known as *web of data*, enables users to participate in the creation or alteration of content.

- *Web 3.0* is strengthened by the Semantic Web, which gives users access to linked information on the web. Recently, there has been a fresh emphasis on this due to the introduction of distributed technologies like blockchain and Ethereum, which are employed by Web 3.0 [1].

2.2 Vocabularies

2.2.1 Vocabulary in Distributed Ledger

A common ontology or vocabulary must be used to describe blockchain concepts in order to create linked data. The semantic web and distributed ledger interfaces are still in their infancy. Systems and vocabularies like FlexLedger, EthOn, and BLONDiE specify such a language [24].

In FlexLedger, HTTP interfaces to blockchains are described together with common language and responses. It is a protocol that represents ledger creation, querying, and data modeling using JSON-LD for decentralized ledger and graph data models. However, neither FlexLedger nor its own internal ontology has any specific language related to ontologies.
Because the FlexLedger meta and the content, data are stored together in the same

graph, whereas the GraphChain blocks content is saved outside the blockchain in a different graph, it is significant to highlight that the FlexLedger is not suited to implement in particular graph models like graph chains [26].

EthOn is an OWL ontology that describes blockchain classes such as *'blocks', 'accounts', 'messages', 'states'* and relations such as *'has parent block'* [25].

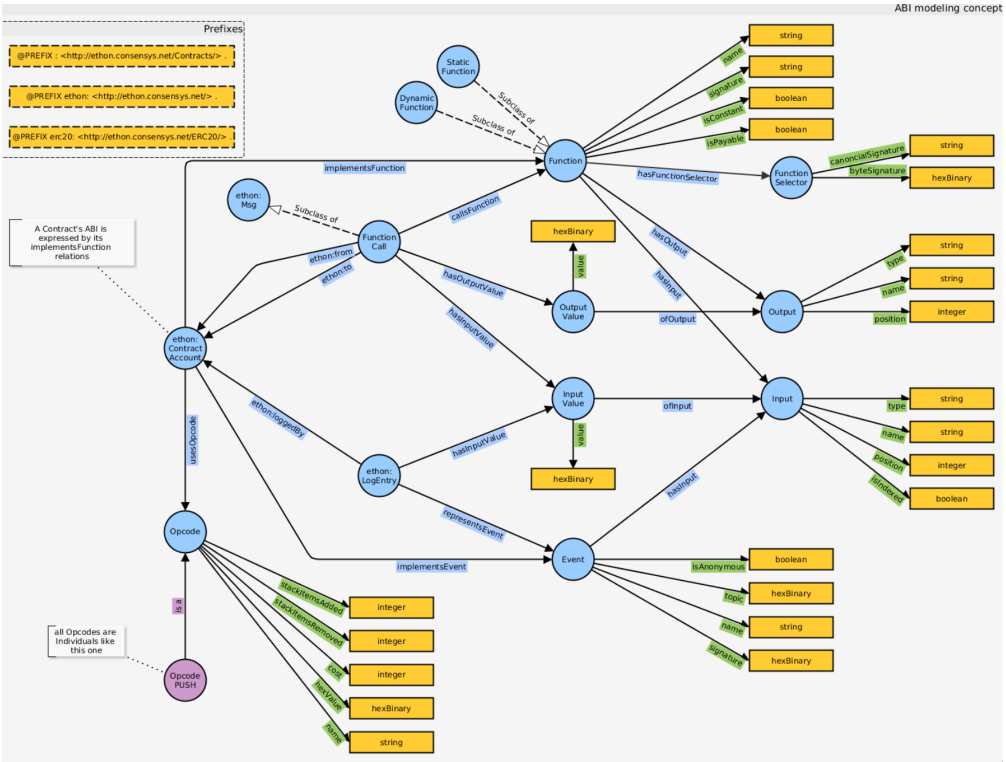


Figure 2.3: EthOn contract model (the blue arrow is object properties, the green arrow is data properties, the purple circle is an instance and blue one is a class) [25]

BLONDiE (Blockchain Ontology with Dynamic Extensibility) is another OWL ontology that describes the structure of a blockchain similar to EthOn. However, it is less specific than EthOn. Examples include the definitions of terminology like "account," "block," and "transaction" as well as some properties like "transaction payload" or "miner address" in EthOn and BLONDiE. Other terms for various blockchains, such as "BitcoinBlockHeader" and "EthereumBlockHeader," are defined by BLONDiE as subclasses of

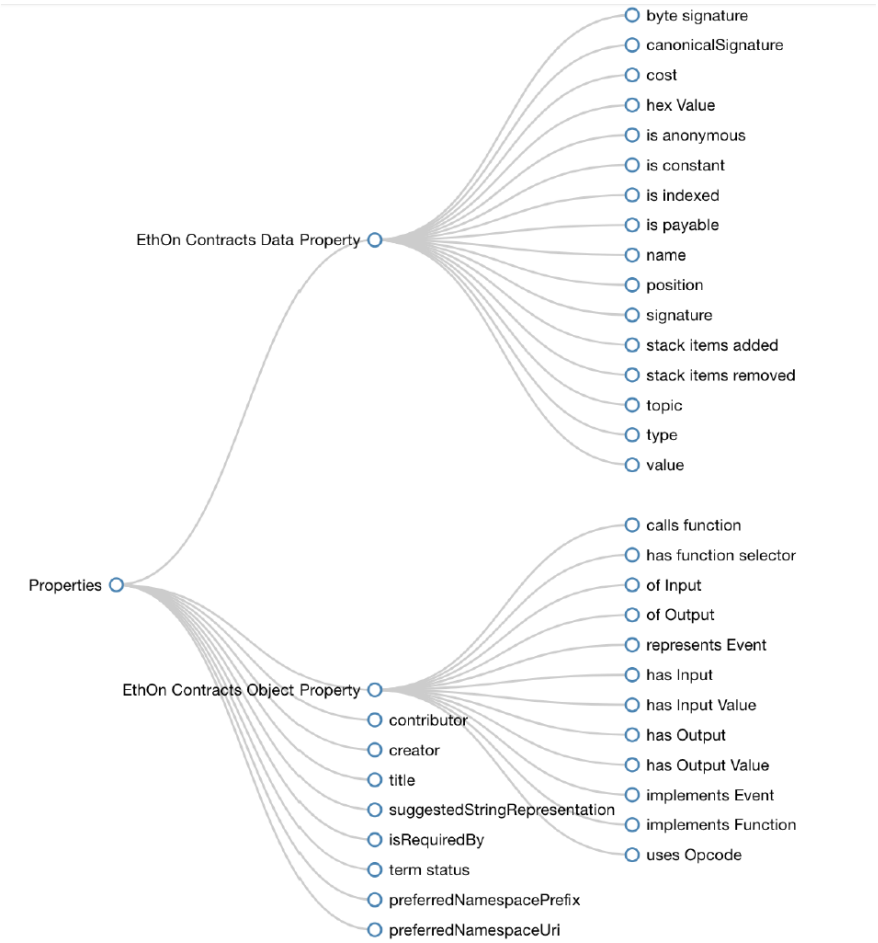


Figure 2.4: EthOn properties [25]

"BlockHeader." Currently, BLONDiE supports two cryptocurrencies, such as Bitcoin and Ethereum, in which all connections and relationships among objects and characteristics are expressed using the RDF (Resource Description Framework) format [24].

2.2.2 Vocabulary in Smart Contract

As was already noted, two concepts that can be employed for smart contracts are EthOn and BLONDiE. We may be able to annotate smart contracts thanks to several works on HTTP-APIs and semantic annotation of the web. Although the implementation may vary and the contracts are not on web APIs, the fundamental idea remains the same.

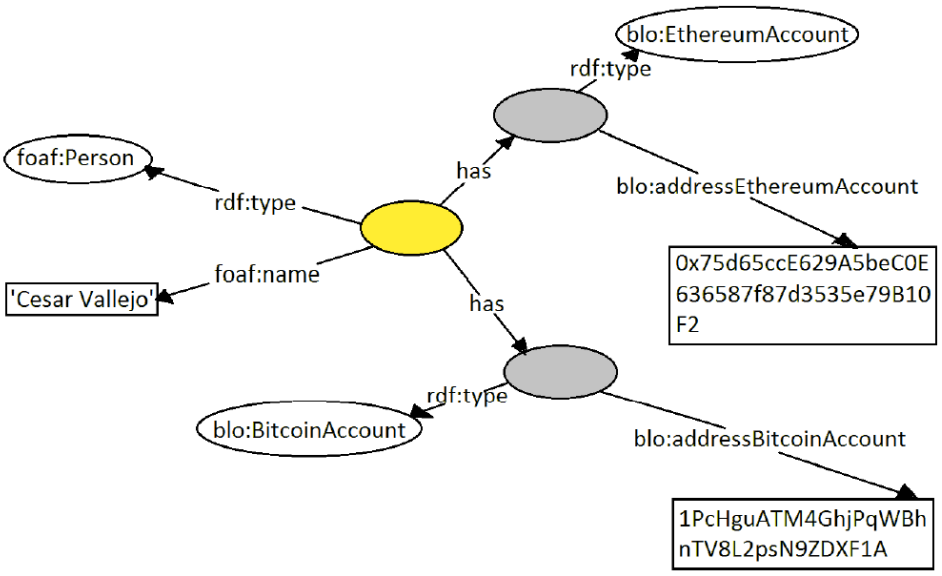


Figure 2.5: BLONDiE usage example [27]

In other words, smart contracts are annotated using the same vocabulary as are used to annotate web services. Due to their profitability, it appears that distributed ledgers, smart contracts, and web services will frequently be combined [24].

2.3 Semantify Blockchain

2.3.1 Semantic Blockchain

Recent growth in the adoption of blockchain technology has increased the demand for semantic reasoning on the distributed ledger as well. To implement semantic web principles in this technology and add a new trusted attribute to a dataset, the blockchain is the best platform.

The idea of using semantic web technology with blockchain is new, and how to implement it with smart contracts is also up for debate

The following are some **definitions of semantic blockchain:**

- Semantic blockchain is the application of semantic web standards on the blockchain; these standards are based on RDF.

-Semantic blockchain is the representation of stored data on a distributed ledger using

linked data [27].

2.3.2 Semantification Process

The industrial world is impacted by semantic blockchain or semantic distributed ledger, and as a result, new applications and frameworks are being developed to bridge the two worlds.

There are some ways to semantify blockchain, as follows:

- Converting blockchain data to RDF while using vocabulary, ontology, and other tools. Due to the high cost of data storage in blockchains, the only viable option is to first save the hash point to the dataset in the blockchain before sharing RDF.

Building a semantic blockchain to communicate internal data protocols in RDF [27].

2.3.3 Semantic Ontology Mapping Using BLONDiE

The basic blockchain elements must be mapped to the pertinent semantic web terms, concepts, and ontologies in order for the system to produce RDF. The BLONDiE query has been enhanced in two ways to improve efficiency: First, an attribute for each entity's hash has been added to records pertaining to blocks and transactions. Second, records pertaining to the transactions have been enhanced with references to other objects, such as blocks or smart contracts.

Blockchain only keeps a binary form of each contract together with its associated metadata. The Application Binary Interface (ABI) standard is necessary in order to communicate with the contract. This specification, which is formed when a smart contract is put together and kept on the blockchain, is in the form of JSON. The ABI establishes all contract functions and offers descriptions of each contract's input and output parameters [24].

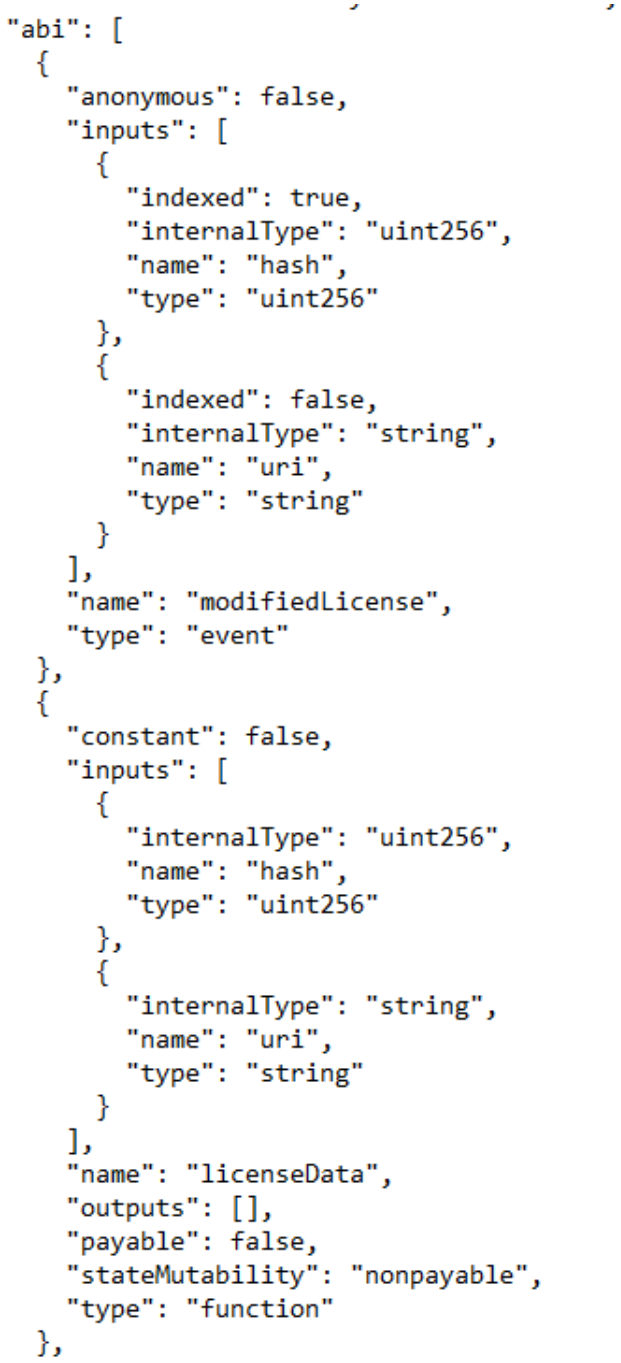


Figure 2.6: Smart contract application binary interface (ABI) [24]

Chapter 3

DALICC

Data Licensing Clearance Center, according to Pellegrini [20], is known as DALICC. It is a software framework that enables automatic rights clearance throughout the production of derivative works, assisting users in protecting data from unauthorized third parties, identifying license problems, and lowering the cost of rights clearance [20].

Rights Expression Language (RELs) These are used to convey rights in a machine-readable format for digital assets and access control. The most widely used RELs are MPEG-21, ODRL-2.0, ccREL, XACML, and WAC, which are used for rights management in a variety of contexts such as data licensing, applications, etc. [19].

3.1 DALICC Requirements

The following requirements would be addressed by the DALICC framework:

- *Tackling license heterogeneity:* Combining different contents with the same license but different names is possible. But in that case, obtaining a license for the resulting contents would be much more challenging. DALICC offers a set of machine-readable representations of licenses that address this problem by enabling us to compare licenses to one another and identify equivalent licenses. It directs the user to potential inconsistencies between numerous combined licenses [19].
- *Tackling REL Heterogeneity:*
  If licenses are represented using the same RELs, combining them is easy. Compar-

ing licenses that have been represented by various RELs is challenging, however. This issue is fixed by DALICC by presenting RELs based on the Semantic Web and mapping the terms to one another. It will reflect current RELs based on W3C-approved standards, enabling the creation of mapping between different RELs [19].

- *Compatibility Check, Conflict Detection, and Neutrality of the Rules:*
  It might be challenging to determine whether the various terminologies used in semantics have the same meaning. These issues arise from listing the classes, instances, and properties that are incompatible with simple mapping [19].
  Here, DALICC steps in to assist the user with a procedure that establishes the usage context before gathering more data to find conflicts and unclear ideas. Based on this data, DALICC analyzes the collection of licenses and deduces directions for the user to follow while processing licenses [19].

- *Legal Validity of Representations and Machine Recommendations:*
  The semantic intricacy of licensing concerns necessitates that the semantics of RELs be in line with the particular application scenario, according to Pellegrini et al. [19]. This includes interpreting the various national laws correctly based on the country of jurisdiction (e.g., German Urheberrecht vs. US copyright), solving multilingualism-related issues, and taking into account existing case law when resolving licensing disputes [19].
  DALICC examines the legality of machine-readable licenses and the conformity of reasoning engine output with the legislation to address this issue. The output of DALICC is being compared to the law, evaluated for semantic accuracy generated from several languages, and changed as necessary [19].

3.2 DALICC Software Architecture

To address the above challenges, the DALICC framework consists of four components:

- **License composer** is a tool that enables the license to be generated from a set of questions that are mapped to the vocabulary and concepts of the ODRL, ccREL, and the DALICC.

- **License library** is a repository that represents licenses in plain text and ODRL policies in a machine-readable format.
- **License annotator** enables adding licenses to the dataset by selecting them from the license library's selection or by writing a new license using the license composer.
- **License negotiator** serves as the primary element of the DALICC system, ensuring licensing compatibility and resolving license conflicts by identifying equivalence licenses with different names [19].

Chapter 4

Implementation: DApp

DApp is frequently characterized as P2P, trustless, with the unique quality that it cannot be controlled by a single server. DApps have at least one user interface or frontend, which could be a desktop program running on a PC or a mobile app. A single organization or business may give the application's data, or end users may do it directly. DApp processes and stores data using smart contracts on Ethereum as the backend. DApp user interfaces typically resemble standard websites, but they also include one or more smart contracts [15].

4.1 Ethereum DApp

The benefits of using the Ethereum blockchain in DApps are as follows [15]:

- 1- The user can see what is going on before submitting any data.
- 2- Once the user has interacted, no one can tamper or delete data.
- 3- The user of the application can directly participate in application management.

4.2 Project Concepts

In this section, we focused on the requirements that we need to address in our application.

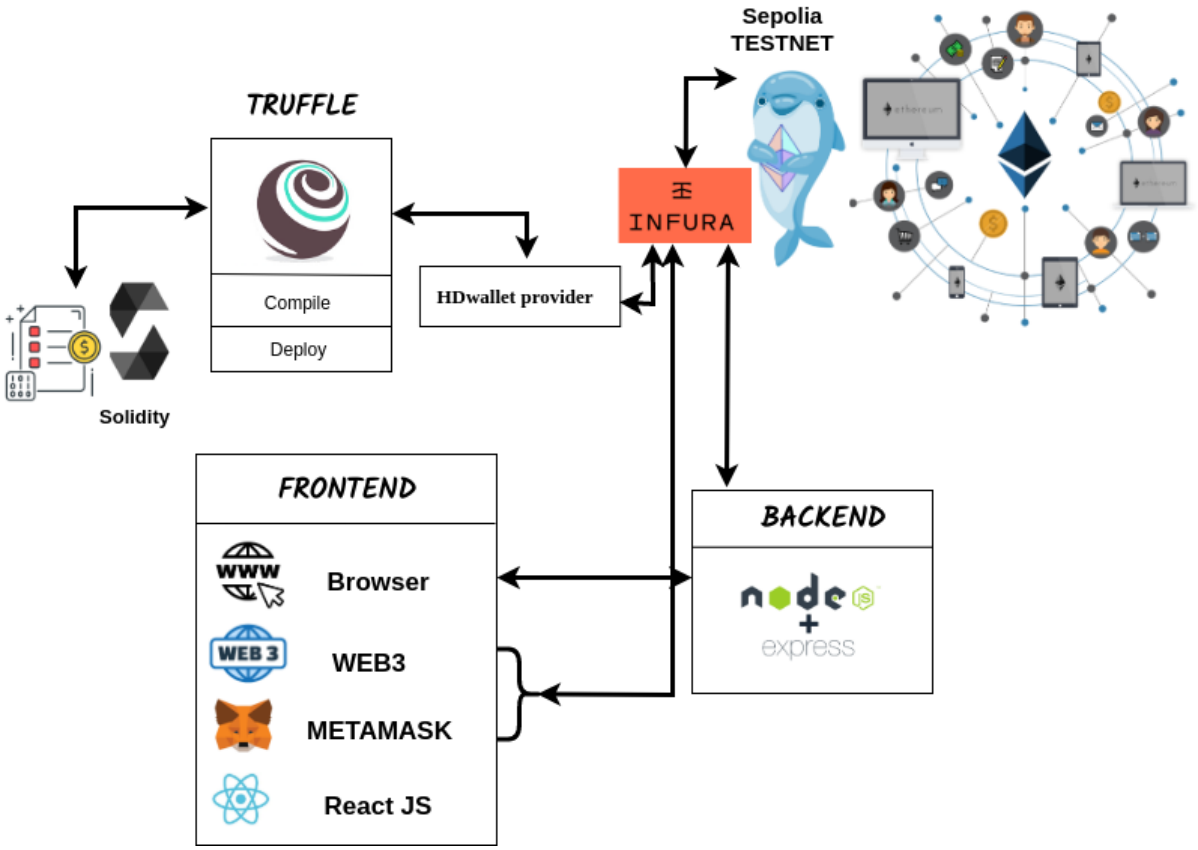


Figure 4.1: DApp infrastructure

4.2.1 Contract and Ontology Specification

Here, we explain some terms related to smart contracts and the ontology of our application. These are needed as the primary requirement to build up this dApp.

- **Definition 1 (Owner)** is the address of a deployed contract or transaction on the blockchain. This is the first person who interacts with the contract. In this case, the owner can change the license of the file.

- **Definition 2 (Non-owner)** is another user that makes no action. In this case, they can retrieve license information related to specific files.

- **Definition 3 (Licensor)** is the address currently interacting with the smart contract.

In this case, they can license their file.

- **Definition 4 (Semantic mappings)**: As we do not have the authority to change stored data on the Ethereum blockchain, one idea is to create a template to have a semantic view of the blockchain. To reach this purpose, we need a mapping between stored transaction data in the blockchain and transaction concepts defined as an ontology, then make a query on the produced data from this mapping. This process consists of three following requirements:

  - *Transaction Schema* are actually some attributes resulting from web3.eth functions that EthOn data properties would be mapped to.
  - *Transaction Triple template* defines the relationship between concepts (block or transaction) and properties. It is known as 'Subject predicts Object' where the Subject is the web3.eth properties, predict is the EthOn ontology predictor (data properties) and Object (web3.eth properties) is a place that would be replaced by the transaction schema on the mapping.
  - *Triplize* is the function that generates data in RDF format by creating a mapping between the two parameters mentioned above as input.

- **Definition 5 (Prefix)** Name is the label or local part separated with ':' and is the abbreviation of terms that reference resources explicitly. According to Wikipedia, Prefixes are declared at the top of the SPARQL query and our triple template, so that the statements can refer to.
  Definition 6 (Triple) Defined in Wikipedia as a statement with three entities that codifies semantic data in the form of *Subject-predicate-Object* expressions.

4.2.2 Technology Usage

DApp[1]

Based on an internet definition, it is a type of open-source smart contract application that runs on a decentralized peer-to-peer network rather than a centralized server. It allows users to transparently execute a transaction on a distributed network.

[1] https://www.techtarget.com/iotagenda/definition/blockchain-dApp

DApp is similar to centralized apps, as they use frontend and backend. The backend of an app is supported by a centralized server or database, whereas the backend of a DApp runs on a decentralized P2P network and is supported by smart contracts stored on the Ethereum blockchain.

Decentralized Application Characteristics:
Open Source: All the requirements are decided by consensus of all available users on a network.
Decentralized Storage: Data is stored on decentralized blocks.
Validation: As the application runs on blockchain, they offer validation of data using cryptographic tokens which are needed for the network.

Ethereum
It is explained in section 1.4.

Solidity
It is explained in section 1.5.5.

SHA3-256
It is explained in section 1.5.3.

Java Script Tools

- `Truffle` is the smart contract development tools and testing network for blockchain applications and supports developers who are looking to build their dApp, etc. Truffle offers some different features:
  - *Smart Contract Management*: Truffle helps to manage artifacts of smart contracts used in dApp and supports library linking, deploying, and other Ethereum dApps.
  - *Contract Testing System*: Truffle helps developers construct smart contract testing systems for all their contracts.

- *Network Management* helps developers by managing their artifacts.

- *Truffle Console* allows the developer to access all Truffle commands and build contracts[2].

- `React.js` is an open-source JavaScript framework which is used as a frontend. In React, a developer builds a web application by using reusable individual components that are assembled from an application's whole user interface.
  React has the advantage of providing a feature that combines the speed of JavaScript with a more efficient method of managing DOM to render web pages faster and create more responsive web applications[3].

- `crypto.js` is the JavaScript library that performs data encryption and decryption. It is a collection of standard algorithms including SHA3-256[4].

- `Web3.js` helps developers to connect to the Ethereum blockchain. It is a collection of libraries that allows developers to perform such actions as sending ether, checking data from smart contracts, and creating smart contracts[5].

- `axios` provides HTTP requests from the browser and handles request/response data[6].

- `HDWalletProvider` is a package that helps developers to connect to a network by configuring the connection to the Ethereum blockchain through *Infura*. This provider is used by Truffle when deploying contracts. In addition, MetaMask providers are also used when we want to interact with the contract in the browser.
  HDWalletProvider provides a custom URL: 'http://127.0.0.1:7545'. This will spawn a development blockchain locally on port 7545[7].

[2]https://moralis.io/truffle-explained-what-is-the-truffle-suite
[3]https://blog.hubspot.com/website/react-js
[4]https://github.com/jakubzapletal/crypto-js
[5]https://www.datastax.com/guides/what-is-web3.js
[6]https://axios-http.com/docs/intro
[7]https://github.com/trufflesuite/truffle-hdwallet-provider

- `Express.js` is a Node.js framework for developing dApps. It provides HTTP methods (GET, POST) to call functions for particular URL routes[8].
  When we run a dApp, we have an HTTP server located on port 3000.

- `FS` provides some functionality to interact with the file system, mostly used functions like: *readFileSync*, *writeFileSync* and *appendFileSync*[9].

Semantic Web Tools

- *EthOn Ontology* is a semantic view of the Ethereum blockchain. It encompasses different classes and relations to cover different concepts of Ethereum such as blocks, transactions, and messages to formalize RDF triples. We used classes and properties related to transaction concepts as a template to model our transaction to DALICC[10].

- *SPARQL query* is defined in Section 2.1.4.

- *Command-Line SPARQL* is a utility of Apache Jena that runs queries on remote SPARQL endpoints or RDF files located on a local computer or web. We used the command as follows[11]:
  - Using command line : *sparql –data rdfFile –query sparqlFile*

Ethereum Tools

- *Infura* is a web3 backend and infrastructure-as-a-Service (IaaS) provider that offers a range of tools to help developers build dApps to connect to the Ethereum blockchain[12].

[8] https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js
[9] https://blog.risingstack.com/fs-module-in-node-js/
[10] https://axios-http.com/docs/intro
[11] http://richard.cyganiak.de/blog/wp-content/uploads/2013/09/jena-sparql-cli-v1.pdf
[12] https://blog.infura.io/post/getting-started-with-infura-28e41844cc89

- *Metamask* is a wallet used to interact with the Ethereum blockchain. It allows users to connect to the network through a browser extension or mobile app. Metamask wallet stores all accounts, each account having its own private key[13].

- *Faucet (ETH faucet)* is the platform that gives some test tokens to a user to test smart contracts or send transactions before deploying them to the mainnet[14].

- *Testnets* are the test blockchain networks that behave similarly to the main blockchain (mainnet). This allows developers to execute their contracts on the test blockchain freely before executing on the main network[15].
  Infura's new testnet faucet is Sepolia ETH which provides the most reliable and high-volume faucets for developers.

- *Web3.eth* is a package that allows interacting with the Ethereum blockchain. It contains many functions to provide more information about executed smart contracts or transactions on the blockchain[16]. In our case, some functions including: *getBlock*, *getTransaction* and *getTransactionReceipt* are chosen to provide some more details which are mapped to EthOn ontology objects properties. These retrieved properties would be used later in triples.

- *Knowledge graph* indicates classes and relations. The idea is to use a graph-based data model to clarify survived transactions, their classes, and relations in much more detail.

[13] https://originstamp.com/blog/metamask-what-is-it-and-how-does-it-work/
[14] https://changelly.com/blog/best-ethereum-faucets/
[15] https://blog.infura.io/post/introducing-infuras-eth-testnet-faucet-get-0-5-eth-daily-to-test-your-dapps
[16] https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html

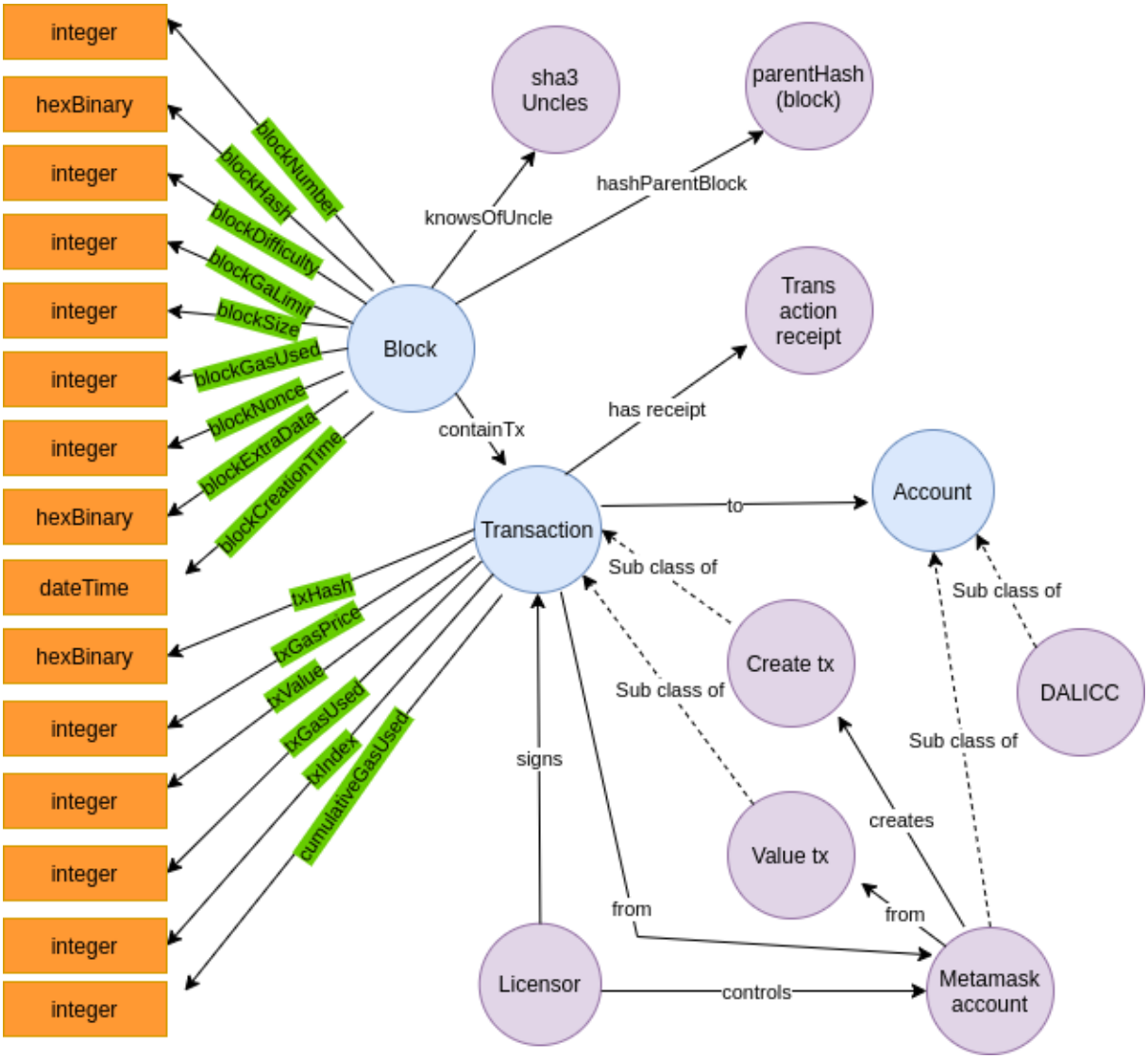


Figure 4.2: Transaction illustration

4.3 Project Architecture

4.3.1 Backend

The backend of this project relies on the Ethereum platform using smart contracts. In addition, DALICC license library the user can license their data or retrieve license information. In the next paragraph, we will focus more on authenticated users as the main challenge in decentralized apps and data licensing systems by users.

Authenticated user[17]

Authentication of users on Ethereum for validation purposes is an essential feature when building dApps. Therefore, programming the functionality of Ethereum authentication for dApps is important for blockchain developers. In this dApp, we consider two solutions for Ethereum authentication as follows:

- *log in to MetaMask:* MetaMask is the most popular cryptocurrency wallet (Ethereum account) to support the Ethereum blockchain. Additionally, MetaMask is a bridge for web3 authentication to an Ethereum-based decentralized application. By logging into MetaMask, users can submit transactions or check the stored data on the blockchain.

- *Using Smart Contract:* The address of the MetaMask is passed to the smart contract as a licensor, then it will use this address as an owner to license their file.

Licensing in Smart Contract follows:

License Information: It represents three elements: licensor address that is passed by MetaMask, the license of the data, and the URI related to this license.
Storage License Information:
Each smart contract that runs on the Ethereum blockchain would be maintained in its permanent storage. The license information is stored in its storage and is changeable only by the smart contract itself.
Retrieve License Information:
As mentioned earlier, only a smart contract can change the data in its memory. Thus, we can consider smart contract as a validating system for license requests. The JavaScript library web3.js is an interface to the Ethereum network for the frontend of this dApp. This allows the frontend to access the smart contract, retrieving or changing license information.

[17] https://medium.com/coinmonks/how-to-do-authentication-in-decentralized-application-dapp-d9bc66b6249c

4.3.2 Frontend

Frontend of this project built with React.js, HTML, and CSS. The React.js is used in this frontend to create the user interface.

User Roadmap

The licensing data from the user interface goes through three phases:

- The user is asked to select a license type from the DALICC Library via Axios and make a request in the frontend. Then, after selecting data in the next field, the user should check if the selected file is already licensed or not. Depending on this verification, the user receives either the confirmation message 'License Detected' or the rejection message 'License Not Detected'.

- The user should go further by pressing the 'License Data/Retrieve License' button to get license information for the confirmation message of the last phase or license their file.
  License information contains some information like license type, license URI retrieved from the DALICC library, and the address of the owner of this license. For licensing data, the user is asked to log in to MetaMask for the authentication process and this address is passed to the smart contract as owner of this license. The licensing process is done by receiving the hash value of the licensed data.

- In the last phase, the user can observe the receipt of this transaction in a table that contains transaction details from the interaction between *web3.eth* and *ethOn* ontology.

Interaction with Backend

Since the backend code (smart contract) of the dApp is on a decentralized network, we focused on the interaction between the smart contract and the Ethereum network. The communication between the frontend and backend is taken over by JavaScript library *web3.js*. For this purpose, web3 provides this connection either with HDWallet-provider to connect to the test network (Ropsten in our case) or Ethereum provider.

Interaction with DALICC

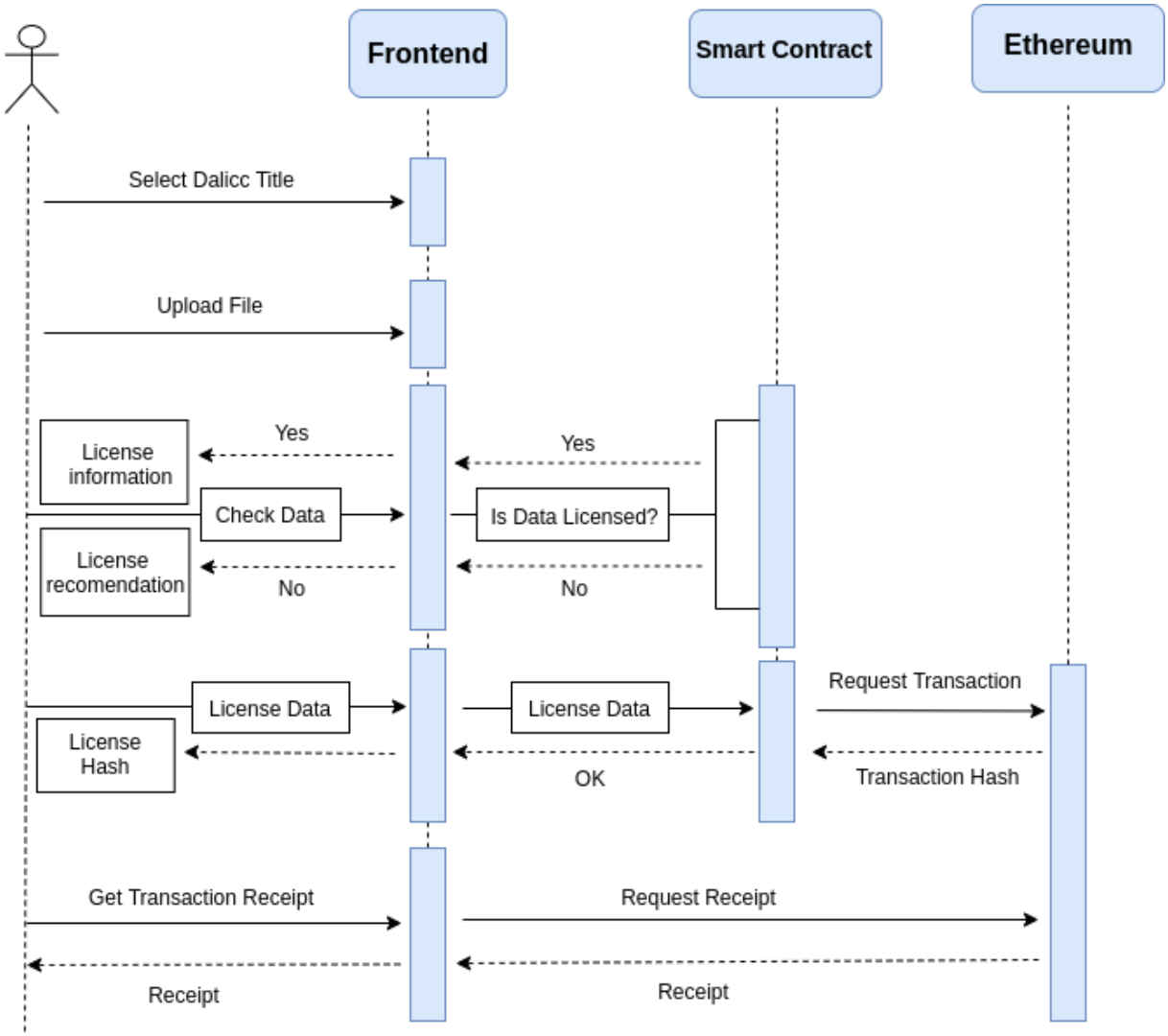


Figure 4.3: Sequence diagram for licensing file

To retrieve the license, an HTTP GET request is sent to the DALICC library endpoint and returns the license which encompasses two elements: license title and license URI. The user can choose just the DALICC title as a license, then the URI of the associated title would be stored subsequently in the smart contract for further processing.

License receipts

After committing a transaction in the blockchain, the user can get transaction details via *web3.eth*, then send transaction details in RDF format to the server to store in a file.

The produced Turtle file will be resulted into a readable format and would be returned to the frontend by an HTTP GET request from the frontend.

4.3.3 DApp Architecture

This application encompasses some components:

- User interface
- Smart contract to communicate with DALICC library
- Ethereum network to support transactions
- Transaction receipt
- EthOn ontology
- SPARQL query

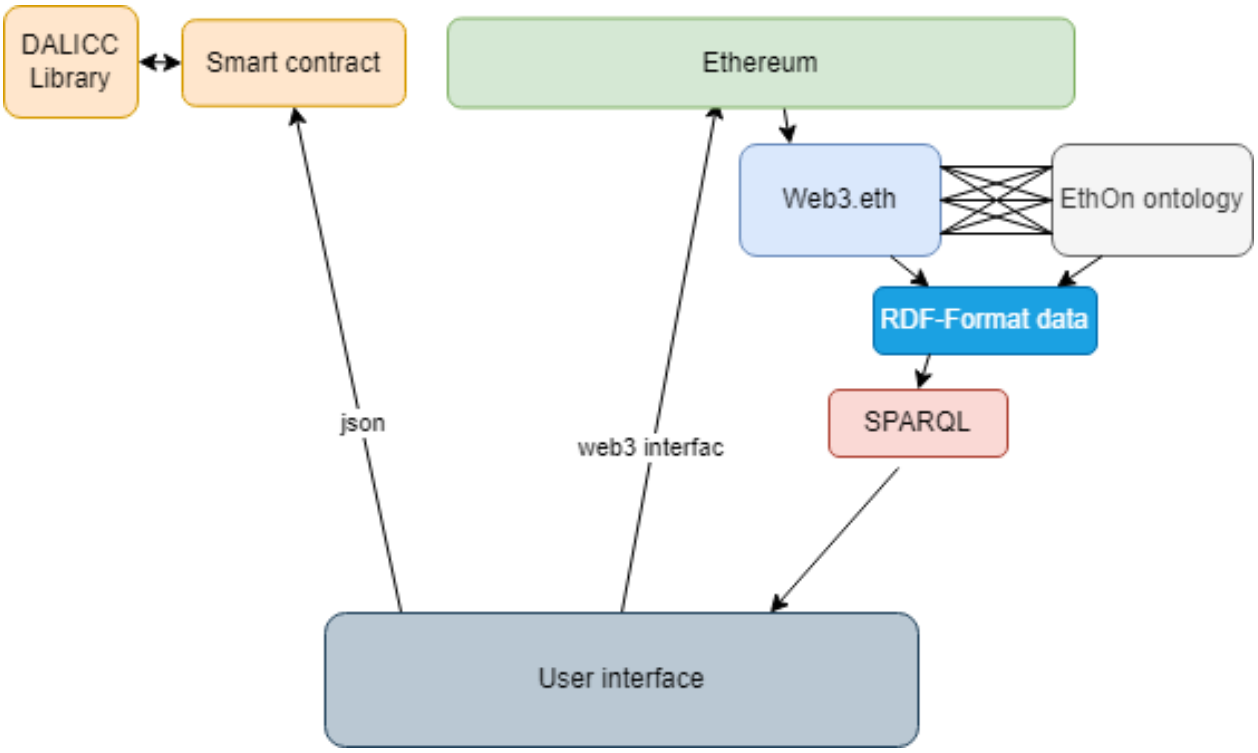


Figure 4.4: DApp architecture

4.4 Implementation

This section comprises the implementation details in a smart contract and some main functions in the application.

4.4.1 Smart Contract Logic

As mentioned before, the backend code of this application is on the Ethereum network. In this section, we will contemplate more on each functionality of the contract in this dApp.

- *Owned Contract*
  This smart contract contains the function to prevent non-owner users from calling the function. This contract contains a function that helps us to restrict access to some functions in another contract.

- *PrimaryLicenseContract*
  This is the main contract that communicates with two other contracts to represent the public interface of the licensing system. It provides functions to license data or retrieve license information.

- *LicenseManager Contract*
  This smart contract is responsible for saving the address of the license or creating one for the new file.

- *License Contract*
  The hash value and licensor address will be stored in this contract. License contract represents only one license and associated license contract for this license. The license contract should have been created by the license manager.

How Does Smart Contract work?

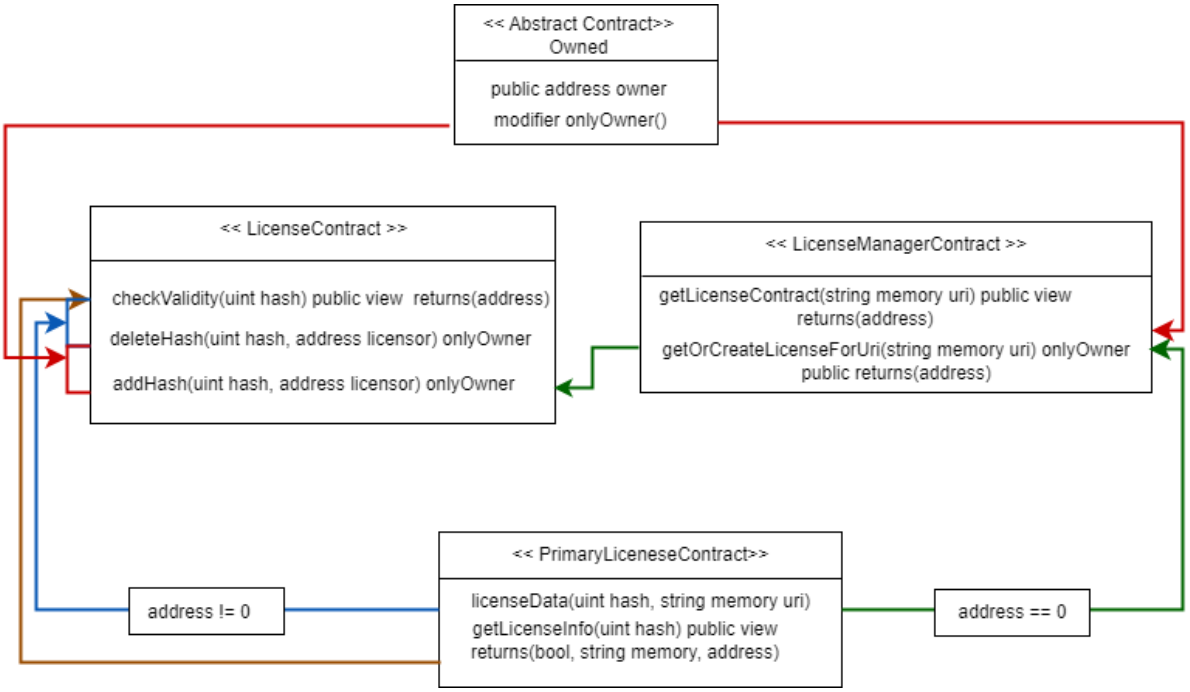


Figure 4.5: Smart contract visualization

- **Licensing System Process**

  To license data, two parameters are needed. The first one is the hash of related data and the second one is the address related to the URI of the license. This smart contract as a *caller* takes over the process:

  License Data

  To license data, the function *licenseData* is used which declares two parameters: the first one is the hash associated with selected data, and the second one is the URI of the selected license. By pressing the 'License Data/Retrieve License' on the frontend, the function *licenseData* would be triggered and perform the steps as follows:

  This function checks if the target address is the zero account, which means the transaction creates a new contract: if not, the function *deleteHash* is called, otherwise *getOrCreateLicenseForUri* would be called. How does licensing data work?

  - *licenseData* function checks if the specified hash is linked to a license and the caller is licensor, then *deleteHash* is called and the hash of related data and

address of licensor would be passed.

Definition *deleteHash*: This function is accessible only for the owner (function caller), which means the caller of this function must be the owner and the passed address should be the licensor. Then the link between the hash and license will be dropped.

- *getOrCreateLicenseForUri* of the license manager is called and the URI of the selected data as input parameter would be passed and returns the address to license contract which represents this URI.

  Definition *getOrCreateLicenseForUri*: This function checks if the caller of this function is the owner, then if there exists a license contract for a given URI. If so, the address of this license contract would be returned. Otherwise, a new license contract will be created and the address of the contract will be stored and returned. The caller's address of the caller will also be used later as the owner of this contract.

- *addHash* function of the License contract is called with two parameters: the first one is the hash of data, and the second one is the function caller.

  Definition *addHash*: This function is accessible only for the owner (function caller), the link between the hash value and the license is created. The second parameter would be stored also as licensor.

- At the end, the event should be emitted to fire the new changes in *PrimaryLicenseContract*.

- **Change License Data**

  This is doable just by the licensor in the same way as license data.

- **Retrieve License Information**

  To retrieve license information, the function *getLicenseInfo* is called, having a hash of data as a parameter, and checks if there is some license information related to this hash. Then, the address is returned; otherwise, a null address and string are returned.

4.4.2 Forntend Workflow

- Define Type: In this step, the user is asked to select a license type from among many different licenses. These licenses are loaded from the DALICC License Library via an Axios GET request, and the user must choose one of them to continue the license processing.

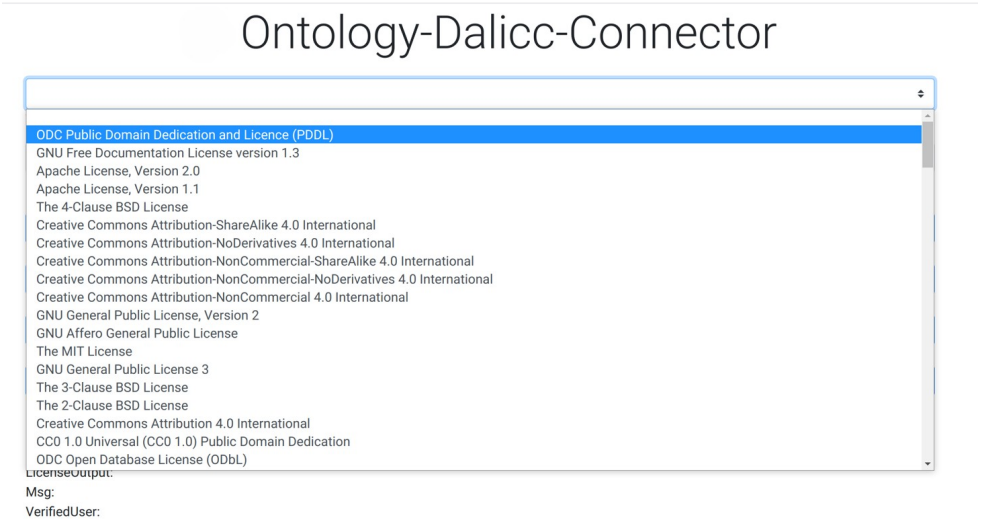


Figure 4.6: Define type from DALICC library

- define content: In this step, the user should choose the file or data that they want to license. Subsequently, the hash SHA3-256 of the selected file is calculated, which is used to retrieve license information later.

- Check Your Data: In this step, the user should check the selected data: if it has been licensed before or not. They will receive just a message either confirming 'License is detected' or rejecting 'License is not detected'. They should go further to obtain more information.

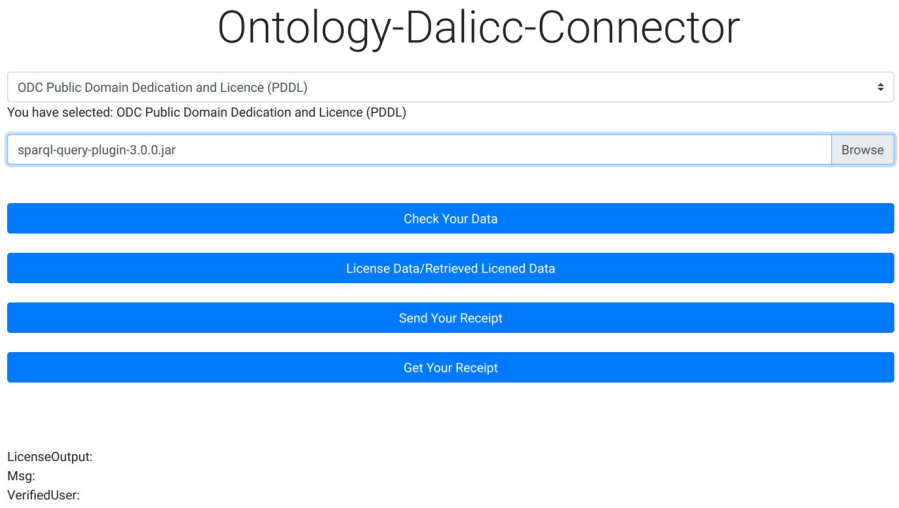


Figure 4.7: Define file from local device

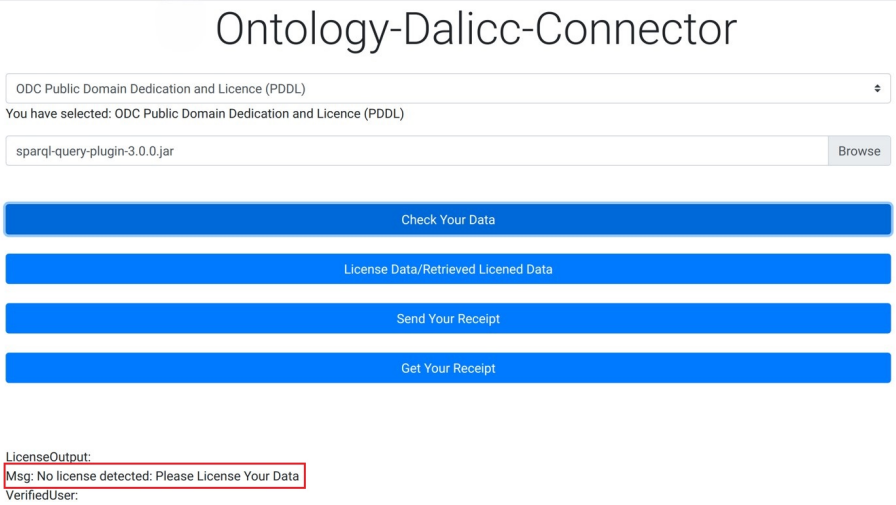


Figure 4.8: Check if file is already licensed?

- License Data/Retrieve License Information: Here, the user receives the result of that last step, by pressing the hash value of selected data `SHA3-256` is calculated and passed to the smart contract using *web.js* interface:

  - License information of the selected file or data from which the user received

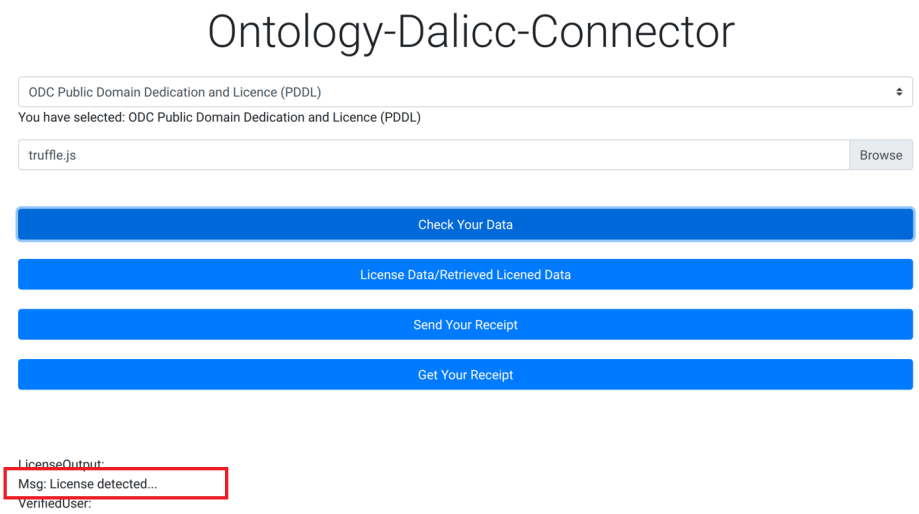


Figure 4.9: Message for licensed data

the 'License Detected' message from the last step. The user receives some information related to the license, licensor address, and URI related to the license.

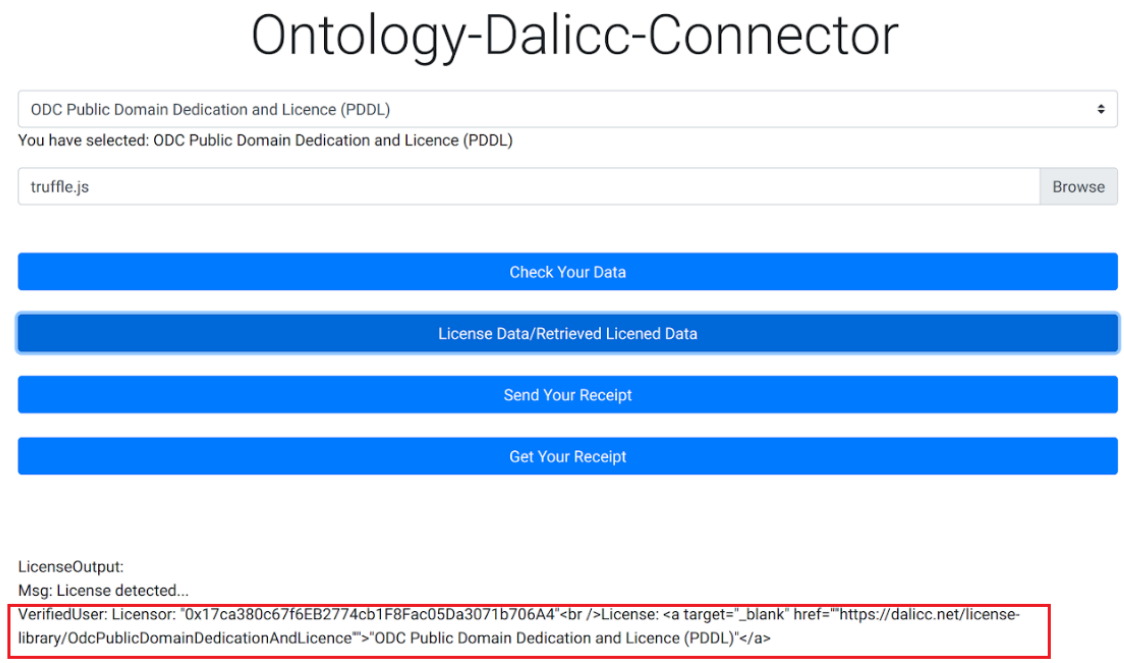


Figure 4.10: License information for licensed data

- Start licensing his/her file, if they received 'No license detected' from the last step. To license data, the user is asked to log into MetaMask to pass this account as the address of the licensor to the smart contract. The SHA3-256 hash value is calculated and passed by the contract to the Web3 JavaScript interface. Users can see this hash in the frontend, depending on the successful transaction or they receive the error message for failed transaction on the front end.

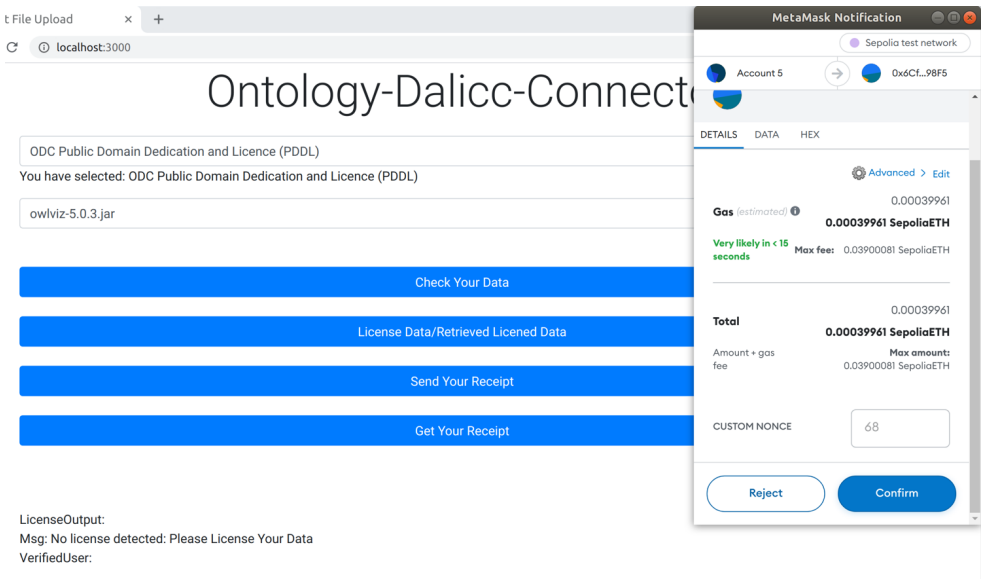


Figure 4.11: Licensing data process

- Send Transaction: After receiving the hash of the transaction in the frontend, the user goes further to get a receipt of this licensing having all details about the transaction. To have this receipt, the user sends a transaction receipt that is not readable to the server for more processing on this raw result, converting to RDF format data and making a query to produce more readable RDF-based data.

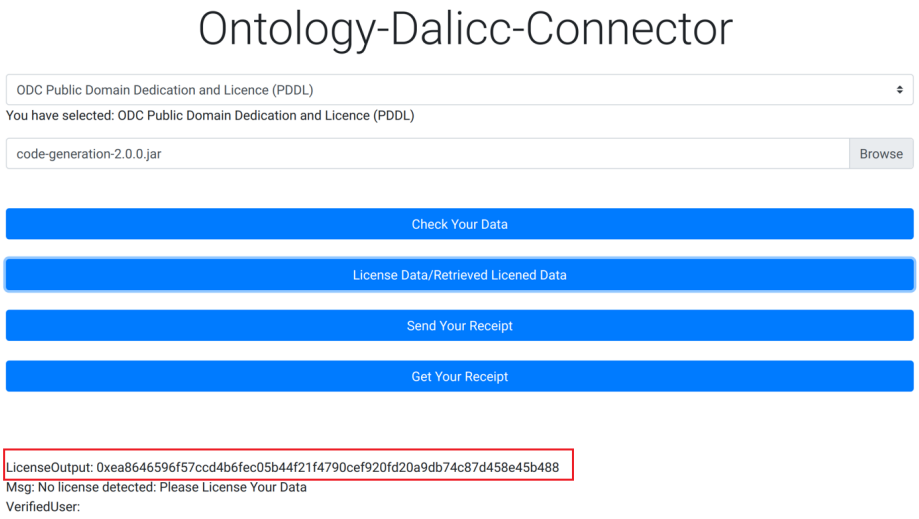


Figure 4.12: License hash after confirming transaction

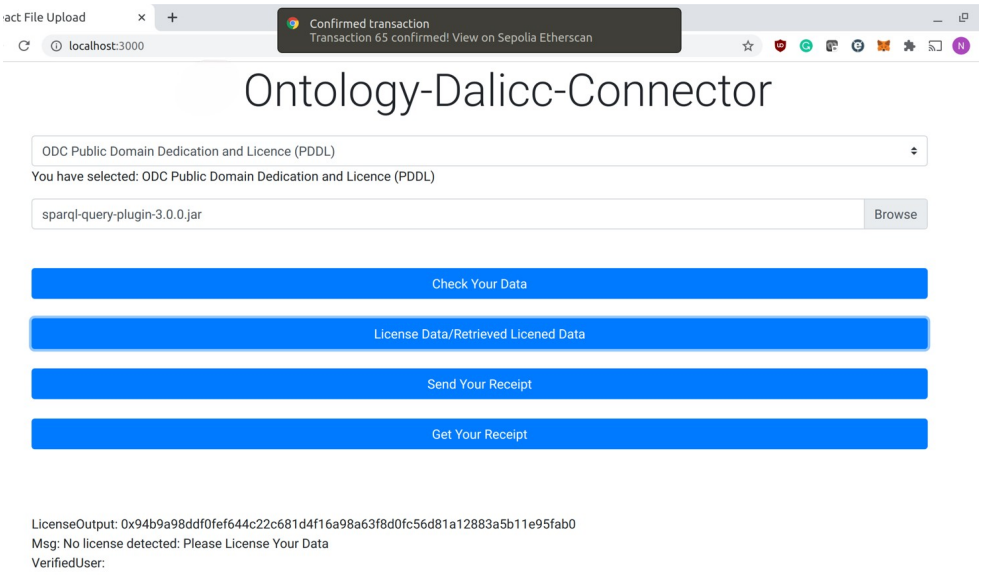
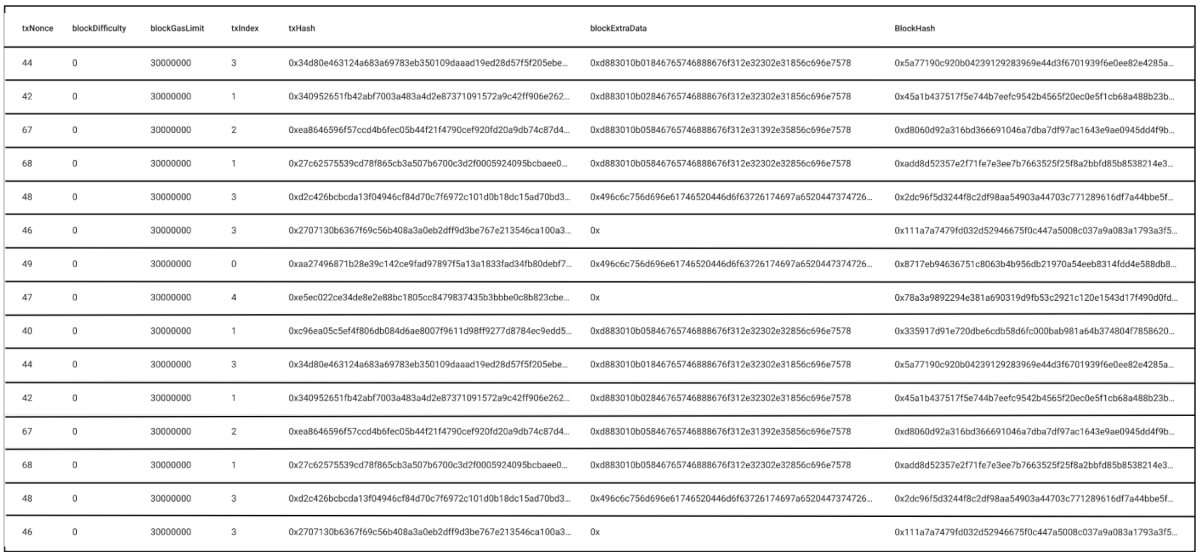


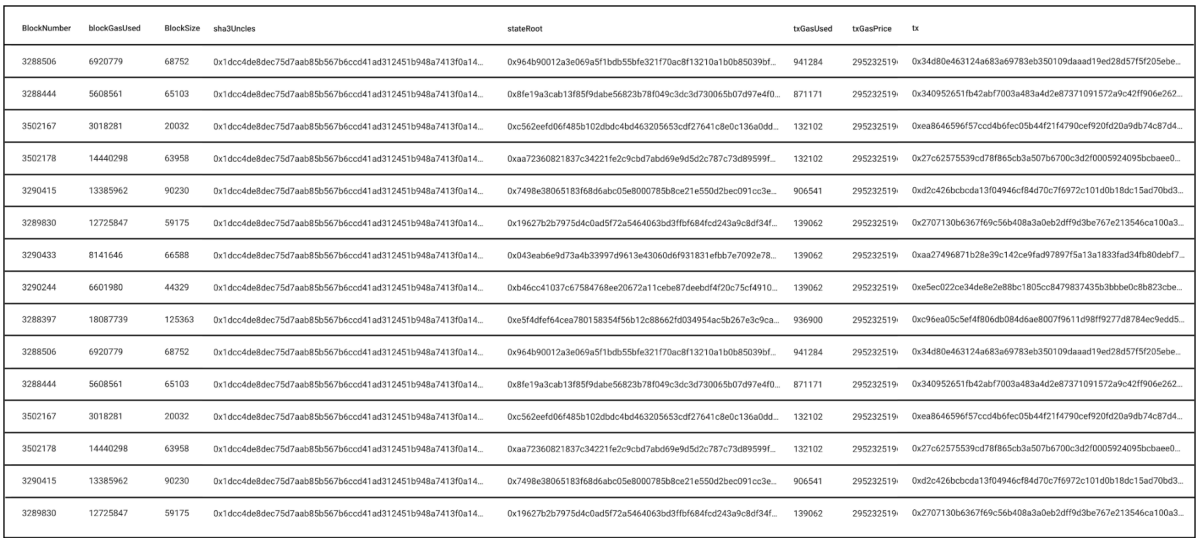
Figure 4.13: Transaction confirmation

- Get Transaction Receipt: In the last step, the user can get a receipt of all transactions that have been done until now as a table.



txNonce	blockDifficulty	blockGasLimit	txIndex	txHash	blockExtraData	BlockHash
44	0	30000000	3	0x34d80e463124a683a69783eb350109daaad19ed28d57f5f205ebe…	0xd883010b01846765746888676f312e32302e31856c696e7578	0x5a77190c920b04239129283969e44d3f6701939f6e0ee82e4285a…
42	0	30000000	1	0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e262…	0xd883010b02846765746888676f312e32302e31856c696e7578	0x45a1b437517f5e744b7eefc9542b4565f20ec0e5f1cb68a488b23b…
67	0	30000000	2	0xea8646596f57ccd4b6fec05b44f21f4790cef920fd20a9db74c87d4…	0xd883010b08467657746888676f312e31392e35856c696e7578	0xd8060d92a316bd366691046a7dba7df97ac1643e9ae0945dd4f9b…
68	0	30000000	1	0x27c62575539cd78f865cb3a507b6700c3d2f000592409 5bcbaee0…	0xd883010b08467657746888676f312e32302e32856c696e7578	0xadd8d52357e2f71fe7e3ee7b7663525f25f8a2bbfd85b8538214e3…
48	0	30000000	3	0xd2c426bcbcda13f04946cf84d70c7f6972c101d0b18dc15ad70bd3…	0x496c6c756d696e61746520446d6f63726174697a6520447374726…	0x2dc96f5d3244f8c2df98aa54903a44703c771289616df7a44bbe5f…
46	0	30000000	3	0x2707130b6367f69c56b408a3a0eb2dff9d3be767e213546ca100a3…	0x	0x111a7a7479fd032d52946675f0c447a5008c037a9a083a1793a3f5…
49	0	30000000	0	0xaa27496871b28e39c142ce9fad97897f5a13a1833fad34fb80debf7…	0x496c6c756d696e61746520446d6f63726174697a6520447374726…	0x8717eb94636751c8063b4b956db21970a54eeb8314fdd4e588db8…
47	0	30000000	4	0xe5ec022ce34de8e2e88bc1805cc8479837435b3bbbe0c8b823cbe…	0x	0x78a3a9892294e381a690319d9fb53c2921c120e1543d17f490d0fd…
40	0	30000000	1	0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd5…	0xd883010b08467657746888676f312e32302e32856c696e7578	0x335917d91e720dbe6cdb58d6fc000bab981a64b374804f7858620…
44	0	30000000	3	0x34d80e463124a683a69783eb350109daaad19ed28d57f5f205ebe…	0xd883010b01846765746888676f312e32302e31856c696e7578	0x5a77190c920b04239129283969e44d3f6701939f6e0ee82e4285a…
42	0	30000000	1	0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e262…	0xd883010b02846765746888676f312e32302e31856c696e7578	0x45a1b437517f5e744b7eefc9542b4565f20ec0e5f1cb68a488b23b…
67	0	30000000	2	0xea8646596f57ccd4b6fec05b44f21f4790cef920fd20a9db74c87d4…	0xd883010b08467657746888676f312e31392e35856c696e7578	0xd8060d92a316bd366691046a7dba7df97ac1643e9ae0945dd4f9b…
68	0	30000000	1	0x27c62575539cd78f865cb3a507b6700c3d2f000592409 5bcbaee0…	0xd883010b08467657746888676f312e32302e32856c696e7578	0xadd8d52357e2f71fe7e3ee7b7663525f25f8a2bbfd85b8538214e3…
48	0	30000000	3	0xd2c426bcbcda13f04946cf84d70c7f6972c101d0b18dc15ad70bd3…	0x496c6c756d696e61746520446d6f63726174697a6520447374726…	0x2dc96f5d3244f8c2df98aa54903a44703c771289616df7a44bbe5f…
46	0	30000000	3	0x2707130b6367f69c56b408a3a0eb2dff9d3be767e213546ca100a3…	0x	0x111a7a7479fd032d52946675f0c447a5008c037a9a083a1793a3f5…

Figure 4.14: Result of semantic mapping



BlockNumber	blockGasUsed	BlockSize	sha3Uncles	stateRoot	txGasUsed	txGasPrice	tx
3288506	6920779	68752	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0x964b90012a3e069a5f1bdb55bfe321f70ac8f132 10a1b0b85039bf…	941284	295232519…	0x34d80e463124a683a69783eb350109daaad19ed28d57f5f205ebe…
3288444	5608561	65103	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0x8fe19a3cab13f85f9dabe56823b78f049c3dc3d730065b07d97e4f0…	871171	295232519…	0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e262…
3502167	3018281	20032	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0xc562eefd06f485b102dbdc4bd463205653cdf27641c8e0c136a0dd…	132102	295232519…	0xea8646596f57ccd4b6fec05b44f21f4790cef920fd20a9db74c87d4…
3502178	14440298	63958	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0xaa72360821837c34221fe2c9cbd7abd69e9d5d2c787c73d89599f…	132102	295232519…	0x27c62575539cd78f865cb3a507b6700c3d2f000592409 5bcbaee0…
3290415	13385962	90230	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0x7498e38065183f68d6abc05e8000785b8ce21e550d2bec091cc3e…	906541	295232519…	0xd2c426bcbcda13f04946cf84d70c7f6972c101d0b18dc15ad70bd3…
3289830	12725847	59175	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0x19627b2b7975d4c0ad5f72a5464063bd3ffbf684fcd243a9c8df34f…	139062	295232519…	0x2707130b6367f69c56b408a3a0eb2dff9d3be767e213546ca100a3…
3290433	8141646	66588	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0x043eab6e9d73a4b33997d9613e43060d6f931831efbb7e7092e78…	139062	295232519…	0xaa27496871b28e39c142ce9fad97897f5a13a1833fad34fb80debf7…
3290244	6601980	44329	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0xb46cc41037c67584768ee20672a11cebe87deebdf4f20c75cf4910…	139062	295232519…	0xe5ec022ce34de8e2e88bc1805cc8479837435b3bbbe0c8b823cbe…
3288397	18087739	125363	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0xe5f4dfef64cea780158354f56b12c88662fd034954ac5b267e3c9ca…	936900	295232519…	0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd5…
3288506	6920779	68752	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0x964b90012a3e069a5f1bdb55bfe321f70ac8f132 10a1b0b85039bf…	941284	295232519…	0x34d80e463124a683a69783eb350109daaad19ed28d57f5f205ebe…
3288444	5608561	65103	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0x8fe19a3cab13f85f9dabe56823b78f049c3dc3d730065b07d97e4f0…	871171	295232519…	0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e262…
3502167	3018281	20032	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0xc562eefd06f485b102dbdc4bd463205653cdf27641c8e0c136a0dd…	132102	295232519…	0xea8646596f57ccd4b6fec05b44f21f4790cef920fd20a9db74c87d4…
3502178	14440298	63958	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0xaa72360821837c34221fe2c9cbd7abd69e9d5d2c787c73d89599f…	132102	295232519…	0x27c62575539cd78f865cb3a507b6700c3d2f000592409 5bcbaee0…
3290415	13385962	90230	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0x7498e38065183f68d6abc05e8000785b8ce21e550d2bec091cc3e…	906541	295232519…	0xd2c426bcbcda13f04946cf84d70c7f6972c101d0b18dc15ad70bd3…
3289830	12725847	59175	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0x19627b2b7975d4c0ad5f72a5464063bd3ffbf684fcd243a9c8df34f…	139062	295232519…	0x2707130b6367f69c56b408a3a0eb2dff9d3be767e213546ca100a3…

Figure 4.15: Result of semantic mapping

Chapter 5

Conclusion

This work adds to the literature by merging semantic licensing from the DALICC library with smart contracts and a few unresolved difficulties, especially in light of the scant number of studies that combine blockchain with the semantic web.
As the foundation of our work, we concentrated on two elements for this purpose:
One such example is the Ethereum platform, which was created and put into use to maintain the system's dependability, security, autonomy, and decentralization. The second is the Ethereum ontology, which in OWL formalizes the ideas and terminology of the Ethereum blockchain and describes Ethereum objects as classes. Blocks, accounts, transactions, contract messages, and the connections between the instances of these classes are only a few of the key blockchain ideas covered. As a result, leveraging Ethereum ontology concepts, a licensing system has been developed that preserves the integrity of license information and offers a semantic perspective of such a deployment environment.
The user is directed step-by-step through the licensing process to begin and receive their license receipt.
This DApp has certain restrictions as well, although most of them have solutions. Thus, in this work, we have created the fundamental techniques for the license attachment to data as well as a semantic mapping for the receipt that is the end product of this attachment procedure.

Appendix A

Semantic mapping column in web3

The appendix comprises semantic mapping elements described and the source code for the smart contracts described in forth chapter.

Column	Type
number	bigint
block_hash	hex_string
parentHash	hex_string
block_nonce	hex_string
sha3Uncles	hex_string
logsBloom	hex_string
transactionsRoot	hex_string
stateRoot	hex_string
miner	hex_string
difficulty	bigint
totalDifficulty	bigint
size	bigint
extraData	hex_string
gasLimit	bigint
gasUsed	bigint
timestamp	bigint
tx_hash	hex_string
tx_nonce	bigint
blockHash	hex_string
blockNumber	bigint
transactionIndex	bigint
value	bigint
gas	bigint
gasPrice	bigint
input	hex_string
status	boolean
blockHash	hex_string
blockNumber	bigint
transactionHash	hex_string
transactionIndex	bigint
from	hex_string
to	hex_string
gasUsed	bigint
cumulativeGasUsed	bigint

Appendix B

RDF triple template in semantic mapping

```
bi:{{number}} a ethon:Block .
bi:{{number}} ethon:number "{{number}}"^^xsd:integer .
bi:{{number}} ethon:blockHash "{{block_hash}}"^^xs:hexBinary .
bi:{{number}} ethon:hasParentBlock ibb:{{parentHash}} .
bi:{{parentHash}} ethon:parentHash "{{parentHash}}"^^xs:hexBinary .
bi:{{number}} ethon:blockNonce "{{block_nonce}}"^^xsd:integer .
bi:{{number}} ethon:knowsOfUncle ibu:{{sha3Uncles}} .
bi:{{number}} ethon:blockLogsBloom "{{logsBloom}}"^^xs:hexBinary .
bi:{{number}} ethon:hasTxTrie ibtx:{{transactionRoot}} .
bi:{{number}} ethon:hasPostBlockState ibs:{{stateRoot}}  .
bi:{{stateRoot}} ethon:stateRoot "{{stateRoot}}"^^xs:hexBinary .
bi:{{number}} ethon:blockDifficulty "{{difficulty}}"^^xs:hexBinary .
bi:{{number}} ethon:blockSize "{{size}}"^^xsd:integer .
bi:{{number}} ethon:blockExtraData "{{extraData}}"^^xs:hexBinary .
bi:{{number}} ethon:blockGasLimit "{{gasLimit}}"^^xsd:integer .
bi:{{number}} ethon:blockGasUsed "{{gasUsed}}"^^xsd:integer    .
bi:{{number}} ethon:blockCreationTime "{{timestamp}}"^^xs:dateTime .
txi:{{blockNumber}} ethon:number "{{number}}"^^xsd:integer .
txi:{{blockNumber}} ethon:containsTx tx:{{tx_hash}} .
txi:{{tx_hash}} ethon:cumulativeGasUsed "{{gas}}"^^xsd:integer .
txi:{{tx_hash}} ethon:txHash "{{tx_hash}}"^^xs:hexBinary .
txi:{{tx_hash}} ethon:txNonce "{{tx_nonce}}"^^xsd:integer .
txi:{{tx_hash}} ethon:txIndex "{{transactionIndex}}"^^xsd:integer .
txi:{{tx_hash}} ethon:txGasPrice "{{gasPrice}}"^^xsd:integer .
txi:{{tx_hash}} ethon:txGasUsed "{{gas}}"^^xsd:integer .
txri:{{tx_hash}} ethon:txHash "{{tx_hash}}"^^xs:hexBinary .
txri:{{tx_hash}} ethon:hasReceipt "{{status}}"^^xsd:boolean .
txri:{{tx_hash}} ethon:txGasUsed "{{gasUsed}}"^^xsd:integer .
txri:{{tx_hash}} ethon:cumulativeGasUsed "{{cumulativeGasUsed}}"^^xsd:integer .
txri:{{tx_hash}} ethon:txIndex "{{transactionIndex}}"^^xsd:integer .
```

Appendix C

SPARQL query example

```
prefix ethon: <http://ethon.consensys.net/>
prefix bi: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/>
prefix ibu: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/>
prefix ibs: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/>
prefix ibb: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/>
prefix ibtx: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/>
prefix ibb: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/>
prefix txi: <https://web3js.readthedocs.io/en/v1.7.3/web3-eth.html#gettransaction/>
prefix tx: <https://web3js.readthedocs.io/en/v1.7.3/web3-eth.html#gettransaction/>
prefix txi: <https://web3js.readthedocs.io/en/v1.7.3/web3-eth.html#gettransaction/>
prefix txri: <https://web3js.readthedocs.io/en/v1.7.3/web3-eth.html#gettransactionreceipt/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix xs: <http://www.w3.org/2001/XMLSchema#int.maxInclusive>



SELECT ?subject ?predicate ?object
WHERE {
  ?subject ?predicate ?object
}
  LIMIT 25
```

Appendix D

SPARQL query example

```
prefix ethon: <http://ethon.consensys.net/>
prefix eth: <http://ethon.consensys.net/>
prefix bi: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/>
prefix ibu: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/>
prefix ibs: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/>
prefix ibb: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/>
prefix txi: <https://web3js.readthedocs.io/en/v1.7.3/web3-eth.html#gettransaction/>
prefix tx: <https://web3js.readthedocs.io/en/v1.7.3/web3-eth.html#gettransaction/>
prefix txi: <https://web3js.readthedocs.io/en/v1.7.3/web3-eth.html#gettransaction/>
prefix ibtx: <https://web3js.readthedocs.io/en/v1.7.3/web3-eth.html#gettransaction/>
prefix txri: <https://web3js.readthedocs.io/en/v1.7.3/web3-eth.html#gettransactionreceipt/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>



SELECT * WHERE {
{
  SELECT  ?number ?blockHash ?parentHash ?blockNonce  ?sha3Uncles ?logsBloom ?stateRoot
          ?blockDifficulty ?blockSize ?blockExtraData ?blockGasLimit ?blockGasUsed
          ?blockCreationTime ?txHash

    WHERE
     {
        { ?block ethon:number ?number . }
        { ?block ethon:blockHash ?blockHash . }
        { ?block ethon:hasParentBlock ?parentHash . }
        { ?block ethon:blockNonce ?blockNonce . }
        { ?block ethon:knowsOfUncle ?sha3Uncles . }
        { ?block ethon:blockLogsBloom ?logsBloom . }
        { ?block ethon:hasPostBlockState ?stateRoot . }
        { ?block ethon:blockDifficulty ?blockDifficulty . }
        { ?block ethon:blockSize ?blockSize . }
        { ?block ethon:blockExtraData ?blockExtraData . }
        { ?block ethon:blockGasLimit ?blockGasLimit . }
        { ?block ethon:blockGasUsed ?blockGasUsed . }
        { ?block ethon:blockCreationTime ?blockCreationTime . }


     }
}

{
 SELECT  ?number ?tx ?txGasUsed ?txHash ?txNonce ?txIndex ?txGasPrice
   WHERE    {

        { ?block ethon:number ?number . }
        { ?block ethon:containsTx ?tx . }
        { ?tx ethon:txGasUsed  ?txGasUsed . }
        { ?tx ethon:txHash ?txHash . }
        { ?tx ethon:txNonce  ?txNonce . }
        { ?tx ethon:txIndex ?txIndex . }
        { ?tx ethon:txGasPrice ?txGasPrice . }
       }
}
}
```

Appendix E

Smart contract

```
pragma solidity >=0.5.0 <0.7.0;

contract Owned {
  address owner;

  constructor() public {
    owner = msg.sender;
  }

  modifier onlyOwner {
    require(msg.sender == owner, "Only Owner can do this operation");
    _;
  }

}

 contract License is Owned {

  mapping(uint => address) private licensors;
  string public uri;

  constructor(address _owner, string memory _uri) public {
      owner = _owner;
      uri = _uri;
  }

  function checkValidity(uint hash) public view  returns(address) {
    return licensors[hash];
  }

  function addHash(uint hash, address licensor) onlyOwner public {
      require(licensors[hash] == address(0) || licensors[hash] == licensor,
                  "Only the licensor can update his license");
     licensors[hash] = licensor;
  }

   function deleteHash(uint hash, address licensor) onlyOwner public {
       require(licensors[hash] == licensor,
      "Only the licensor can delete his license");
       licensors[hash] = address(0);
    }

 }
```

Appendix F

Smart contract

```
contract LicenseManager is License {

    mapping(string => address) private licenseUris;
    constructor (address _owner) License (_owner, uri) public {
        owner = _owner;
    }

    function getLicenseContract(string memory uri) public view returns(address) {
        return licenseUris[uri];
    }


   function getOrCreateLicenseForUri(string memory uri)  onlyOwner public returns(address) {

    address addressOfLicense = licenseUris[uri];
    if(addressOfLicense == address(0)) {
            addressOfLicense = address(new License(owner, uri));
            licenseUris[uri] = addressOfLicense;
        }
     return addressOfLicense;
    }
}
```

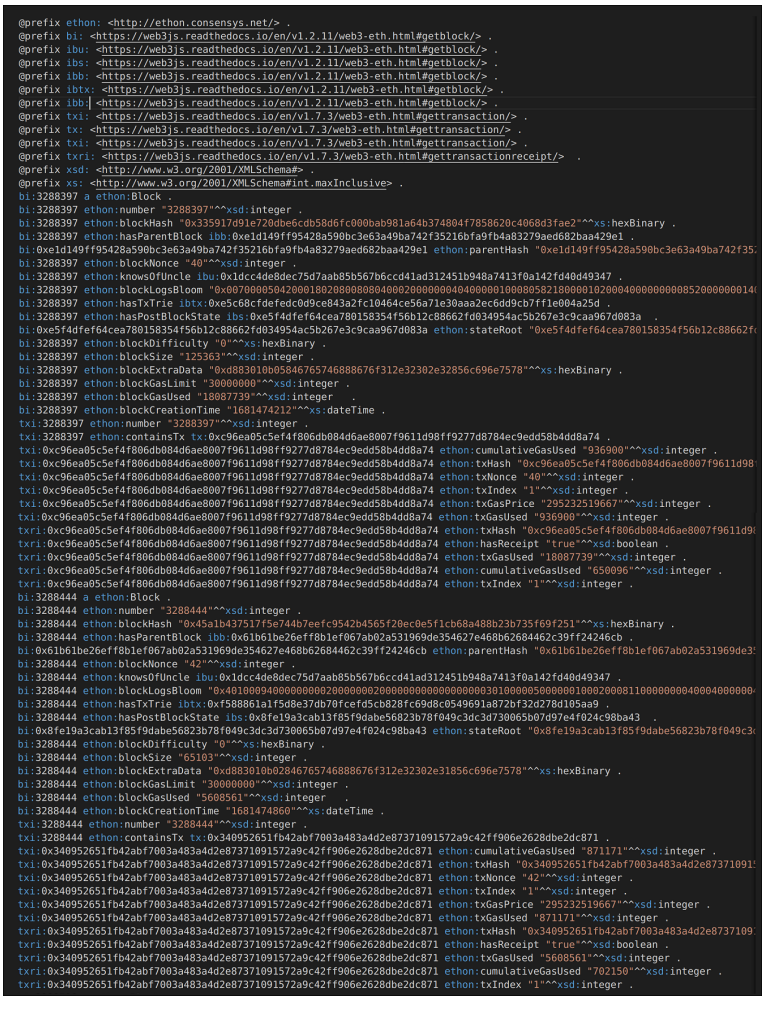
Appendix G

Smart contract

```
contract PrimaryLicenseContract {

  event modifiedLicense(uint indexed hash, string uri);
  mapping(uint => address) private hashes;
  LicenseManager licenseManager = new LicenseManager(address(this));

function licenseData(uint hash, string memory uri) public {

    address addressOfLicense = hashes[hash];
    if(addressOfLicense != address(0)) {

        License retreivedLicense = License(addressOfLicense);
        require(retreivedLicense.checkValidity(hash) == msg.sender, "Only the licensor can update
        retreivedLicense.deleteHash(hash, msg.sender);

      }

      addressOfLicense = licenseManager.getOrCreateLicenseForUri(uri);
      License retreivedLicense = License(addressOfLicense);
      retreivedLicense.addHash(hash, msg.sender);
      hashes[hash] = addressOfLicense;
      emit modifiedLicense(hash, uri);
 }

 function getLicenseInfo(uint hash) public view returns(bool, string memory, address) {
     address addressOfLicense = hashes[hash];
     if(addressOfLicense == address(0)) {
                    return (false, "", address(0));

       }

       License retreivedLicense = License(addressOfLicense);
       return (true, retreivedLicense.uri(), retreivedLicense.checkValidity(hash));

    }
}
```

Appendix H

Result of semantic mapping in RDF format



```
@prefix ethon: <http://ethon.consensys.net/> .
@prefix bi: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/> .
@prefix ibu: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/> .
@prefix ibs: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/> .
@prefix ibb: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/> .
@prefix ibtx: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/> .
@prefix ibb: <https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getblock/> .
@prefix txi: <https://web3js.readthedocs.io/en/v1.7.3/web3-eth.html#gettransaction/> .
@prefix tx: <https://web3js.readthedocs.io/en/v1.7.3/web3-eth.html#gettransaction/> .
@prefix txi: <https://web3js.readthedocs.io/en/v1.7.3/web3-eth.html#gettransaction/> .
@prefix txri: <https://web3js.readthedocs.io/en/v1.7.3/web3-eth.html#gettransactionreceipt/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix xs: <http://www.w3.org/2001/XMLSchema#int.maxInclusive> .
bi:3288397 a ethon:Block .
bi:3288397 ethon:number "3288397"^^xsd:integer .
bi:3288397 ethon:blockHash "0x335917d91e720dbe6cdb58d6fc000bab981a64b374804f7858620c4068d3fae2"^^xs:hexBinary .
bi:3288397 ethon:hasParentBlock ibb:0xe1d149ff95428a590bc3e63a49ba742f35216bfa9fb4a83279aed682baa429e1 .
bi:0xe1d149ff95428a590bc3e63a49ba742f35216bfa9fb4a83279aed682baa429e1 ethon:parentHash "0xe1d149ff95428a590bc3e63a49ba742f35
bi:3288397 ethon:blockNonce "40"^^xsd:integer .
bi:3288397 ethon:knowsOfUncle ibu:0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347 .
bi:3288397 ethon:blockLogsBloom "0x0070000504200018020800080040002000000040400001000805821800001020004000000000852000000014
bi:3288397 ethon:hasTxTrie ibtx:0xe5c68cfdefedc0d9ce843a2fc10464ce56a71e30aaa2ec6dd9cb7ff1e004a25d .
bi:3288397 ethon:hasPostBlockState ibs:0xe5f4dfef64cea780158354f56b12c88662fd034954ac5b267e3c9caa967d083a .
bi:0xe5f4dfef64cea780158354f56b12c88662fd034954ac5b267e3c9caa967d083a ethon:stateRoot "0xe5f4dfef64cea780158354f56b12c88662f
bi:3288397 ethon:blockDifficulty "0"^^xs:hexBinary .
bi:3288397 ethon:blockSize "125363"^^xsd:integer .
bi:3288397 ethon:blockExtraData "0xd883010b05846765746888676f312e32302e32856c696e7578"^^xs:hexBinary .
bi:3288397 ethon:blockGasLimit "30000000"^^xsd:integer .
bi:3288397 ethon:blockGasUsed "18087739"^^xsd:integer .
bi:3288397 ethon:blockCreationTime "1681474212"^^xs:dateTime .
txi:3288397 ethon:number "3288397"^^xsd:integer .
txi:3288397 ethon:containsTx tx:0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd58b4dd8a74 .
txi:0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd58b4dd8a74 ethon:cumulativeGasUsed "936900"^^xsd:integer .
txi:0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd58b4dd8a74 ethon:txHash "0xc96ea05c5ef4f806db084d6ae8007f9611d98
txi:0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd58b4dd8a74 ethon:txNonce "40"^^xsd:integer .
txi:0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd58b4dd8a74 ethon:txIndex "1"^^xsd:integer .
txi:0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd58b4dd8a74 ethon:txGasPrice "295232519667"^^xsd:integer .
txi:0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd58b4dd8a74 ethon:txGasUsed "936900"^^xsd:integer .
txri:0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd58b4dd8a74 ethon:txHash "0xc96ea05c5ef4f806db084d6ae8007f9611d9
txri:0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd58b4dd8a74 ethon:hasReceipt "true"^^xsd:boolean .
txri:0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd58b4dd8a74 ethon:txGasUsed "18087739"^^xsd:integer .
txri:0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd58b4dd8a74 ethon:cumulativeGasUsed "650096"^^xsd:integer .
txri:0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd58b4dd8a74 ethon:txIndex "1"^^xsd:integer .
bi:3288444 a ethon:Block .
bi:3288444 ethon:number "3288444"^^xsd:integer .
bi:3288444 ethon:blockHash "0x45a1b437517f5e744b7eefc9542b4565f20ec0e5f1cb68a488b23b735f69f251"^^xs:hexBinary .
bi:3288444 ethon:hasParentBlock ibb:0x61b61be26eff8b1ef067ab02a531969de354627e468b62684462c39ff24246cb .
bi:0x61b61be26eff8b1ef067ab02a531969de354627e468b62684462c39ff24246cb ethon:parentHash "0x61b61be26eff8b1ef067ab02a531969de3
bi:3288444 ethon:blockNonce "42"^^xsd:integer .
bi:3288444 ethon:knowsOfUncle ibu:0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347 .
bi:3288444 ethon:blockLogsBloom "0x4010009400000000020000002000000000000000000030100000500000001000200081100000000400040000000
bi:3288444 ethon:hasTxTrie ibtx:0xf588861a1f5d8e37db70fcefd5cb828fc69d8c0549691a872bf32d278d105aa9 .
bi:3288444 ethon:hasPostBlockState ibs:0x8fe19a3cab13f85f9dabe56823b78f049c3dc3d730065b07d97e4f024c98ba43 .
bi:0x8fe19a3cab13f85f9dabe56823b78f049c3dc3d730065b07d97e4f024c98ba43 ethon:stateRoot "0x8fe19a3cab13f85f9dabe56823b78f049c3
bi:3288444 ethon:blockDifficulty "0"^^xs:hexBinary .
bi:3288444 ethon:blockSize "65103"^^xsd:integer .
bi:3288444 ethon:blockExtraData "0xd883010b02846765746888676f312e32302e31856c696e7578"^^xs:hexBinary .
bi:3288444 ethon:blockGasLimit "30000000"^^xsd:integer .
bi:3288444 ethon:blockGasUsed "5608561"^^xsd:integer .
bi:3288444 ethon:blockCreationTime "1681474860"^^xs:dateTime .
txi:3288444 ethon:number "3288444"^^xsd:integer .
txi:3288444 ethon:containsTx tx:0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e2628dbe2dc871 .
txi:0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e2628dbe2dc871 ethon:cumulativeGasUsed "871171"^^xsd:integer .
txi:0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e2628dbe2dc871 ethon:txHash "0x340952651fb42abf7003a483a4d2e873710915
txi:0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e2628dbe2dc871 ethon:txNonce "42"^^xsd:integer .
txi:0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e2628dbe2dc871 ethon:txIndex "1"^^xsd:integer .
txi:0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e2628dbe2dc871 ethon:txGasPrice "295232519667"^^xsd:integer .
txi:0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e2628dbe2dc871 ethon:txGasUsed "871171"^^xsd:integer .
txri:0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e2628dbe2dc871 ethon:txHash "0x340952651fb42abf7003a483a4d2e8737109
txri:0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e2628dbe2dc871 ethon:hasReceipt "true"^^xsd:boolean .
txri:0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e2628dbe2dc871 ethon:txGasUsed "5608561"^^xsd:integer .
txri:0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e2628dbe2dc871 ethon:cumulativeGasUsed "702150"^^xsd:integer .
txri:0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e2628dbe2dc871 ethon:txIndex "1"^^xsd:integer .
```

Bibliography

[1] Sareh Aghaei, Mohammad Ali Nematbakhsh, and Hadi Khosravi Farsani. Evolution of the world wide web: From web 1.0 to web 4.0. *International Journal of Web & Semantic Technology*, 3(1):1–10, 2012.

[2] Paul Baran. On distributed communications networks. *IEEE transactions on Communications Systems*, 12(1):1–9, 1964.

[3] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227. IGI global, 2011.

[4] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *Spacial section on the plethora of research in internet of thing (IoT)*, 12: 2292 – 2303, 3 2016. doi: 10.1109/ACCESS.2016.2566339.

[5] Morris J. Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. 37, 2015. doi: http://dx.doi.org/10.6028/NIST.FIPS.202. URL https://www.nist.gov/publications/sha-3-standard-permutation-based-hash-and-extendable-output-functions.

[6] Matthew English, Sören Auer, and John Domingue. Blockchain technologies & the semantic web: A framework for symbiotic development. In *Computer Science Conference for University of Bonn Students*, volume 47, page 61, 2016. URL http://cscubs.cs.uni-bonn.de/2016/proceedings/paper-10.pdf.

[7] Ilya Grishchenko, Matteo Maffei, and Clara Schneidewind. A semantic framework for the security analysis of ethereum smart contracts. In *Principles of Security and Trust: 7th International Conference, POST 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings 7*, pages 243–269. Springer, 2018.

[8] Manav Gupta. *Blockchain for dummies*, volume 51. John Wiley, 2018. URL https://www.ibm.com/downloads/cas/D8O9VBAK.

[9] Péter Hegedűs. Towards analyzing the complexity landscape of solidity based ethereum smart contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, pages 35–39. ACM Digital Library home, 2018.

[10] David Gatta, Kilian Hinteregger, and Anna Fensel. Making licensing of content and data explicit with semantics and blockchain. In *Information Management and Big Data - 8th Annual International Conference, SIMBig 2021, Proceedings*, pages 370–379. Cham

: Springer (Communications in Computer and Information Science), 2022. doi: https://doi.org/10.1007/978-3-031-04447-2_25.

[11] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. Zeus: analyzing safety of smart contracts. In *Ndss*, pages 1–12, 2018.

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

[13] Max Luke, Anna Dimitrova, Stephen James Lee, and Zdenek Pekarek. Blockchain in electricity: a critical review of progress to date. *NERA Econ. Consult. eurelectric*, 36, 2018. URL https://www.researchgate.net/publication/332138382.

[14] Faraz Masood and Arman Rasool Faridi. An overview of distributed ledger technology and its applications. *International Journal of Computer Sciences and Engineering*, 6(10): 422–427, 2018.

[15] William Metcalfe et al. Ethereum, smart contracts, dapps. *Blockchain and Crypt Currency*, 77, 2020.

[16] Harish Natarajan, Solvej Krause, and Helen Gradstein. *Distributed Ledger Technology(DLT) and Blockchain*, volume 60. World Bank Group, 2017.

[17] M Tamer Özsu and Patrick Valduriez. Distributed and parallel database systems. *ACM Computing Surveys (CSUR)*, 28(1):125–128, 1996.

[18] P Paul, PS Aithal, and Ricardo Saavedra. Blockchain technology and its types—a short review. *International Journal of Applied Science and Engineering (IJASE)*, 9(2):189–200, 2021.

[19] Tassilo Pellegrini, Victor Mireles, Simon Steyskal, Oleksandra Panasiuk, Anna Fensel, and Sabrina Kirrane. Automated rights clearance using semantic web technologies: The dalicc framework. *Semantic Applications: Methodology, Technology, Corporate Use*, pages 203–218, 2018.

[20] Tassilo Pellegrini, Giray Havur, Simon Steyskal, Oleksandra Panasiuk, Anna Fensel, Victor Mireles, Thomas Thurner, Axel Polleres, Sabrina Kirrane, and Andrea Schönhofer. Dalicc: a license management framework for digital assets. *Proceedings of the Internationales Rechtsinformatik Symposion (IRIS)*, 10, 2019.

[21] Pablo Lamela Seijas, Simon Thompson, and Darryl McAdams. Scripting smart contracts for distributed ledger technology. *Cryptology ePrint Archive*, 2016. URL https://eprint.iacr.org/2016/1156.

[22] Ali Sunyaev. Distributed ledger technology. *Internet computing: Principles of distributed systems and emerging internet-based technologies*, pages 265–299, 2020.

[23] Nick Szabo. Smart contracts: Building blocks for digital markets. *URL: http://www.fon. hum. uva. nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo. best. vwh. net/smart_ contracts_2. html (: 30 2018 .)*, 11, 2017. URL http://www.truevaluemetrics.org/DBpdfs/BlockChain/Nick-Szabo-Smart-Contracts-Building-Blocks-for-Digital-Markets-1996-14591.pdf.

[24] Allan Third and John Domingue. Linked data indexing of distributed ledgers. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 1431–1436. ACM Digital Library home, 5 2017. doi: https://doi.org/10.1145/3041021.3053895.

[25] Umar Rashid, Allan Third, and John Domingue. Web service for semantic negotiation of smart contracts. 6, 6 2018. URL https://openreview.net/forum?id=BJg2gCu1e7.

[26] Mirek Sopek, Przemyslaw Gradzki, Witold Kosowski, Dominik Kozuski, Rafa Troczak, and Robert Trypuz. Deriving validity time in knowledge graph. In *Companion proceedings of the the web conference 2018*, volume 8, pages 1771–1776, 5 2018. doi: https://doi.org/10.1145/3184558.3191639.

[27] Hector Ugarte. A more pragmatic web 3.0: Linked blockchain data. *Bonn, Germany*, 15, 4 2017. doi: DOI:10.13140/RG.2.2.10304.12807/1. URL https://semanticblocks.files.wordpress.com/2017/03/linked_blockchain_paper_final.pdf.

[28] Wesley Egbertsen, Gerdinand Hardeman, Maarten van den Hoven, Gert van der Kolk, and Arthur van Rijsewijk. Replacing paper contracts with ethereum smart contracts contract innovation with ethereum. *Semantic Scholar*, 35:1–35, 2016. URL https://allquantor.at/blockchainbib/pdf/egbertsen2016replacing.pdf.

[29] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.