Semantic Smart Contracts And Their Integration With Semantic Data Licensing

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

A thesis submitted for the degree of

Master's Program
Computer Science

2023

Abstract

This thesis deals with the problem of integrating semantic licensing from the Data Licenses Clearance Center (DALICC) library into a smart contract and semantic representation of the deployment environment of such a smart contract. For this purpose, we divide this research into four sections: first, we focused on how smart contracts build up on the blockchain and some more details about this system. Second, we demonstrate how blockchain can be integrated with semantic web technology. It is focused on some solutions for indexing and executing smart contracts on the Ethereum blockchain. And the third section is focused on DALICC as a framework that helps to detect license conflict and reduce the costs of rights clearance. And in the last section, we built up DApp to integrate semantic licenses with contents using smart contracts and represent a semantic model of these deployments.

Contents

List of Figures

0.1 Introduction

Distributed ledger on the blockchain has become the popular in recent years [4]. The features which make blockchain useful in cryptocurrency such as proof of consensus, secure transactions, and transparently - also make it suitable in many other contexts like healthcare, social, supply chain management, etc.

Ethereum blockchain is one of the well-known ledgers which is capable of generating decentralized applications and managing cryptocurrency using smart contracts [4]. It is the platform for developers coding smart contracts in a decentralized paradigm to generate decentralized application DApp [16]. A smart contract is a code written in a solidity program language, stored in blockchain, and executing some functionality out of blockchain based on the terms of the contract in the most trusted way [4].

As distributed ledger like blockchian has been widely used in diverse type (such as public, private and etc), the need of making queries over data, and index entries, becomes more important. The most important thing is integrating stored data on the blockchain with external resources, in other words, there is a need for linked data [4].

According to Tim-Berners-Lee (creator of the World Wide Web), 'linked data is the semantic web done right, and the web done right'[26]. The semantic web is a set of standards that enable access, integration, and query on data sources, and it creates a variety of insight between different application domains. It can also use ontology to represent Ethereum entities like blocks, and transactions to allow queries from the web using SPARQL and linking datasets [4].

In this thesis, we deal with some problems: First one is to find a method to integrate semantic licenses from the DALICC library into a smart contract. DALICC stands for Data Licenses Clearance Center is a framework that supports automated clearance of rights to help detect license conflict and reduce the cost of high transactions related to manual clearance of licensing contents [18].

The second one is that licenses from DALICC are already connected to the blockchain, but it does not have a semantic representation of the records themselves. Therefore, we used EthOn ontology[1] to generate a semantic model, mapping this model with information retrieved from deployed smart contracts in RDF format.

To address these issues in our thesis, we created a DApp adapted from previous work on

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

a similar topic to attach licenses to contents[2][2], then applied semantic web techniques linking extracted data from Ethereum transactions to related EthOn ontology concepts. In the first chapter, we described distributed ledger technology, blockchain, Ethereum, and some relevant details to have a better insight into the used technology. In the next chapter, we represented some works around how the semantic web can be applied to the blockchain. And in the last chapter, we presented DApp to show smart contract integration with the DALICC semantic licenses library and semantic representation of this deployment.

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

Chapter 1

Smart Contract and Distributed Ledger Technology

With the advances in technology, transferring money is done over networks, and we just need to update related database entries that are controlled by central authorities like banks. All these processes require a third party to conduct appropriate transactions between unknown parties.

There are many problems with having third parties, such as controlling the whole transaction by a single authority and invalidating any transaction that serves their purpose. All these issues can be resolved using the novel idea of a distributed ledger. In this new technology, third parties are eliminated, and all transactions are conducted over the public network between parties within minutes.
With the presence of new technology, no one can manipulate transactions stored in a ledger or trace the involved parties in these transactions [15].

1.1 Ledger

A ledger is a book or computer that records transactions associated with a financial system. There are two distinct ledgers, as follows:
Centralized Ledger Contains all recordings of transactions related to company assets, costs, 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 [14].

1.2 Distributed Vs. Decentralized

The difference between decentralized and distributed is made by Baran (1964) [1].Decentralized means that there is no single authority to make a decision. Each participant can make a decision individually, and the system gathers all responses as resulting behavior. However, there is no single authority in the distributed system too, but the process is spread across all participants and decisions will be centralized. The main difference between distributed and decentralized is that a decentralized database is a collection of interconnected databases that work independently in different locations. Ozsu et al. [30] define a distributed database as a'collection of multiple, logically interrelated databases distributed over a computer network and distributed database makes a transparent distribution to all users' [30]. Based on this definition, blockchain technology covers both types, as it appears as a single system to its users and performs a task across a network. Thus, blockchain is a form of a distributed database system [14].

1.3 Distributed Ledger Technology (DLT)

DLT refers to a database that provides identical copies of shared data among participants, which would be updated using a complex consensus mechanism between participants. It is used to reduce costs and increase transparency, traceability, and the speed of the process.
This technology involves many challenges, and some of them have not been resolved so far. The most common challenges of DLT concern scalability, inseparability, and data privacy [27].

How Does DLT Work?

DLT is the result of combining the main three technologies:
- *P2P*: all participants (nodes) act simultaneously as client and server, consuming and

contributing resources.
- *Cryptography* is used to authenticate the identity of the participant and the information between the two parties. Using encryption helps prevent third parties from accessing information.
- *Consensus algorithm* allows network participants to come into agreement to add a new node (block) to the ledger [27].

1.4 Blockchain

According to what World Bank Group [9] referred, blockchain is the most popular distributed ledger that stores and publishes data in packages called 'blocks'. Each block contains information such as nonce, timestamp, block hash, and a hash pointer to the previous block in its header. Therefore, all these blocks are connected in a digital chain [9].
Luke et al. [20] refer to blockchain as a list of blocks that are linked to each other and secured cryptography. The participants on networks have an identical copy of these records stored locally on the computers of all participants. Blockchain starts processing, when the user request transaction whether is a transaction, contract, or other information. The transaction is broadcast on P2P network of nodes. Following that, the verification process takes place where all of the nodes in the P2P network verify the transactions via the hashes which are generated by some algorithm. Once verification is completed, transaction detail will be stored in a block. Finally, a new block is added to a chain in a way that is permanent and unchangeable [20]. The initial block in the blockchain known as *Genesis* block, the other nodes will be added to the chain after the process of consensus between nodes. The consensus mechanism allows the blockchain to grow without fear of manipulating the information of blocks. Since the blocks contain transactions, the consensus process takes place in a predefined time interval. This interval is the duration of when the initiation of the transaction took place and the addition of the transaction into a blockchain. This confirmation time is varied based on block size, transaction, and consensus algorithm. There are different methods for consensus mechanism listed:

- Proof of Work (PoW): It is a mechanism that ensures consensus is done without any central control. By the usage of this mechanism miners compete to complete their transaction first into blockchain and get rewards (e.g: Bitcoin, Ether). Miners (actors who participate in cryptocurrency transactions) connect to blockchain and accomplish tasks validating transactions to add new blocks by solving a cryptographic puzzle and anybody who completes their task sooner can add their block first in blockchain [24].

- Proof of Stake (PoS): It is an alternative to proof-of-work that fewer CPU computations for mining. In proof of stake, the chance of mining the next block depends on node balance. In private networks, however, where the participants know each other, consensus mechanisms such as proof of work are not required. This particularly removes the need for mining and gives us more variety of consensus protocols for picking from [3].

- Proof of Authority (PoA): It confirms accounts and allows them to add a transaction in blocks. Since this approach is much more centralized and transaction speed is faster, is prone to be attacked more than the other methods [20].

- Practical Byzantine Fault Tolerance (PBFT): Blockchain tries to solve the problem called'Byzantine Generals' which refers to some members on the network who send incoherent or fake information related to the transaction to others. Since there is no authority on blockchain to correct them, leads to the unreliability of blockchain. PBFT algorithm tries to achieve a consensus to solve this issue in a way that uses the concept of primary and secondary vote. Secondary vote automatically evaluate the decisions made by primary vote and can collectively change to a new primary, if primary vote is compromised [20].

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

- Distributed ledger: The data is not controlled by any single authority. It is shared and updated across the network and the new changes are replicated to all participants.

- Orchestrated and flexible: Since smart contracts can be executed on the blockchain, the blockchain can be evolved to support business processes and activities.
- Transparent and auditable: There is no need for a third party or other authority, as all participants have access to the same ledger, verify transactions, and identify the owner.
- Secure, private, and indelible: Blockchain provides these features using some capabilities such as permissions and cryptography which ensures that unauthorized users do not have any access to the network. It means that participants are truly who they claim.
- Consensus: All nodes on the network should agree to validate transition and blockchain performs this process by consensus algorithm [8].

Type of Blockchain

According to Aithal et al. [7], blockchain is used to transfer and exchange information through a secure network. Primarily, there were two types of blockchain technology: public and private networks. Based on further analysis, blockchain can also be classified as consortium blockchain technology and hybrid blockchain technology [7].
It should be noted that all kinds of blockchains consist of nodes and operate on a P2P (peer-to-peer) network. Aithal et al. [7] divided blockchain into three types: public blockchain, private blockchain, and consortium blockchain. Additionally, there is another type of blockchain known as the hybrid blockchain.

- **Public Blockchain** is the primary type of blockchain that is decentralized and open by nature. In this system, anyone is allowed to join the network and create consensus. In a public blockchain, any miner (participant) can create consensus mechanisms such as proof of work and proof of stake to validate transactions with a low rate of validity [22].
- **Private Blockchain** is not open and restricted in a way that only authorized participants have the right to validate transactions. Therefore, it provides better privacy, improves scalability, and mitigates security issues. This blockchain does not have mining computations to reach consensus because all participants are known in this network [22].

- **Consortium Blockchain** is semi-decentralized blockchain and is used to perform activities for a single organization, such as a bank, etc. The difference between private blockchain and this type is that Consortium Blockchain is controlled by a group rather than a single authority [7].

- **Hybrid Blockchain** is a combination of public and private blockchains. Thus, it combines the benefits of privacy in private blockchain with the security and transparency of the public blockchain. In this type of blockchain, the user can control who has access to which data on the blockchain. A transaction can be verified in a private network, and the user can release it to the public blockchain. By doing so, only selected parts of the records can be accessible in public, while the rest can remain confidential in a private network [7].

1.5 Ethereum

Ethereum is the most active public blockchain in the world at present. It is another cryptocurrency similar to Bitcoin that is built on top of the blockchain. The participants publish the transactions on the network, which are then stored in blocks and added to the blockchain using a consensus mechanism. The term state in Ethereum refers to the state of different accounts populating on the blockchain. An account in Ethereum can either be an external account related to the user or a contract account that obtains constant storage in the blockchain. The virtual currency in this system is *Ether*. The transaction can change the state of the system by creating a new contract or invoking an existing contract [13].

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:

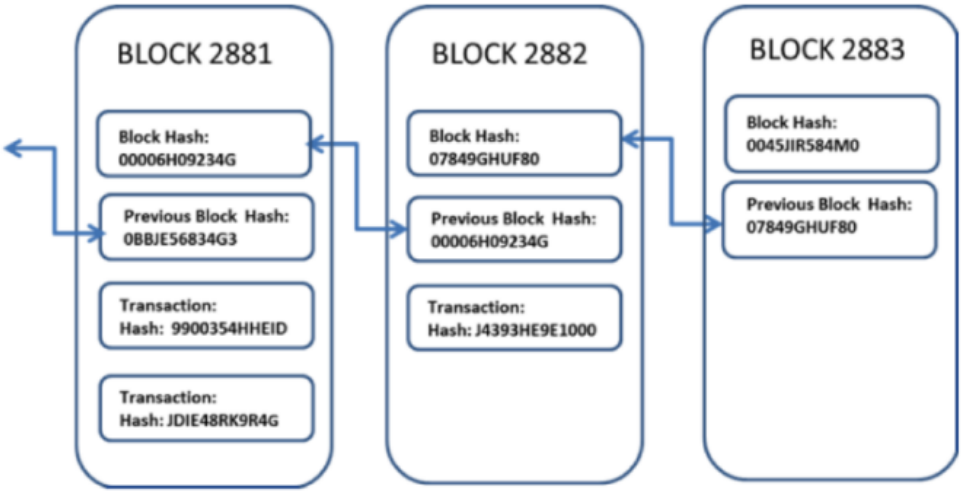


Figure 1.1: Block contents [1]

- **Block** as a data structure within the blockchain contains different functions, which include transaction hashes and some other additional information for blockchain technology. Gavin Wood et al. [29] described some relevant information below, and we used this information 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. The miner adds a new block, and others check the validity of the new block. Any participant can take part in the mining pool, but the chance of finding a valid block depends on the power of the computer to perform calculations. Sometimes a miner will find an uncle block. 'An uncle block is a block that is initially valid but is surpassed by another faster block'. An Uncle block is rewarded with $\frac{7}{8}$ of the full block value, and a hash will be added to a valid block. A maximum of two uncle blocks can be added to a valid block, and the miner of the valid block also receives $\frac{1}{32}$ extra Ether for each uncle block [28].
- **Mining pool** is a way that miners gather together, start mining, solve blocks, and win rewards, and then the reward is split among the involved miners [28].

1.6.2 Ether

It is the form of payment and fuel for Ethereum. The base for mining (finding the solution and adding blocks) successfully is five ethers. If the miner finds a solution but not fast enough, it becomes less ether, like 4.375 ether, and will be an uncle block. Each block can contain just two uncle blocks and receive $\frac{1}{32}$ per uncle block. If another miner also finds a solution, this block cannot be added to the blockchain, and the miner just receives 2-3 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 ether or 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 not be changeable anymore. All information about this account is stored in the database under the corresponding hash code for later retrieval [29].

Hash Function The process of SHA3 standardization at the National Institute of Standards and Technology (NIST) was completed in August 2015. This standard specifies the SHA3 (Secure Hash Algorithm-3) family of functions for binary data. Each hash function is based on the *Kacak* algorithm, known as NIST, the winner of the SHA3 cryptographic hash algorithm competition. The SHA3 family includes four functions with different lengths of 224, 256, 384, and 512 bits. In the hash function, the input is called a *message* and the output is called a *hash value*, where the length of the message can vary but the length of the hash value is always fixed [10].

"Gas is fuel on the Ethereum platform that is paid before executing a transaction. In a case that the transaction is rolled back, the consumed gas will not be returned [28].

- **Transaction** Gavin Wood et al. [29] described a transaction as a cryptography-signed instruction that is executed by an external actor, which can be a human or another contract. 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 fired by a contract and contains all attributes the same as a transaction, but $gasPrice$ [28].

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

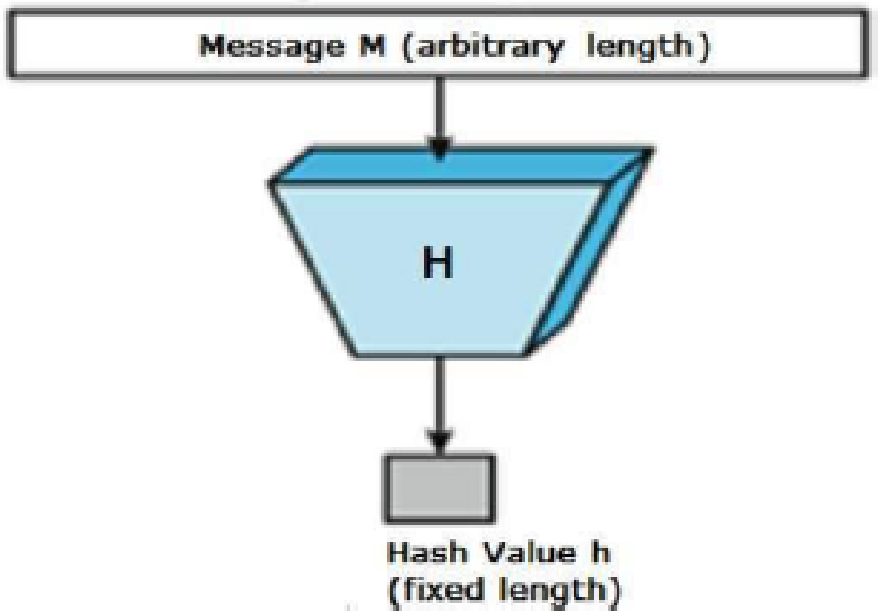


Figure 1.2: Image of hash function [10]

1.6.4 Contract

It is an account on the Ethereum blockchain that has its own code and is controlled by code. The code inside the contract is triggered whenever it receives a message, allowing it to read and write contract storage or send a message.
A contract in Ethereum is an autonomous agent that performs some operations that are programmed to fulfill the user's goals, meaning that the contract is an autonomous agent that is executed when it receives a message or transaction, having control over its balance and the key/value store as constant variables. The keys and values stored in the contract are long-lasting and get used whenever the contract starts running [28].

1.6.5 Smart Contract

The term smart contract was coined in 1994 by Nick Szabo [23] who realized that DLT can also be used for smart contracts. According to Nick Szabo *'Smart contract is a*

computerized transaction protocol that executes the terms of a contract'[2]. He envisioned a way to write agreements that enforce the conditions between parties involved in a transaction automatically and more efficiently. Smart contracts are run by each node as part of the block creation process. Block creation occurs when transactions take place in the blockchain. An important part of a smart contract is that each contract has its own address. Since the contract code is carried to the transaction, a node can create the specific transaction, assigning an address to the contract, and this transaction is capable of running contract code at the time of creation. After that, the contract will be part of the block, and the address will never change. Whenever the node wants to call a method inside the contract, it should send a message to the address of the contract that has the method and input data. The contract will run as part of the creation of a new block and then return value or store data on the blockchain [19].

Solidity is a high-level, Turing complete language with a JavaScript-like syntax. The contract is similar to classes in an object-oriented language, which contain fields for persistent storage of contracts and methods to be invoked by internal and external transactions. For interacting with another contract, we either need to create a new instance of this contract or make a transaction to a known contract address.
In principle, Solidity provides some basics to access blocks and transaction details, like: *msg.sender* for accessing the address of an account or *msg.value* to access the amount of *wei* transferred by the transaction. It also uses some functions to transfer money to another contract, such as *call* and *send*. These functions get used to transfer value and translate into an internal call to a transaction, which causes the contract to also execute code or may fail to execute due to insufficient gas [13].

[2]https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vv

Chapter 2

Blockchain as the Infrastructure of Semantic Web

A distributed ledger is increasingly used to represent transactions between multiple parties throughout the world. It is difficult to search for specific information in a distributed ledger without an index. Therefore, indexing data in the distributed ledger is a requirement that provides the ability to search across multiple ledgers, enhancing the power and usability of this system as well [4].

2.1 Distributed Ledgers and Indexing

A distributed ledger based on a blockchain does not have central control. Blockchains are organised into multiple blocks; the initial block is created manually, and the other blocks are added by some consensus process between nodes. The Ethereum smart contract provides the possibility to automatically control what happens with cryptocurrency on the blockchain without involving untrustworthy external sources. Ethereum smart contracts have an account which can normally store, update, or perform a function with the input and output. As smart contracts are time-ordered, where data is stored in blocks, the data must be indexed. Indexing the smart contract gives us the capability to access the data, search, analyse services on the distributed ledger, and expose them to the outside world for more interactions. There are different levels of indexing smart contracts: The basic level is the fundamental level for the next step. It indexes basic entities such as accounts and blocks related to distributed ledgers, and data can be stored or retrieved here. At the functional level, smart contracts contain a lot of functional interfaces that depict the other functionality of platforms such as Ethereum [4].

2.1.1 Why Do We Use Ontology for Blockchain?

Generally, blockchain is a distributed database that is replicated over all nodes and uses a cloud computing architecture. In other words, data in the database is distributed across many organizations. Thus, data should have a common interpretation to be understandable for organizations. These interpretations are applicable via a formal specification that enables verification and inference within software and applications executed on the network.
This is where ontology plays the main role in ensuring a common interpretation of data in the shared database among different enterprises. 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 automate inference and verify the operation of the blockchain. In other words, blockchain modeling based on a formal ontology can help with the development of smart contracts to execute on the blockchain [12].

Also, we can use ontologies to capture data within the blockchain: On the one hand, it facilitates a better understanding of blockchain concepts for humans. On the other hand, it enables interlinking with other linked data to convey deductions and formal reasoning [12]. Vocabulary used within an ontology increases the transparency of transactions by describing the transaction in the context of linked data and facilitating the graphical representation of the location of such transactions. Thus, it also increases the capability of analysis by users [12].

2.1.2 Linked Data

According to Tim Berners-Lee et al. [17], 'Linked data is about using the Web to create typed links between data from different sources'. When information is presented as linked data, other related information can be easily retrieved, and new information can also be easily linked to it. *Berners-Lee* described four rules for linked data:

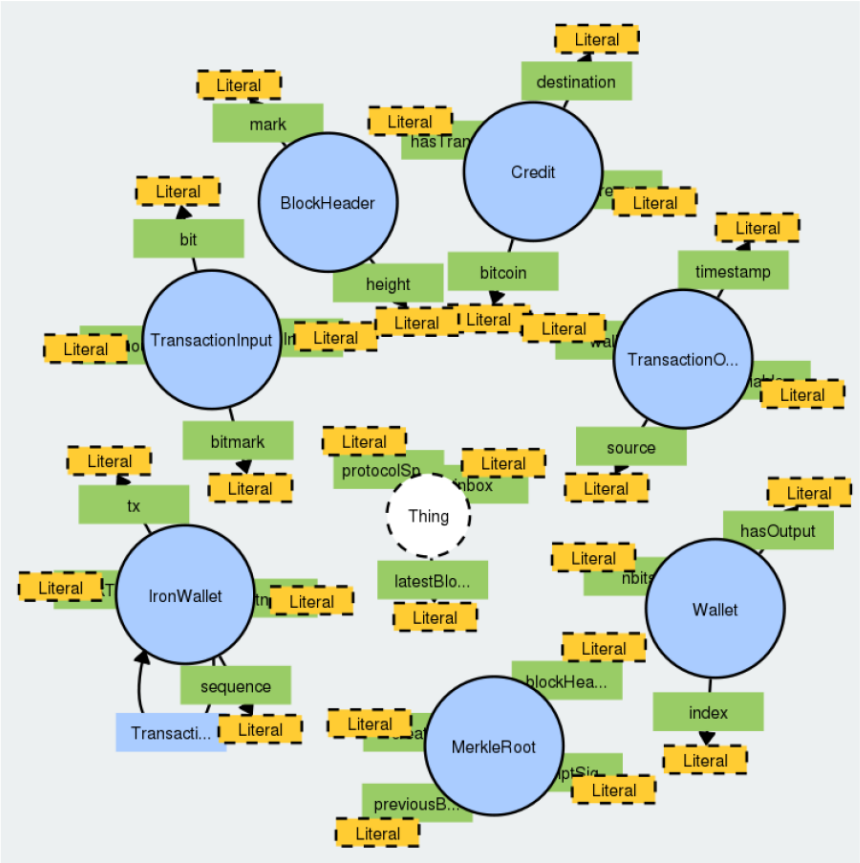


Figure 2.1: Illustration of ontology diagram [5]

- **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 [26].

2.1.3 RDF

The Resource Description Framework (RDF) is a family of W3C specifications. RDF is used to describe and model information. It describes a subject that predicts an object and is called a triple. i) Subjects that RDF expressions describe. ii) Predicate specific

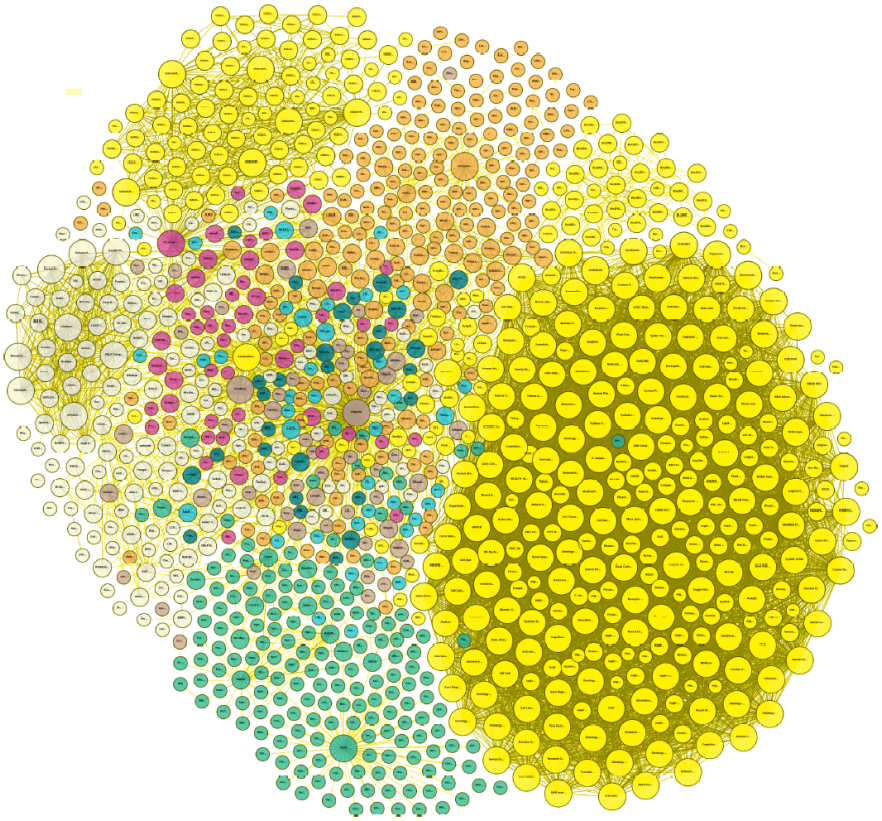


Figure 2.2: Linked data diagram [26]

properties, attributes, or relationships to describe a resource. iii) The object is the name of a property or value [26].

2.1.4 SPARQL

According to Wikipedia's definition, 'it is a semantic query language for a dataset that enables us to retrieve and modify data stored in RDF format known as triples'. SPARQL can query one, two, or all elements of triples.

2.1.5 OWL (Ontology Web Language)

Ontology Web Language is designed to represent knowledge about things and the relations among them. OWL is a computational logic-based language, which means the

language modeled in OWL can operate in a computer program such as negation, intersection, and so on [26].

2.1.6 Evolution of World Wide Web

The evolution and interaction of people on the Internet are classified based on three technologies:

- *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*, has the capability of providing space for users to collaborate on content creation or modification.

- *Web 3.0* is strengthened by the Semantic Web, where people have access to linked information on the web. Recently, with the arrival of distributed technologies like blockchain and Ethereum which are used by Web 3.0, there has been a new focus on this [11].

2.2 Vocabularies

2.2.1 Vocabulary in Distributed Ledger

Generating linked data requires the use of a standard ontology or vocabulary to explain blockchain concepts. Interfaces between distributed ledgers and the semantic web are still in their early stages. There are some systems and vocabularies that specify such a vocabulary like FlexLedger, EthOn, and BLONDiE [4].

FlexLedger describes HTTP interfaces to blockchains, with a standard vocabulary and responses to these interfaces. It is a protocol for decentralized ledger and graph data models that represent ledger creation, querying, and data modelling using JSON-LD. However, FlexLedger does not have an explicit vocabulary about ontology, nor does it have a concrete ontology for itself.
It is noteworthy to say that the FlexLedger is not suitable to implement in some graph

models like graph chains because in the FlexLedger meta and the content data are stored together in the same graph, whereas the GraphChain blocks content is stored outside the blockchain in a separate graph [25].

EthOn is an OWL ontology that describes blockchain classes such as *'blocks', 'accounts', 'messages', 'states'* and relations such as *'has parent block'* [6].

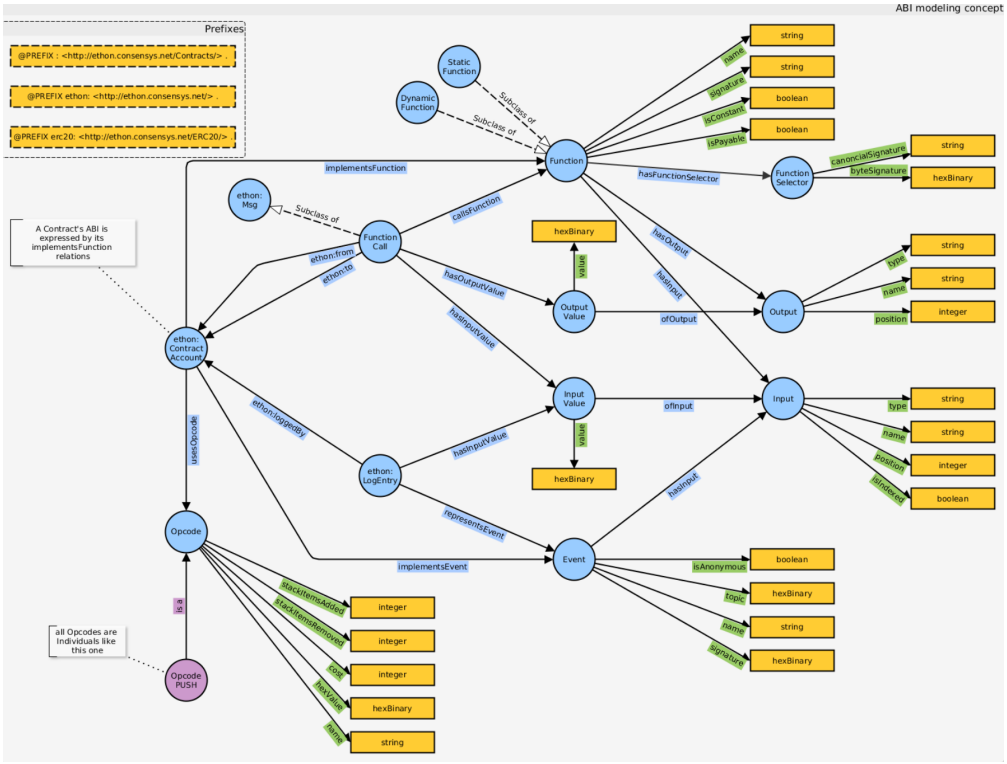


Figure 2.3: EthOn contract model (blue arrow is object properties, green arrow is data properties, purple circle is an instance and blue one is a class) [6]

BLONDiE (Blockchain Ontology with Dynamic Extensibility) is another OWL ontology for describing the blockchain structure, like EthOn. But it is more generic than EthOn. For example, EthOn and BLONDiE both define some terms such as 'account', 'block', and 'transaction', as well as some attributes such as 'transaction payload' or 'miner address'. BLONDiE defines some other concepts for different blockchains, such as 'BitcoinBlockHeader' and 'EthereumBlockHeader' as subclasses of 'BlockHeader'. At the

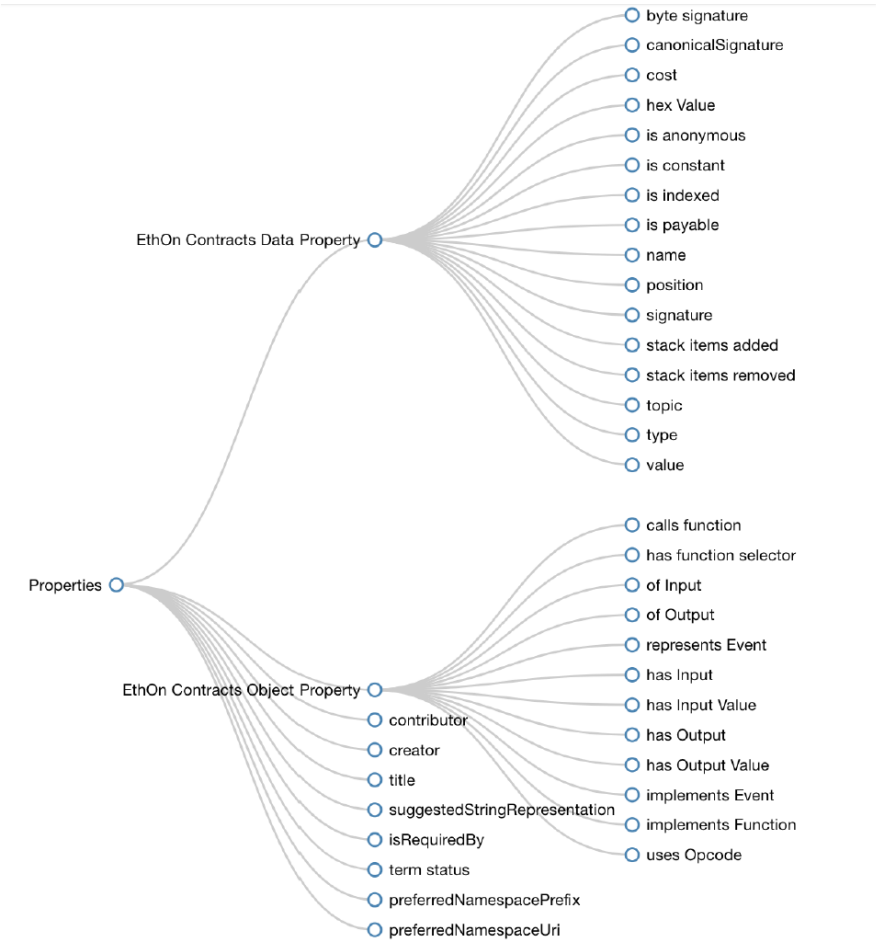


Figure 2.4: EthOn properties [6]

moment, BLONDiE supports two cryptocurrencies like Bitcoin and Ethereum, where all links and relations between objects and attributes are represented in RDF (Resource Description Framework) [4].

2.2.2 Vocabulary in Smart Contract

As mentioned earlier, EthOn and BLONDiE are both similar concepts that can be used for smart contracts. There are many works on semantic annotation of the web and HTTP-APIs that may enable us to annotate smart contracts as well. However, the contracts are not web APIs, and the implementation may differ, but the main concept

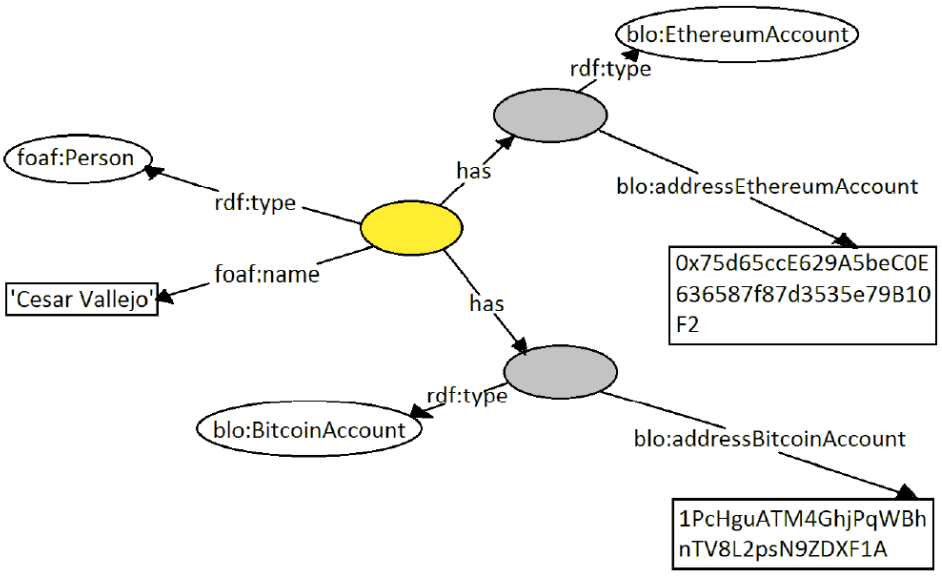


Figure 2.5: BLONDiE usage example [26]

does not. In other words, the vocabularies used to annotate web services are also used to annotate smart contracts. It seems that the combination of a distributed ledger with a smart contract and web service due to profitability will become commonplace [4].

2.3 Semantify Blockchain

2.3.1 Semantic Blockchain

With increasing the use of blockchain technology recently, the need for semantic reasoning on the distributed ledger is on the rise as well. The blockchain is the best platform to utilize semantic web principles in this technology and add a new trusted property to a dataset. Using semantic web technology on the blockchain is a novel idea and the way to apply this technology in blockchain and smart contracts is also a controversial issue. There are some **definitions of semantic blockchain:**

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

- Semantic blockchain is the application of semantic web standards on the blockchain; these standards are based on RDF [26].

2.3.2 Semantification Process

Semantic blockchain or semantic distributed ledger affects the industrial world and, consequently, the result leads to the development of new applications and frameworks to combine the two worlds. There are some ways to semantify blockchain, as follows:
- Mapping the blockchain data to RDF, making use of vocabulary, ontology, and so on. Storage of data in a blockchain is expensive, so the only way is to store the hashpoint to the dataset in the blockchain and then share RDF on the blockchain. Creating a semantic blockchain that exchanges internal data protocols in RDF format [26].

2.3.3 Semantic Ontology Mapping Using BLONDiE

To generate RDF, it needs to map the basic blockchain entities to relevant semantic web terms, concepts, and ontologies. To make the query more efficient, BLONDiE has been extended in two ways: Firstly, records relating to both blocks and transactions have been augmented with an attribute for the hash of each entity.Secondly, records relating to the transactions have been augmented with links to entities such as blocks or smart contracts. Blockchain stores just a binary form of each contract with metadata. To interact with the contract, the Application Binary Interface (ABI) specification is required. This specification is in the form of JSON and is created when a smart contract is compiled and stored on the blockchain. The ABI determines all functions of contracts and provides descriptions of input and output parameters for each contract [4].

```
"abi": [
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": true,
        "internalType": "uint256",
        "name": "hash",
        "type": "uint256"
      },
      {
        "indexed": false,
        "internalType": "string",
        "name": "uri",
        "type": "string"
      }
    ],
    "name": "modifiedLicense",
    "type": "event"
  },
  {
    "constant": false,
    "inputs": [
      {
        "internalType": "uint256",
        "name": "hash",
        "type": "uint256"
      },
      {
        "internalType": "string",
        "name": "uri",
        "type": "string"
      }
    ],
    "name": "licenseData",
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
  },
```


Figure 2.6: Smart contract application Bbinary interface (ABI)

Chapter 3

DALICC

According to Pellegrini [21], DALICC stands for Data Licensing Clearance Center. It is a software framework that helps people legally secure data from a third party, detect licensing conflicts, and reduce the costs of rights clearance by enabling automated clearance rights in the creation of derivative works [21].

Rights Expression Language (RELs) are used to express a machine-readable form of rights for access management and digital asset. The most popular RELs vocabularies include MPEG-21, ODRL-2.0, ccREL, XACML and WAC that serve the rights management in different areas like data licensing, application, etc[18].

3.1 DALICC Requirements

The following requirements would be addressed by the DALICC framework:

- *Tackling license heterogeneity:* It is possible to combine the various contents that have the same license with different names. But that would make it much more difficult to license the result of the combined contents. To solve this issue, DALICC provides a set of machine-readable representations of licenses that allow us to compare licenses to each other to identify equivalent licenses. It guides the user to possible conflicts between various combined licenses.
- *Tackling REL Heterogeneity:*
  Combining licenses is simple if they are expressed through the same RELs. But it

is difficult to compare licenses that have been represented by different RELs. DALICC resolves this problem by representing RELs based on the Semantic Web and mapping the terms to each other. It will represent existing RELs based on W3C-approved standards, thus allowing mapping between various RELs to be created.

- *Compatibility Check, Conflict Detection, and Neutrality of the Rules:*
  It is difficult to be sure the meaning of the different terms in semantics is aligned. These problems result from indicating the classes, instances, and properties, which cannot be handled just by mapping.
  This is where DALICC comes into play and helps the user with a workflow that defines the usage context and then gathers additional information to detect conflicts and ambiguous concepts. Based on this information, DALICC makes reasoning over the set of licenses and infers instructions to the user on how to process the license processing.

- *Legal Validity of Representations and Machine Recommendations:*
  According to Anna Fensel et al. [18], the semantic complexity of licensing issues means that the semantics of RELs must be aligned within the specific application scenario. This includes a correct interpretation of the various national legislations according to the country of origin of a jurisdiction (i.e., German Urheberrecht vs. US copyright), the resolution of problems that are derived from multilingualism, and the consideration of existing case law in the resolution of licensing conflicts [18].
  To solve this problem, DALICC will check the legal validity of machine-readable licenses and the compatibility of reasoning engine output with the law. DALICC output will be tested against the law, checked for semantic precision derived from different languages, and adjusted accordingly [18].

3.2 DALICC Software Architecture

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

- **License composer** is a tool that allows the license to be created from a set of questions that are mapped to ODRL, ccREL, and the DALICC vocabularies and concepts.
- **License library** is a repository that represents licenses in a machine-readable format, the former as ODRL policies and the latter as plain text.
- **License annotator** allows attaching licenses to the dataset, either by choosing available licenses in the license library or creating a new license using the license composer.
- **License negotiator** is the main component in the DALICC framework that checks the license compatibility and supports license conflicts by detecting equivalence licenses having various names [18].

Chapter 4

Implementation: DApp

DApp is often described as P2P, trustless with a special characteristic that there is not a single server to control it. DApp includes at least one user interface or frontend that could be a mobile app, desktop application installed on a computer. The data relating to the application could be provided by a single group or company or may be provided by end users themselves. DApp uses Ethereum as the backend of its data storage and processing using smart contracts. The user interface of DApp is usually like traditional websites but is one website plus one or multiple smart contracts [16].

4.1 Ethereum DApp

The benefits of using the Ethereum blockchain in DApps are as follows [16]:

- 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.

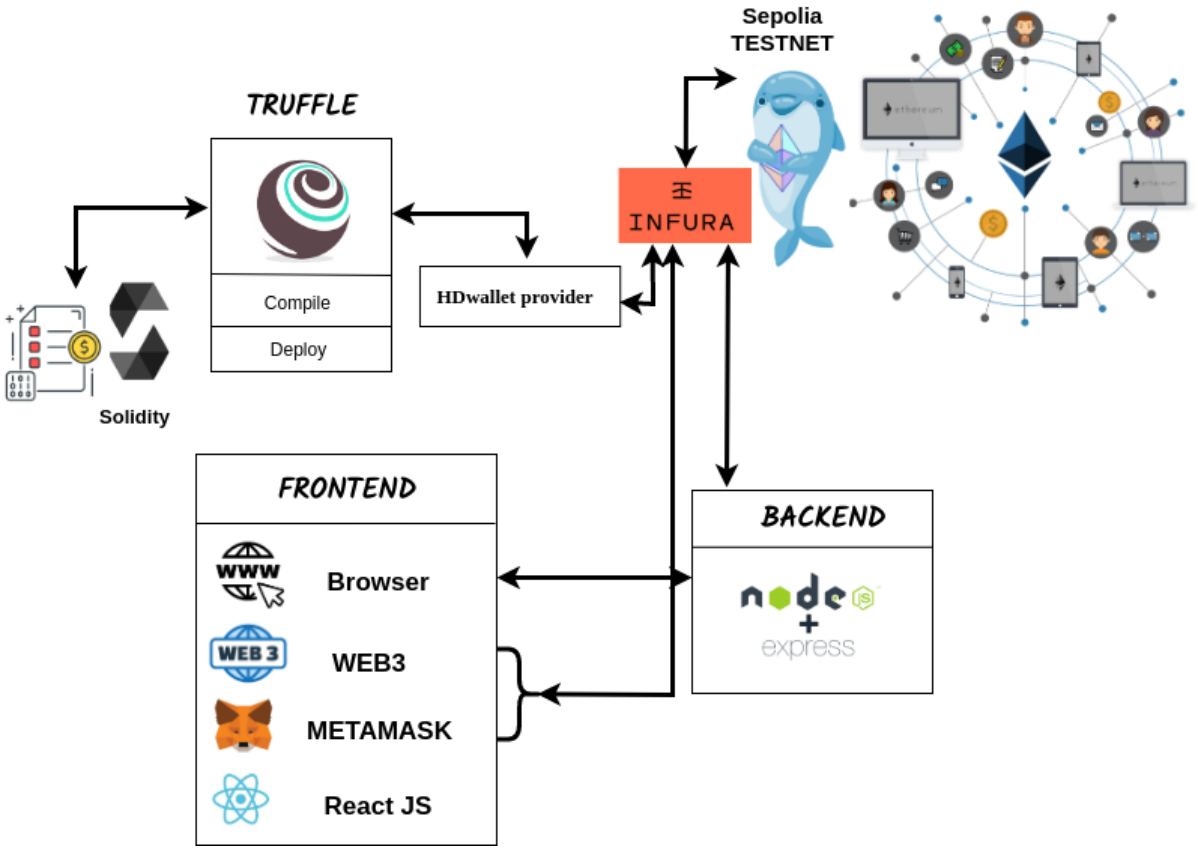


Figure 4.1: DApp infrastructure

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 their 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

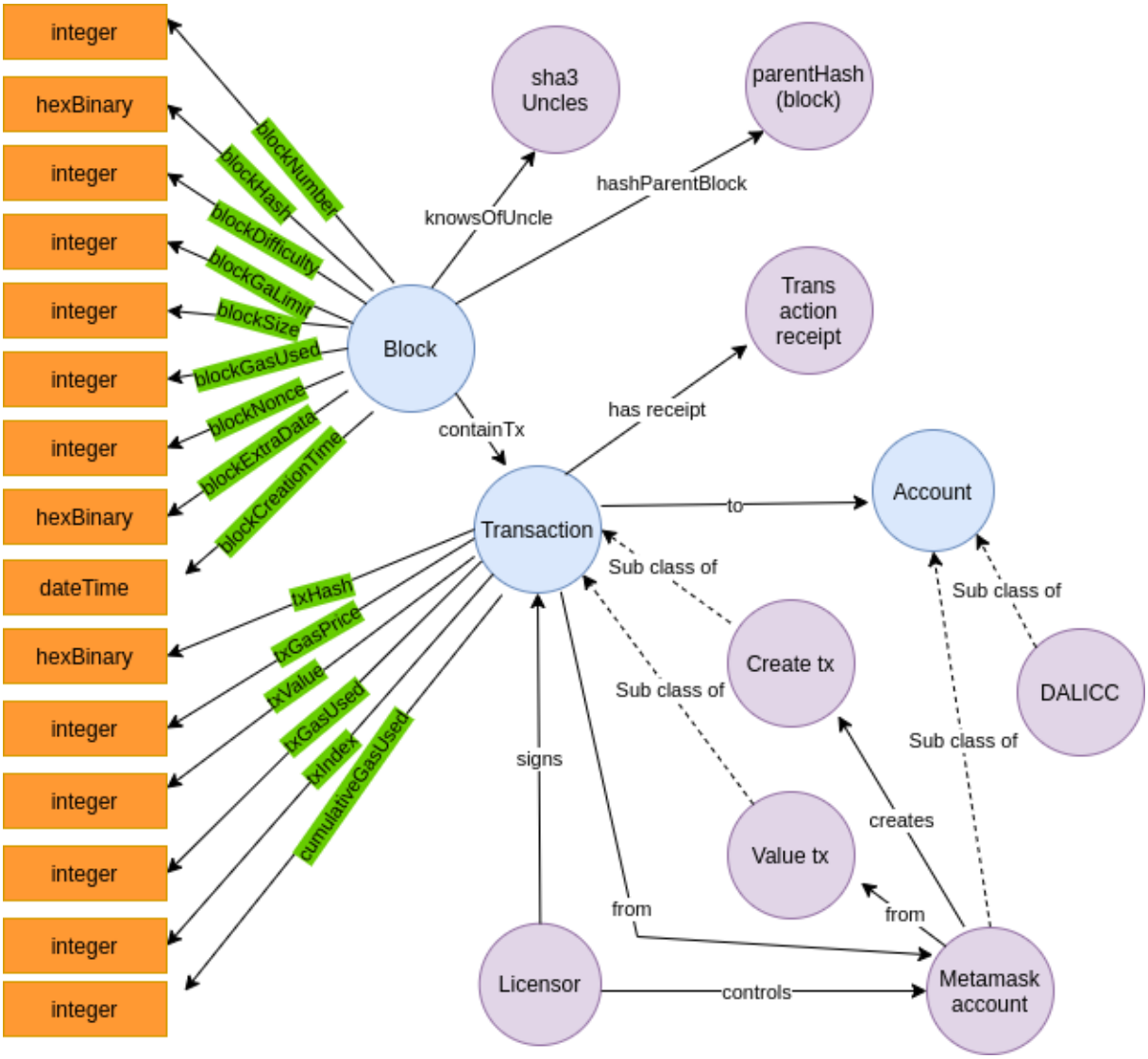


Figure 4.2: Transaction illustration

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

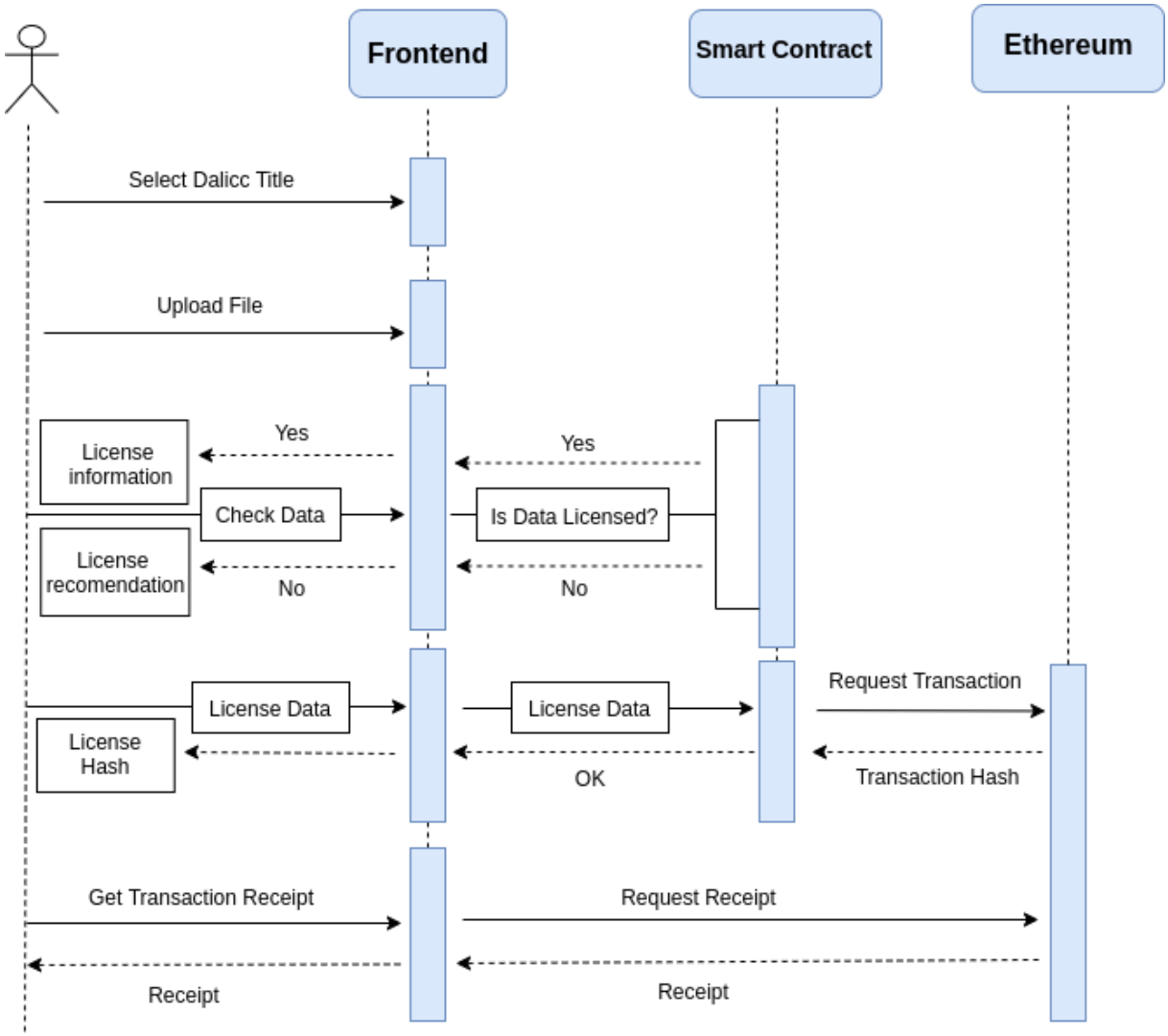


Figure 4.3: Sequence diagram for licensing file

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

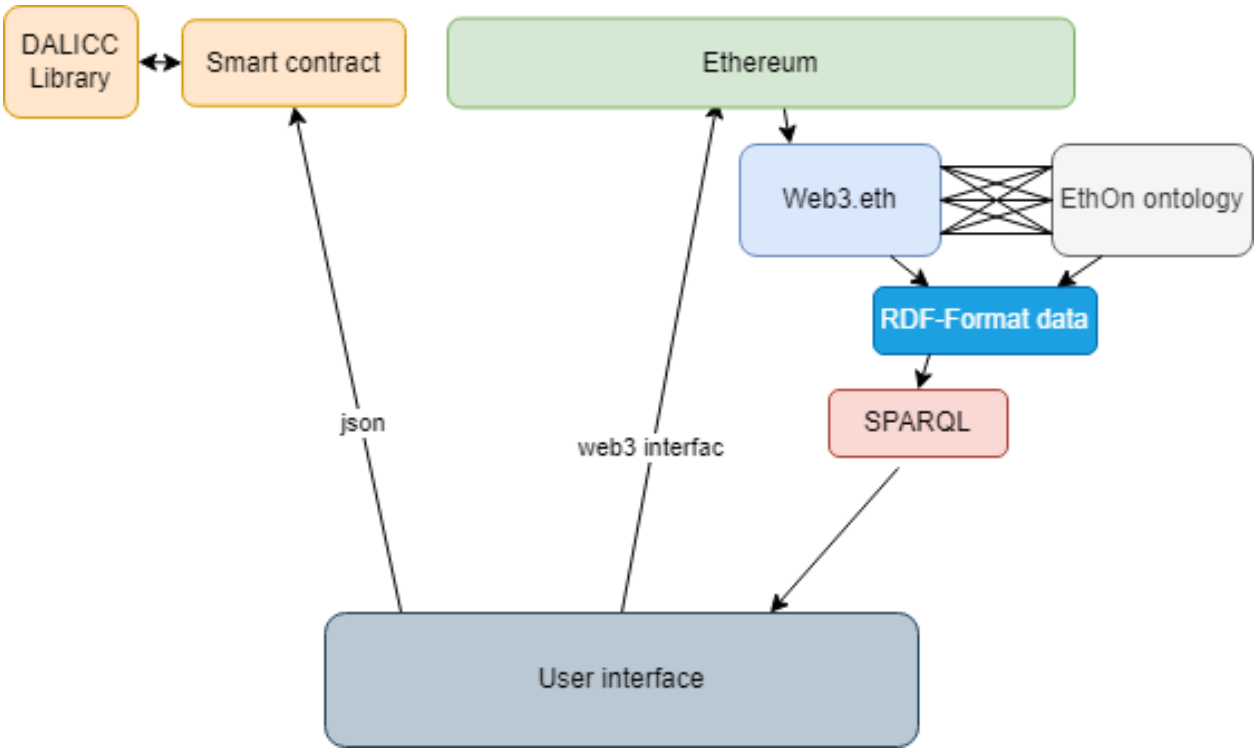


Figure 4.4: DApp architecture

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?

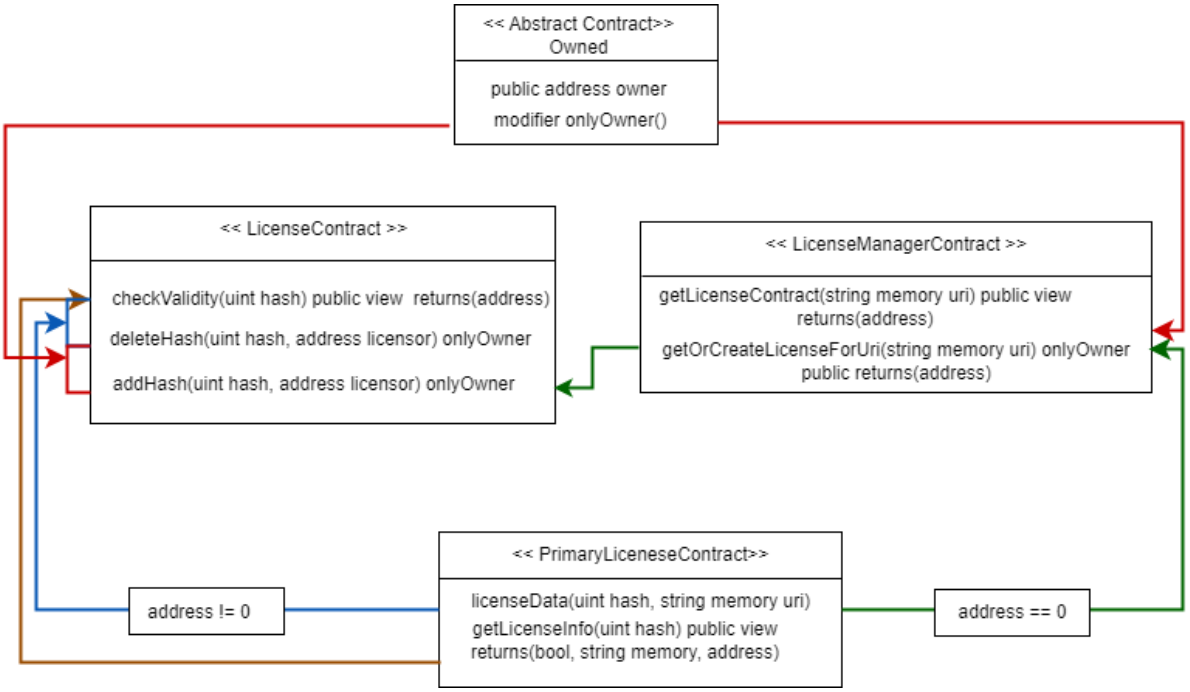


Figure 4.5: Smart contract visualization

- **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.

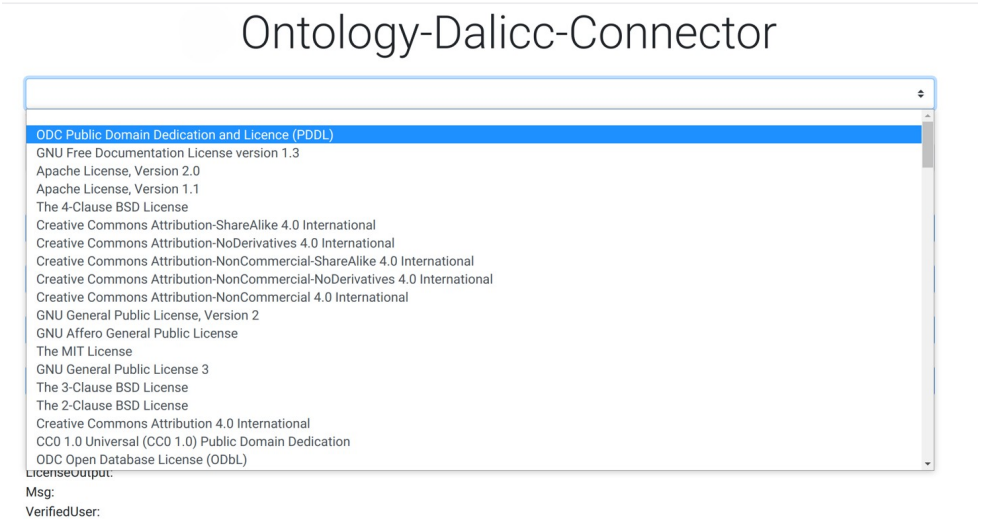


Figure 4.6: Define type from DALICC library

- 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.

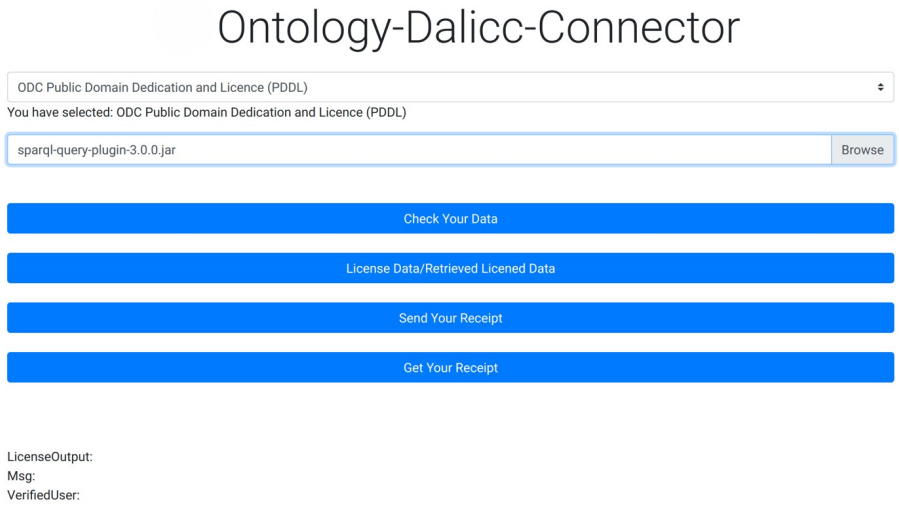


Figure 4.7: Define file from local device

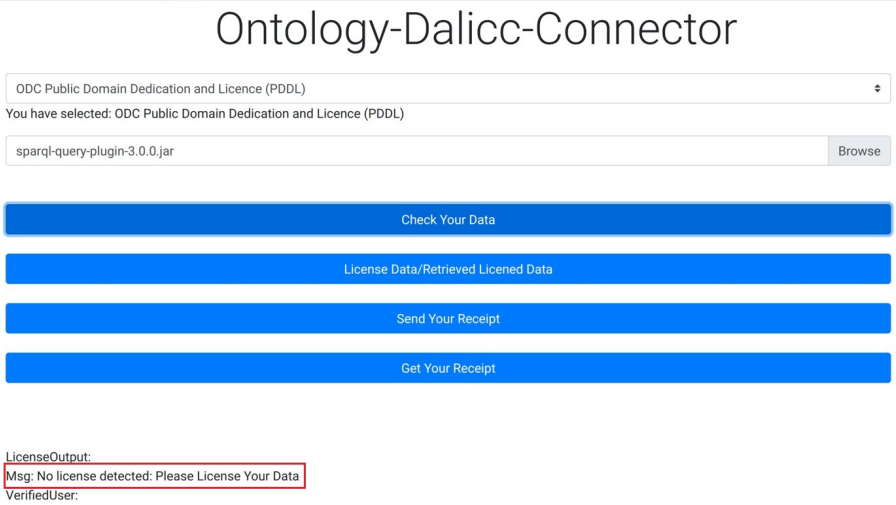


Figure 4.8: Check if file is already licensed?

- 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

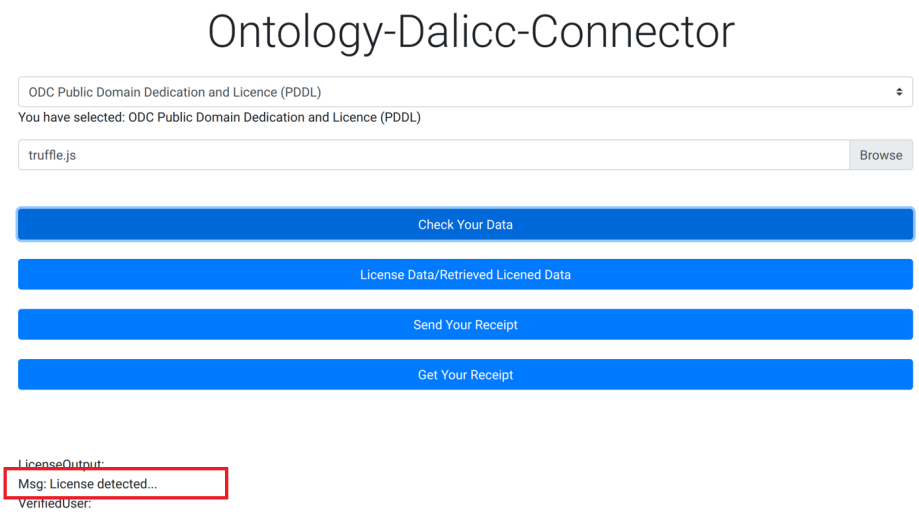


Figure 4.9: Message for licensed data

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.

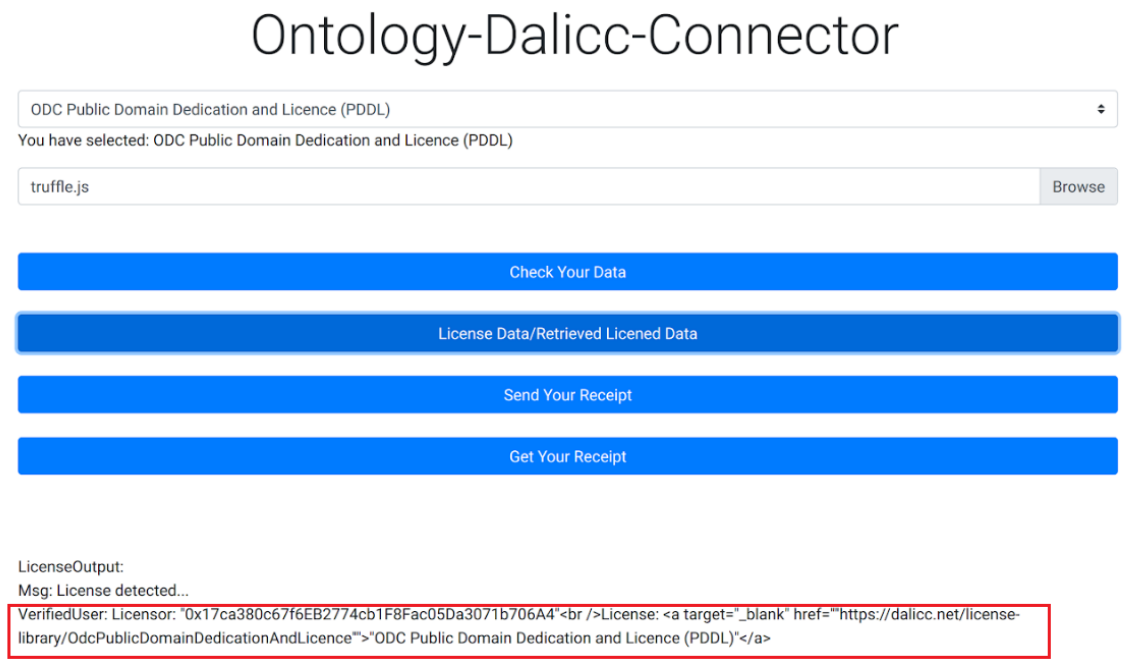


Figure 4.10: License information for licensed data

- 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.

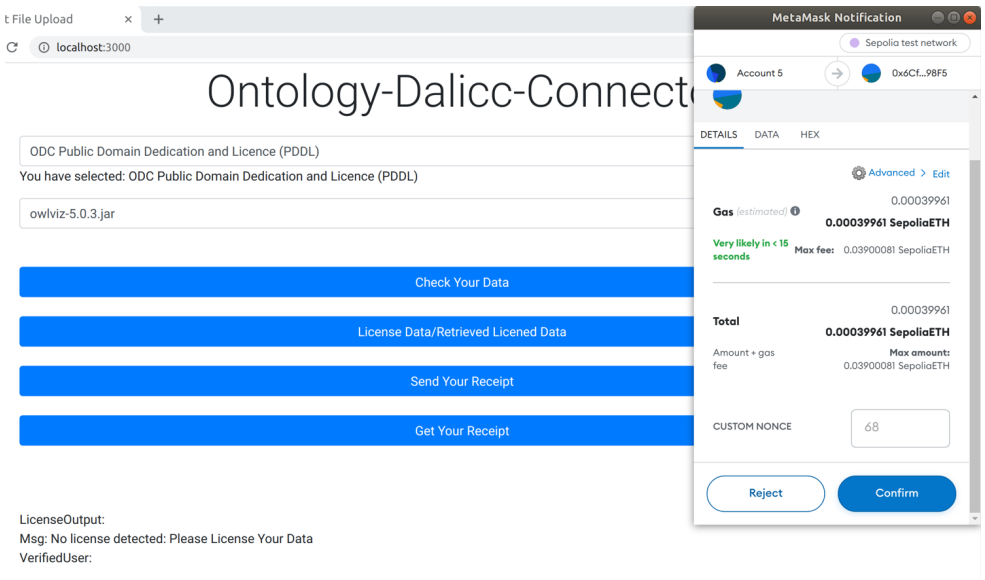


Figure 4.11: Licensing data process

- 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.

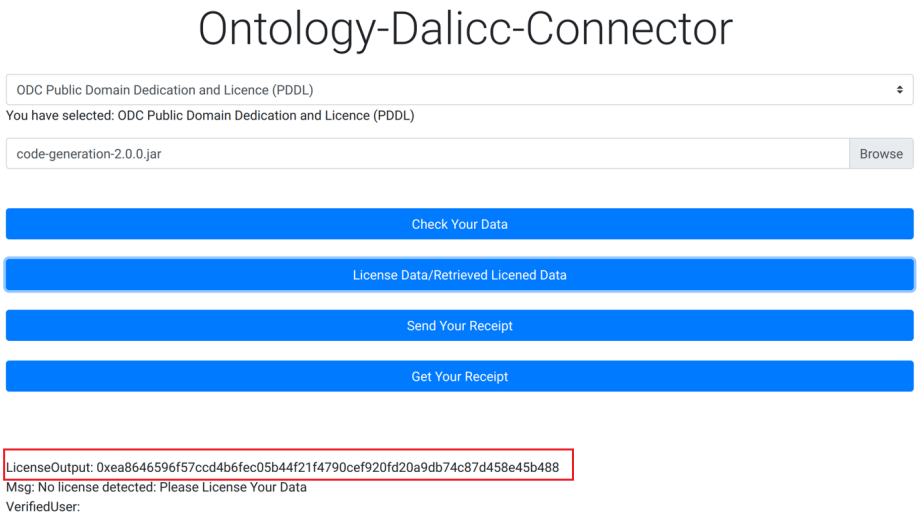


Figure 4.12: License hash after confirming transaction

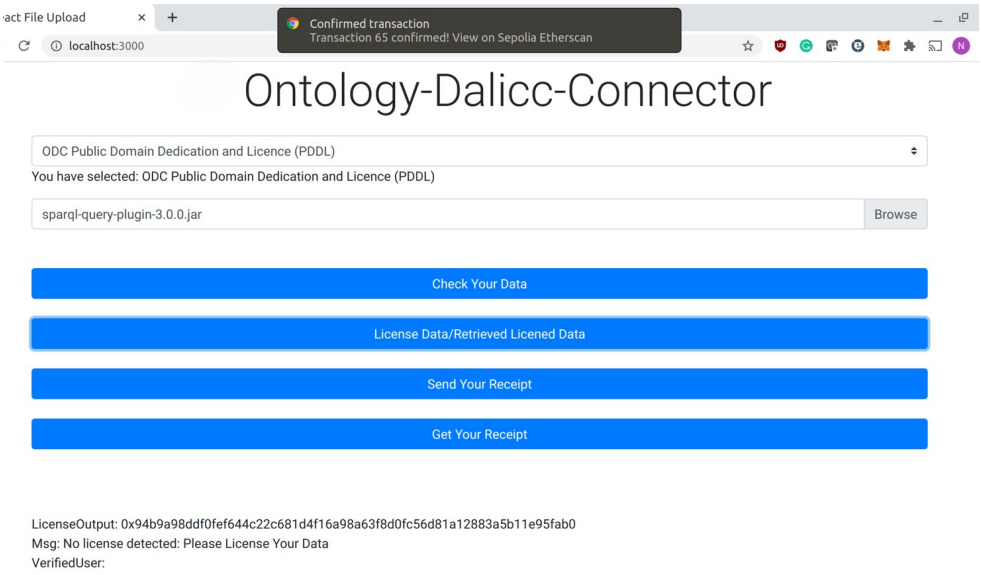
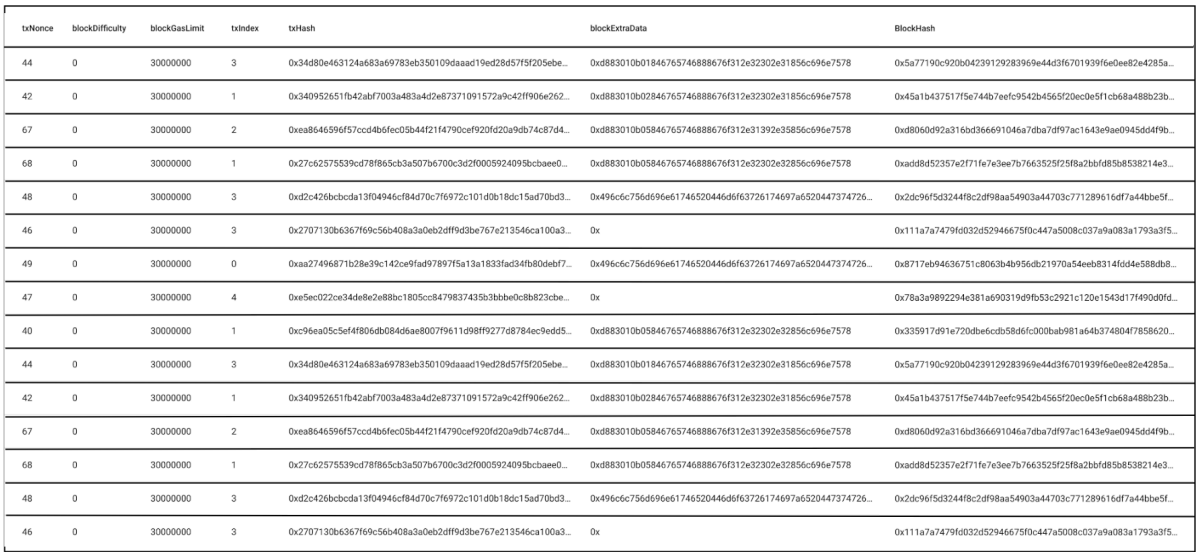


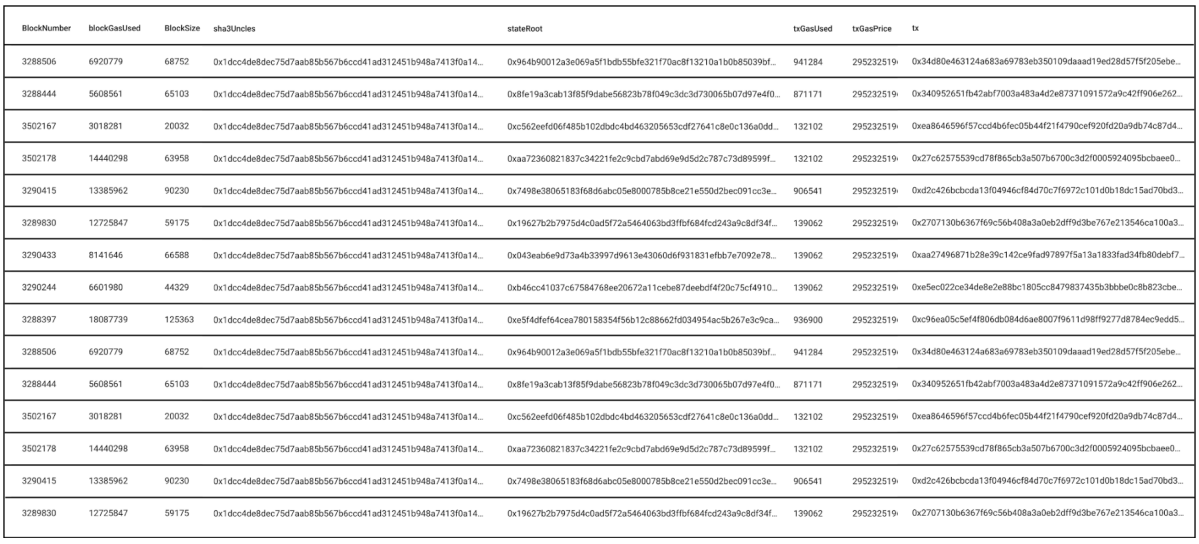
Figure 4.13: Transaction confirmation

- 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 contributes to integrating semantic licences from the DALICC library with smart contracts and some problems that are still open in the literature, especially when considering the few works that integrate blockchain with the semantic web.
For this purpose, we focused on two things as the infrastructure of our work: One is the Ethereum platform, which is developed and implemented to keep the system reliable, secure, autonomous, and decentralized. The second one is Ethereum ontology, which formalizes the concepts and terms of the Ethereum blockchain in OWL, describing the Ethereum objects as classes in ontology. It covers the main blockchain concepts such as blocks, accounts, transactions, contract messages, and the relationship between the instances of these classes. Therefore, a licensing system has been created which keeps the integrity of license information and also provides a semantic view of such a deployment environment using Ethereum ontology concepts.
To do this, the user is guided to start licensing and get their licensing receipt step by step.
This DApp also has its limitations and, for most of them, solutions have been defined. Thus, in this work, we have developed the basic methods for the license attachment to data and also developed semantic mapping for the result of this attachment process as a receipt.

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;
    }
}
```

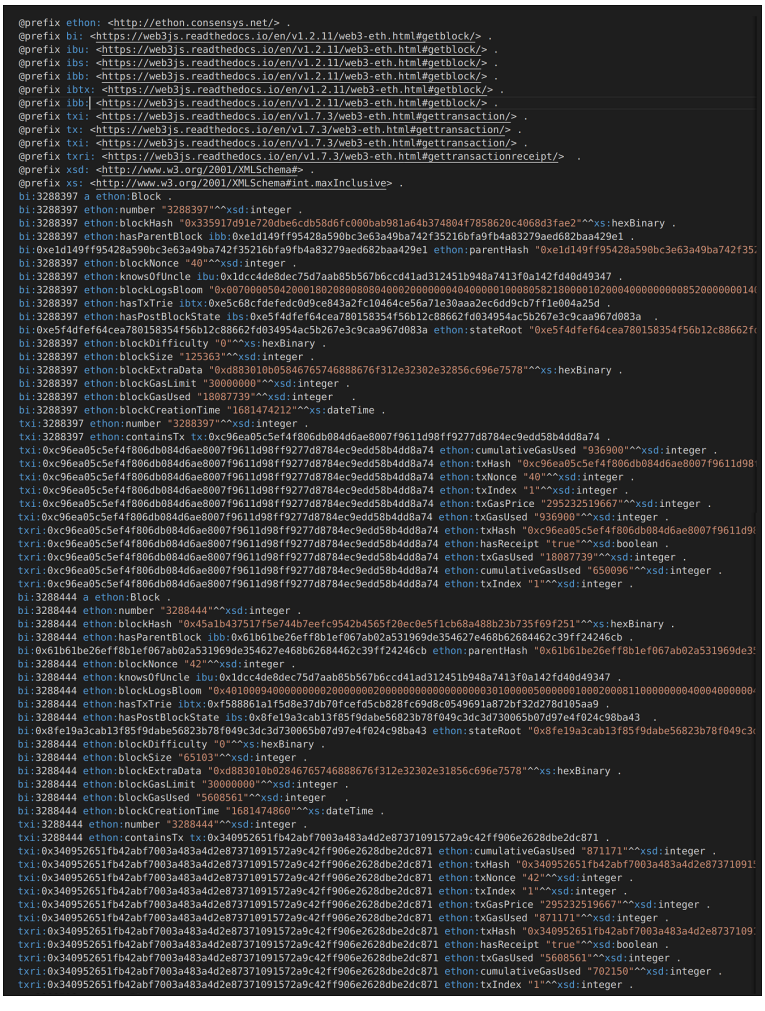
Appendix G

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] Paul Baran. On distributed networks. 51, 1964. URL https://web.stanford.edu/class/cs244/papers/DistributedCommunicationsNetworks.pdf.

[2] Kilian Hinteregger David Gatta. Creation of national semantic licenses and their attachment to data and content, 3 2020.

[3] Konstantinos Christidis, Michael Devetsikiotis. Blockchains and smart contracts for the internet of things, 3 2016.

[4] Allan Third, John Domingue. Linked data indexing of distributed ledgers. In *Companion Proceedings of the 26th International Conference on World Wide Web Companion*, volume 8, 5 2017. doi: 10.1145/3041021.3053895.

[5] Matthew English, Soeren Auer, John Domingue. Block chain technologies and the semantic web: A framework for symbiotic development. 15, 2016.

[6] Umar Rashid, Allan Third, John Domingue. Web service for semantic negotiation of smart contracts. 6, 1 2018. URL https://openreview.net/forum?id=BJg2gCu1e7.

[7] P.K. Paul, P.S. Aithal, Ricardo Saavedra, Surajit Ghosh. Blockchain technology and its types. 13, 2021. URL https://www.researchgate.net/publication/359051731.

[8] Manav Gupta. *Blockchain for dummies*. John Wiley, 2018.

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

[10] 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.

[11] Bijoy Chhetrin, Pranay Kujur. Evolution of world wide web. 6, 2015. URL https://www.researchgate.net/publication/280944777.

[12] Henry M. Kim, Marek Laskowski. Toward an ontology-driven blockchain design for supply-chain provenance. 18, 2018. doi: doi:10.1002/isaf.1424.

[13] Ilya Grishchenko, Matteo Maffei, and Clara Schneidewind. A semantic framework for the securlty analysis of ethereum smart contracts. 27, 2018. doi: https://doi.org/10.1007/978-3-319-89722-6_10.

[14] Kalimeris Markos. Indexes for blockchain data, 2 2019.

[15] Faraz Masood. An overview of distributed ledger technology and its applications. 7, 2018. doi: 10.26438/ijcse/v6i10.422427.

[16] William Metcalfe. Ethereum, smart contracts, dapps. 17, 2020. URL https://doi.org/10.1007/978-981-15-3376-1_5.

[17] Christian Bizer, Freie Universitat Berlin, Germany Tom Heath, Talis Information Ltd, United Kingdom Tim Berners-Lee, Massachusetts Institute of Technology, USA. Linked data - the story so far. 27, 2009. doi: 10.4018/jswis.2009081901ÂůSource:OAI.

[18] Anna Fensel, Tassilo Pellegrini, Simon Steyskal, Oleksandra Panasiuk. Automated rights clearance using semantic web technologies. 16, 2018. doi: 10.1007/978-3-662-55433-3_14.

[19] Sebastian Payrott. introduction to ethereum and smart contracts. 68, 2017. URL https://auth0.com/blog/an-introduction-to-ethereum-and-smart-contracts-part-2/.

[20] Max Luke, Anna Dimitrova, Stephen James Lee, Zdenek Pekarek. Blockchain in electricity: a critical review of progress to date. 38, 2018. URL https://www.researchgate.net/publication/332138382.

[21] Tassilo Pellegrini. Detecting licensing conflicts with dalicc. 5, 2017. URL http://ffhoarep.fh-ooe.at/bitstream/123456789/1025/1/Panel_120_ID_193.pdf.

[22] Sukrit Kalra, Seep Goel, Mohan Dhawan, Subodh Sharma. Zeus: Analyzing safety of smart contracts. 15, 2 2018. URL http://dx.doi.org/10.14722/ndss.2018.23082.

[23] Nick Szabo. Smart contracts: Building blocks for digital markets. 11, 2018. URL http://www.truevaluemetrics.org/DBpdfs/BlockChain/Nick-Szabo-Smart-Contracts-Building-Blocks-for-Digital-Markets-1996-14591.pdf.

[24] Pablo Lamela Seijas , Simon Thompson and Darryl McAdams. Scripting smart contracts for distributed ledger technology. 30, 12 2016.

[25] Mirek Sopek, Przemyslaw Gradzki, Witold Kosowski , Dominik Kozuski, Rafa Troczak, Robert Trypuz. Companion proceedings of the the web conference 2018. In *GraphChain – A Distributed Database with Explicit Semantics and Chained RDF Graphs*, volume 8, 5 2018. doi: 10.1145/3184558.3191554.

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

[27] Jose Luis Romero Ugarte. Distributed ledger technology. 11, 11 2018.

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

[29] DR.Gavin Wood. Ethereum: A secure decentralized generalised transaction ledger. 34, 2017. URL https://gavwood.com/paper.pdf.

[30] Tamer Özsu. Distrubuted database systems. 25, 2002. URL https://www.researchgate.net/publication/228806512_Distributed_Database_Systems.