Semantic Smart Contracts And Their Integration With Semantic Data Licesing

Zahra Jafari
University of Innsbruck

A thesis submitted for the degree of
Master of Informatic

2019

This page is intentionally left blank.

This page is intentionally left blank.

Abstract

Smart contracts as computer code which reside on blockchain, are receiving great attention in new business application because they allow parties to represent contract terms in program code and thus eliminate the need for a trusted third party. The creation process of writing valid, transparent contracts is difficult task. Blockchain as distributed ledger technology is increasingly used as transnational data storage between parties and it gains good popularity among new industries in last few years. Blockchain implemented in different areas of applications such as social, healthcare, logistic and etc. It is also capable to execute smart contract and prevents data tampering by validating transaction through consensus protocol. Research on this topic is still on early stage in science. Since our goal is to analysis the way of integrating semantic web licensing using DALICC library and deploying smart contract, we divide this research into four sections: first we focused on how smart contract build up on blockchain. Second, we demonstrate how blockchain can integrate with semantic web technology. It is focused on some solutions for indexing and executing smart contract on Ethereum blockchain. And third section, It is focused on Data Licenses Clearance Center (DALICC) as a framework helps to detect license conflict and reduce the costs of rights clearance. And in the last section, we present project to present all these topics in the form of DApp.

Contents

List of Figures

0.1 Introduction

Etherem is one of the most popular blockchain platform which capable to generate decentralized application using smart contract. Ethereum is well-known for managing cryptocurrency using smart contract to automate this transaction. It has been the best platform for developer coding smart contract in decentralized paradigm to generate decentralized application DApp.

Ethereum can be used in different scenarios, one strategy is to extract data from its blockchain network, linking this data with the datasets of traditional web and query on them to generate new insight.

Another idea is, using semantic wen to standardize model and integrate data on web. *Tim Berners-Lee* described semantic web as model to extend web and allows to integrate machine with human.

Using semantic web enable access, integration and query on data sources, and create verity insight between different application domains. It uses ontology to allow query from the web using SPARQL and linking datasets.

Ontologies represent ethereum entities lick blocks, transaction message using RDF and Web Ontology Language(OWL). EthOn ontology formalize the concepts of ethereum blockchain on OWL, describing the ethereum objects as classes, data properties in ontology.

This thesis, we focused at two perposes: One ine to integrate semantic licenses from DALICC with smart contract. The other is to represent semantic model of deployed smart contracts.

This thesis present our efforts to build DApp to deploy smart contract in the presence of semantic licenses from DALICC, attaching licenses to contents, then applying semantic web techniques to link extracted date from ethereum transaction to semantic concepts adapted from Ethon ontology.

In the first chapter, we describe distributed ledger technology, blockchain, ethereum and some more details to have better insight of used technology. In the next chapter, we represent some works around how semantic web can be applicable on blockcain. And on the last chapter, we present DApp to show smart contract integration with DALLICC semantic licences library and sementifying this deployment.

DALICC stands for Data Licenses Clearance Center is a framework that supports automated clearance of rights to help detecting license conflict and reduce the cost of high transaction related to manually clearance of licensing contents. DALICC will develop and integrate different functionalities that allows the automated clearance of rights issues.

In this thesis, deals with two things: One is the problem of linking license to contents for this purpose, we use Therefore, this work aims to find an appropriate and generic method to attach semantic licenses to data and content. To achieve this, an approach with a blockchain was chosen to design the entire system in a decentralized and therefore trustworthy manner. Specifically, the Ethereum Platform was chosen, because it is one of the most popular platforms for decentralized applications in the

Smart contract is computer program that expressed the content of agreements and preform transaction on blockchain when specific conditions are met. Smart contract preform verified transaction on blockhain without third party or any supervisors, thus ensuring us to transparent, valid and secure transaction. AS Blockchain as distributed ledger become much popular and widely used in many field, the need of making query of such data becomes more important. As blockain is distributed ledger which store data and transactions, querying them becomes challenges task. This due to the fact that blockchian can allow transaction and payment without needing intermediary. Moreover there is need to integrate blockchin with semantic web service, thus making use of some linked data tools to index blocks and transaction according to Ethereum ontology.

Chapter 1

Smart Contract and Distributed Ledger Technology

1.1 Ledger

Ledger is a book or computer that records transactions associated with a financial system. There are two different ledgers as below:
Centralized Ledger contains all recording 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 of 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 by the consensus algorithm to choose the correct copy of the ledger. Once the consensus has been done, other nodes will synchronize with the latest version of ledger[6].

1.1.1 Distributed Vs. Decentralized

The difference between decentralized and distributed is made by Baran(1964). Decentralized means 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 inter-connected databases that work independently in different locations. *Ozsu and*

Valduriez 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"[**?**]. Based on this definition Blockchain technology covers both definitions, 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.[6].

1.2 Distributed Ledger Technology(DLT)

DLT refers to a database that provides identical copies of shared data among participants which would be updated by consensus of the participants.
DLT is a well-known technology due to the complexity of the consensus mechanism, which makes it easy to implement. DLT is utilized to reduce the costs and increase transparently, traceability, and speed of the process.
This technology is involved many challenges some of them are not been resolved so far. The most common challenges rebuilding to DLT concern scalability, inseparability, and data privacy[17].

1.2.1 How DLT works?

DLT is the result of combining 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[17].

1.3 Blockchain

According to what World Bank Group in their book 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[3].
Luke[8], refers 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[8]. 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 as bellow:

- Proof of Work (PoW): It is a mechanism that ensures consensus is done without any central control. With POW miners compete to complete their transaction first into blockchain and get rewards(e.g: Bitcoin, Ether).
  Miners(actors who participate in cryptocurrency transactions) connected 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[10].

- Proof of Stack (PoS): It is an alternative to proof-of-work that fewer CPU computations for mining. In proof of stack, 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[5].

- Proof of Authority: It confirms accounts and allows them to add a transaction in blocks. As this approach is much more centralized and transaction speed is faster, is prone to be attacked more than the other methods [8].

- Proof of Stake: Blockchain tries to solve the problem called 'Byzantine Generals' which refers to some members on the network who send incoherent information related to the transaction to others. Since there is no authority on blockchain to correct them, leads to the unreliability of blockchain. The Practical Byzantine Fault Tolerance (PBFT) algorithm tries to achieve some algorithm to solve this issue in a way that uses the concept of primary and secondary "duplicates". Secondary copies automatically evaluate the decisions made by primitive and can collectively change to a primitive, if Primary is compromised [8].

Blockchain is associated with cryptocurrencies like Ethereum, bitcoin, lite coin, etc. Gupta (2017) identified five core attributes that blockchain builds trust through them:

- distributed ledger: the data is not controlled by any single authority. the data is shared, and updated across the network and the new changes will be 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 another 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 really who they claim.

- consensus: all nodes on the network should agree to validate transition and blockchain perform this process by consensus algorithm [4].

1.3.1 Type of Blockchian

According to Athital[**?**] blockchain is used to transfer and exchange information through the secure network. Primarily, there were two types of blockchain technology public and private networks. Some analyses on blockchain technology can also be called blockchain as consortium blockchain technology and hybrid blockchain technology.
That should be noted that all kinds of blockchains consist of nodes and works on *P2P* network. Athital [**?**] classified blockchain into three types as bellow: public blockchain, private blockchain, and consortium blockchain. Besides this, there is another type of blockchain, known as the hybrid blockchain.

- **Public Blockchain** This system allows anyone to join to network and create consensuses such as Ethereum. In a public blockchain, any miner (participant) can create consensus mechanisms such as proof of work, and proof of stack to validate the transaction with a low rate of validity[14].

- **Private Blockchain**: In this system, the only restricted participant has the right to validate the transaction. Therefore, it provides better privacy, improves scalability, and mitigates security issues. This blockchain does not have mining computation to reach the consensus because all participants are known in this network[14].

- **Consortium Blockchain**: This is semi-decentralized blockchain. This type of blockchain is used to do activities for a single organization like an organization, bank, etc. The difference between private blockchain with this type is that Consortium Blockchain is controlled by a group rather than a single authority [13].

- **Hybrid Blockchain**: This type is a combination of public and private blockchains. Thus, it makes the benefit of privacy in private blockchain combined with the security and transparency of the public blockchain. In this type of blockchain, the user can control who gets 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 and the rest could still maintain confidential in a private network[13].

1.4 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 participant publishes the transaction on the network and is then divided into a node (called the miner) and added to the blockchain using a consensus mechanism. The state of the system refers to the state of an account which can be an external account related to the user of the system(that contains information about balance) or a contract account that obtain a contract code or constant storage of that account. 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. calling the external account just transfers the Ether but calls the contract account to execute the code of that contract and may perform a transaction or change the storage of that account[**?**].

1.5 How Ethereum works?

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

1.5.1 Blockchain

The blockchain contains some information that we have used in our project, that's why we focus on them more as below:

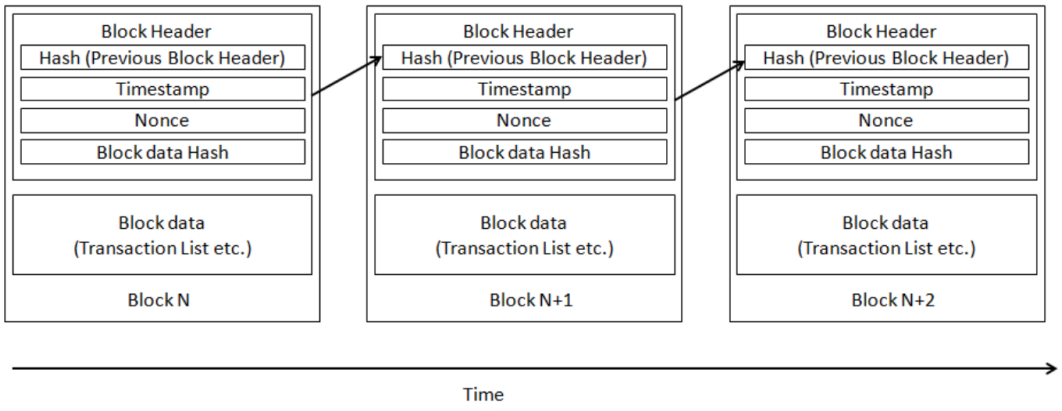


Figure 1.1: Block contents

- **Block** The data stored in the block contains different functions which include transaction hashes and some other additional information for blockchain technology. *Gavin Wood* described some relevant information as bellow and we used this information in our project:

  - *parentHash*: The hash of the parent blockâĂŹs header.
  - *sateRoot*: The hash of the root node of the state, after transactions are executed.
  - *transactionRoot*: The hash of the root node of data populated with a transaction in the transactions list inside the block.
  - *receiptRoot*: 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*
  - *extraData* is a byte array containing relevant information in the block.
  - *nonce* is several computations that have been done in the block.

- **mining** It is a process of computation on the blockchain to verify and add a block. 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 valid 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. Uncle block is rewarded with $\frac{7}{8}$ of full block value and hash will be added to a valid block. A max of two uncle blocks can add to a valid block and the miner of the valid block also receive $\frac{1}{32}$ extra ether for each uncle block[**?**].

- **mining pool** Mining can be done alone or in the mining pool. A mining pool is a better way to solve a block and get rewards as compared to mining alone. Miners in the pool mine together and rewards will be split to all members in the pool[**?**].

1.5.2 Ether

Is the form of payment and fuel for Ethereum. The base for mining(find the solution and add block) successfully mining block is five ether. If the miner finds a solution but not fast, it becomes less ether like 4.375 ether, and will be 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 can not be added to the blockchain and the miner just receives 2-3 Ether[**?**].

1.5.3 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.[**?**]. An account encompasses four fields:

- *nonce* is the number of transactions sent from this address [**?**].
- *balance* is the number of Wei owned by this address[**?**].
- *storageRoot* is the hash of the root node of the Merkle Patrica tree which encodes the content of an account. It should be also not that the Merkle tree is used for data representation in block header[**?**].
- *codeHash* The hash associated with this account 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.

Hash function The process of SHA-3 standardization by NIST was completed in Auguts 2015. This standard specify SHA3 Secure hash algorithm-3 family of function on binary data. Each hash function is based on *Kacak* algorithm that is known as NIST as winner of SHA-3 cryptographic hash algorithm competition. The SHA3 family includes four functions with different lengths of 224, 256, 384, and 512 bits. In hash function, input called as *message* and output called as *has value* which the length of message can vary but, the length of hash value is always fixed [2].

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

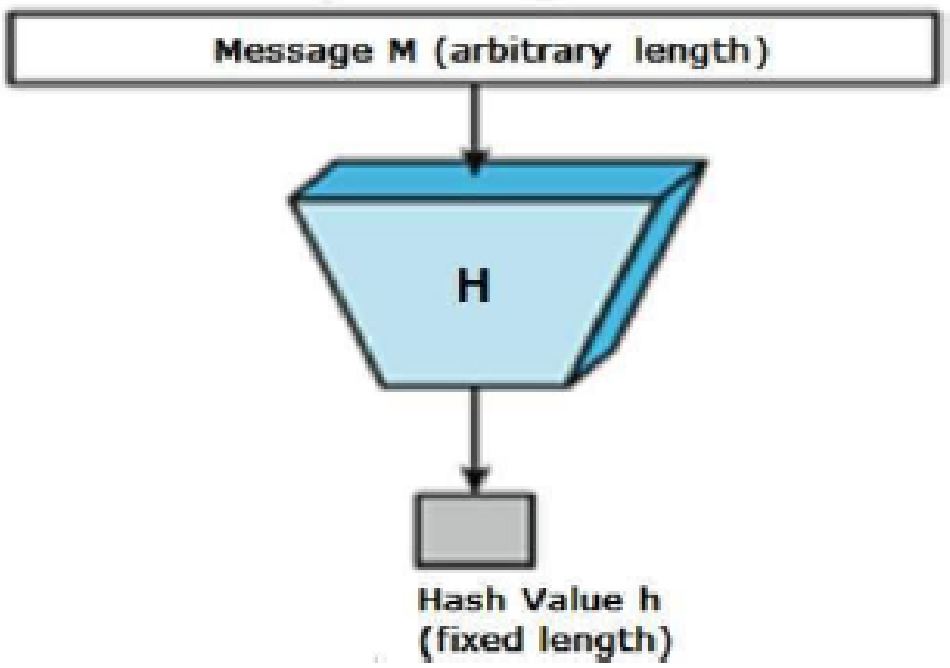


Figure 1.2: Image of hash function

Gas Transaction in the Ethereum platform needs fuel to execute called gas which is used internally and paid in advance to execute a transaction. If the transaction gets run off gas, means the transaction is executed. If the transaction rolled back but consumed gas

will not be returned.
To enable easier calculations, Ether also has some sub-denominations[**?**]:

- Wei - 10^0
- Szabo - 10^12
- Finney - 10^15
- Ether - 10^18
- **Transaction** *Gavin Wood* described a transaction as a cryptography-signed instruction that is executed by an external actor. an external actor can be a human or another contract. 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** As already said blockchain fires transactions when receiving a transaction. When an account sends a transaction means sending a message. The message contains all attributes the same as a transaction, but $gasPrice$. the only difference between message and transaction is that message is fired by contract[**?**].

1.5.4 Contracts

is an account in the Ethereum blockchain having its code and 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 which are programmed to fulfill the user's goals, meaning that the contract is an alive autonomous agent which is executed when it receives a message or transaction, having control over its balance and the key /value store to constant variables. the key and values stored in the contract are long-lasting and get used whenever the contract starts running [**?**].

1.5.5 smart contract

The term smart contract was coined in 1994 by Nick Szabo who released that DLT can be used for smart contracts. According to Nick Szabo *A smart contract is a computerized transaction protocol that executes the terms of a contract*[1]. He visualized an away to write agreement which enforces the conditions between parties involved in transaction automatically and more efficiently. Smart contracts run by each node as part of the block creation process. Block creation means when transactions take place in the block. An important part of a smart contract is that each contract has its address. Since the contract code is con carried to the transaction, a node can create the spacial transaction, assigning an address to the contract, then this transaction is capable to run 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, should send a message to the address of the contract having the method and input data. the contract will run as the part of creation of a new block, then return value or store data in the blockchain. [11].

Solidity is high-level truing complete language with Java script similar syntax. The contract is similar to classes in an object-oriented language which contains the fields as persistent storage of contracts and methods to be invoked by internal and external transactions. For interacting with another contract, you 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 transactions details like: *msg:sender* for accessing the address of an account or *msg:value* to access the amount of *wei* transferred by the transaction. Solidity uses some functions to transfer money to another contract such as *call* and *send*. This function gets used to transfer value and translate as an internal call to a transaction which causes to call contract also execute code or may fail to execute due to insufficient gas [**?**].

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

Chapter 2

Blockchain as the infrastructure of semantic web

2.1 Distributed ledgers and indexing

A distributed ledger based on a blockchain does not have central control. Blockchains are organized into multiple blocks the initial block is created manually and the other blocks are added by some consensus process between nodes. Ethereum smart contract provides the possibility to control automatically what happens with cryptocurrency on the blockchain without involving untrusted external sources. Ethereum smart contracts have an account that can normally store, update, or make a function with the input and output. As smart contracts are time-ordered where data are stored in blocks, requires the data to index. Indexing the smart contract gives us the capability to access the data, search, analyze services on the distributed ledger, and expose them to outside the world for more interactions. There are different levels of indexing smart contracts: Basic level is the fundamental level for the next step. It indexes basic entities such as accounts, blocks related to distributed level, 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 [1].

2.1.1 Why do we use ontology for Blockchain?

Generally, Blockchain is the distributed database that is replicated over all nodes as a cloud computing architecture. These databases are distributed across multiple organizations. Thus, data should have a common interpretation to be understandable for

organizations. Interpretations are applicable via formal specification that enables verification and inference within software and applications executed on the network.
This is where ontology plays the main role to ensure a common interpretation of data in the shared database among different enterprises. blockchain as a modeling form used a different type of ontology:
informal/semi ontology facilitates search and enhances a better understanding of the business process for developing and applying on the blockchain.
Formal ontology helps the formal specification to automate inference and verify the operation of the blockchain. In other words, blockchain modeling based on formal ontology can help the development of smart contracts to execute on the blockchain.
Also, we can use ontology to capture data within blockchain: On one hand, It facilitates a better understanding of blockchain concepts for humans. On the other hand, enables interlinking with other link data to convey deductions and formal reasoning[**?**].
Vocabulary used within ontology increases the transparency of transactions in a way that by describing the transaction in the context of linked data and facilitates the graphical representation of the location of such transaction. Thus, it increases also the capability of analysis by users[**?**].

2.1.2 Linked Data

Linked data is not a single format or standard but, as *Tim Berbers-Lee* referred to in some informal documents is the *expectation of behavior*. When information is presented as Linked Data, other related information can be easily retrieved and new information also can be easily linked to it. *Berners-Lee* described four rules for linked data:
- URIs(Uniform Resource identifier) as names.
- HTTP to search for names.
- (SPARQL, RDF) when a user searches for something, provides related information.
- Link to other URLs to provide more information[Ugarte].

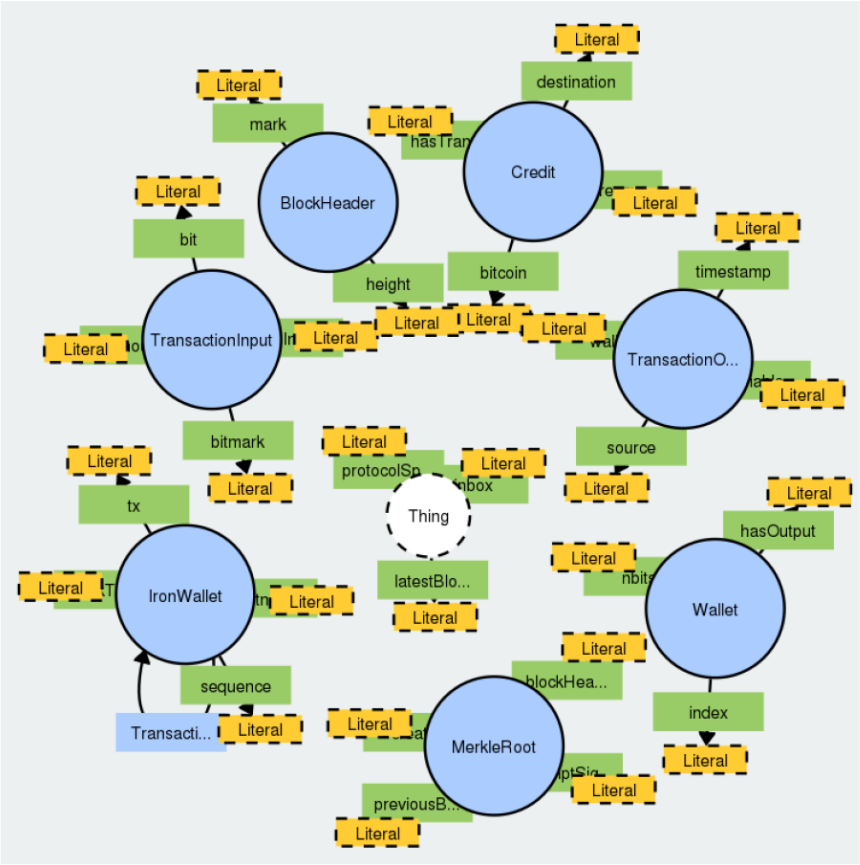


Figure 2.1: Illustration of Ontology diagram[7]

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 is called a triple. i) Subjects that RDF expressions describe them. ii) predict is specific properties, attributes, or relation to describe a resource. iii) The object is the name of a property or value. We can build a graph based on these three objects[Ugarte].

2.1.4 SPARQL

According to Wikipedia definition: It is a semantic query language for a dataset that makes us able to retrieve and modify data stored in RDF format known as triple. SPARQL can query the one, two, or all elements of triples.

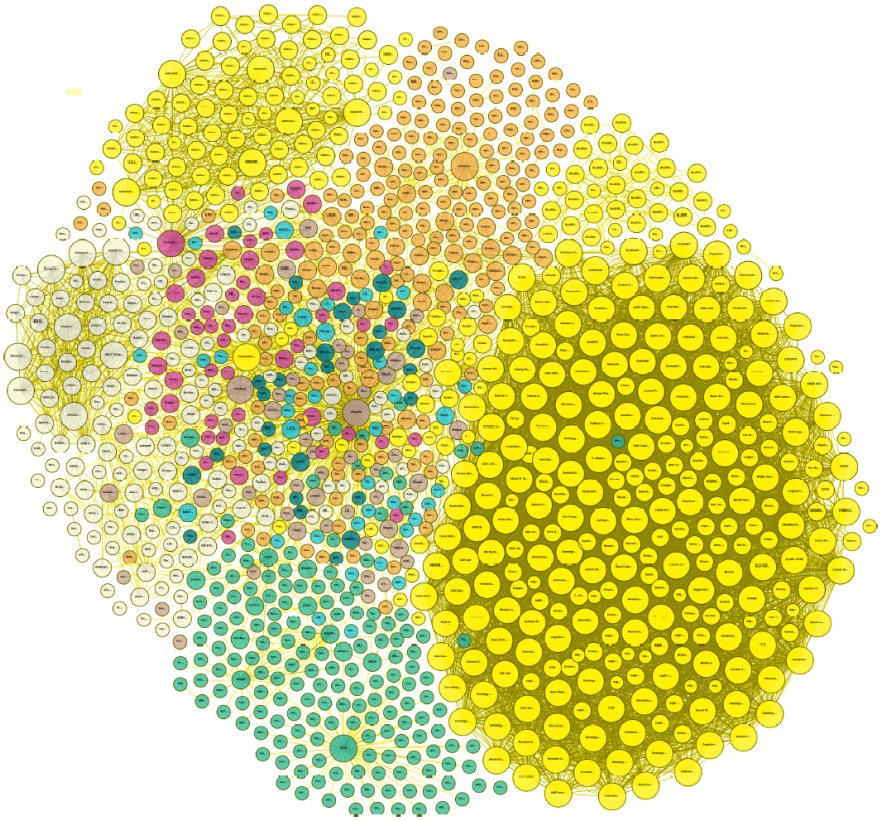


Figure 2.2: Linked data Diagram[Ugarte]

2.1.5 Ontology Web Language

Ontology Web Language is made 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 like negation, intersection, and so on[Ugarte].

2.1.6 Web 3.0

[Ugarte] Evolution and interaction of people on the Internet is classified based on three technologies:
Web 1.0 is 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 people* has the capability of providing space for users to collaborate with content creation or modification.
Web 3.0 is strengthened by the semantic web, where people have access to linked information on the web. But newly, with the arrival of distributed technology like blockchain, and Ethereum which get used by Web 3.0, this is a new focus on this.

2.2 Vocabularies

2.2.1 Vocabulary in Distributed Ledger

Generating linked data requires to use a standard ontology or vocabulary to explain the blockchain concepts. Interfaces between distributed ledgers and the Semantic Web are still in an early stage. Some systems define such vocabulary such as FlexLedger, EthOn, BLONDiE[1].

FlexLedger: describes HTTP interfaces to blockchains, with a standard vocabulary and responses of these interfaces. FLexLedger is a protocol for decentralized ledger and graph data model which represents ledger creation, querying, and data model using JSON-LD. However, FlexLedger does not have explicit vocabulary about ontology nor has concrete ontology for itself.
It is striking to say that the FlexLedger is not suitable to implement in some graph models like graph chain because in FlexLedger meta and the content data are stored together in the same graph whereas the GraphChain blocksâĂŹ content is stored outside the blockchain a separate graph [9].

EthOn is an OWl ontology that describes blockchain classes such as *"blocks, accounts, message", "state"* and relations such *"has parent block"*[18].

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 defined some terms such as 'account', 'block', and 'transaction', and some attributes such as 'transaction payload' or 'miner address'. BLONDiE defines some other concepts for different Blockchain such as 'BitcoinBlockHeader' and 'EthereumBlockHeader' as subclasses of 'BlockHeader'. At the

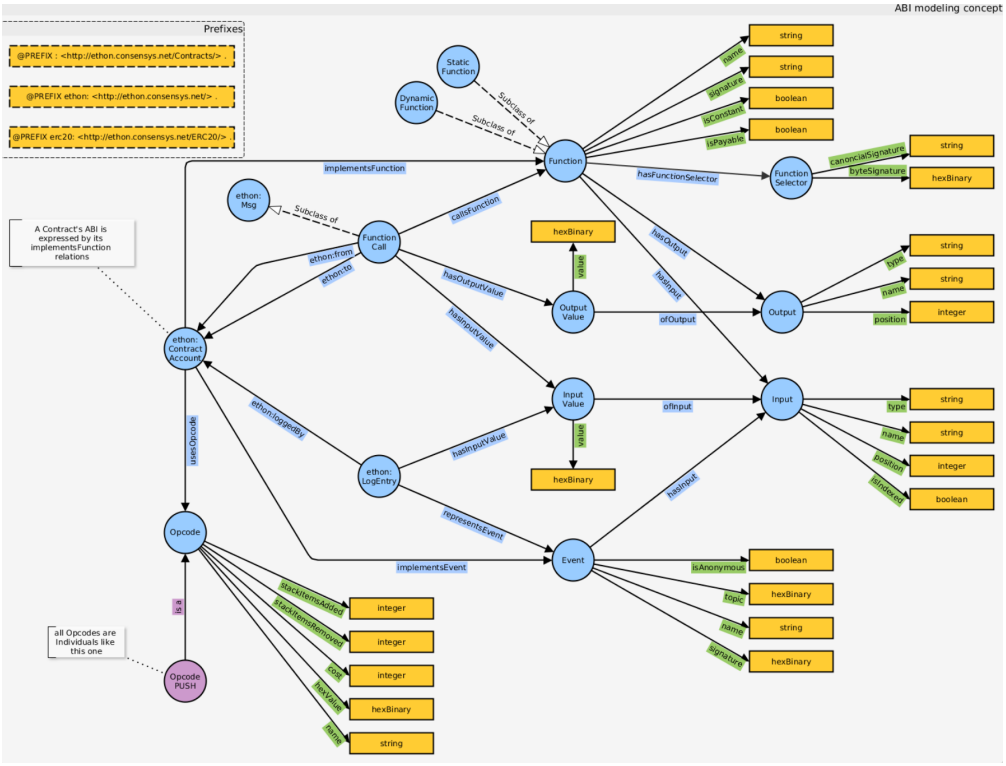


Figure 2.3: EthOn Contract Model(blue arrow is object properties, green arrow is data properties, purple circle is instance and blue one is class)[18]

moment, BLONDiE supports two cryptocurrencies like bitcoin and Ethereum where all links and relationships between objects and attributes represent in RDF(Resource Description Framework)[1].

2.2.2 Vocabulary in Smart Contracts

As mentioned earlier, EthOn and BLONDiE are both similar concepts that can be used for smart contracts. As the smart contract is the executable software, the semantic and vocabulary apply to other software too.

There are many works on semantic annotation of the web and HTTP APIs which may enable us to annotate smart contracts as well. However, the contracts are not Web API and implementation may differ but the main concept does not differ. In other words, the vocabularies used to annotate web services, are used to annotate smart contracts

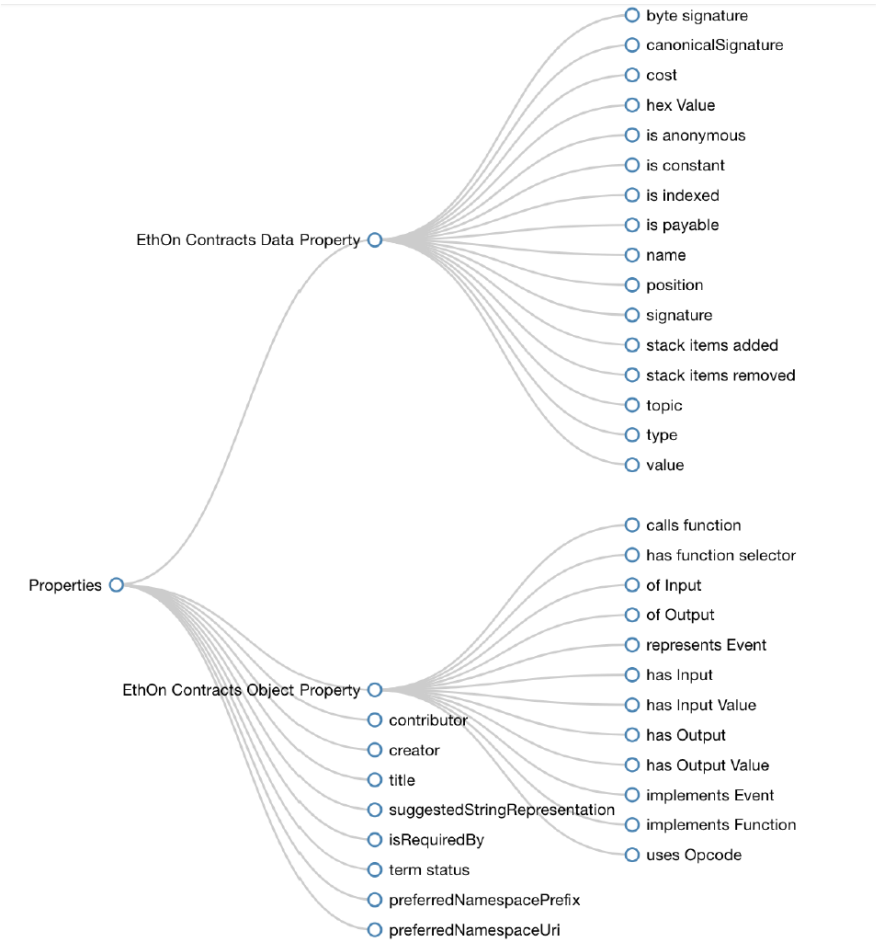


Figure 2.4: EthOn classes[18]

too. It seems that the combination of a distributed ledger with the smart contract and web service due to profitability becomes common[1].

2.3 semantify Blockchain

2.3.1 Semantic Blockchain

With increasing the usage of blockchain technology recently, the need for semantic reasoning on the distributed ledger is on the crease 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. It makes a new dataset so trustworthy. Using semantic web technology on the blockchain is a novel idea and the way how to apply this technology

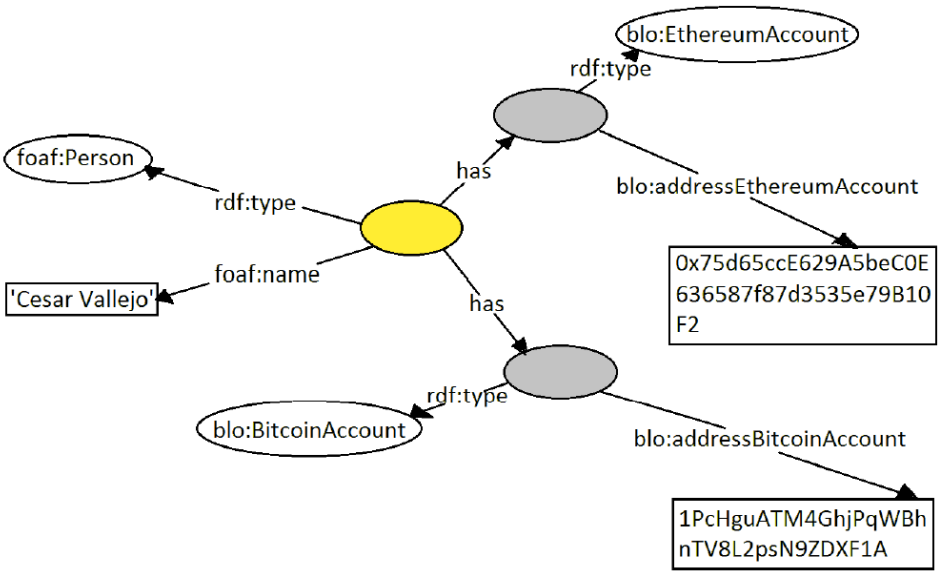


Figure 2.5: BLONDiE usage example[Ugarte]

in blockchain and smart contracts is also a controversial issue.

There are some **definition of semantic blockchain:**

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

- Semantic blockchain is the applying semantic web standard on the blockchain that these standards are based on RDF.

2.3.2 Semantification process

Semantic blockchain or semantic distributed ledger affects the industrial world and subsequently, the result leads to start development new application and frameworks to combine two worlds: these are some ways to sanctify blockchain:

-Mapping the blockchain data to RDF making usage of vocabulary, ontology, and so on.

- Storage of data in a blockchain is expensive, The only way is to store the hashing point to the data set in the blockchain and then share RDF on the blockchain.

- Create semantic blockchain that internal data exchange protocol in RDF format[Ugarte].

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 ontology. To make the query more efficient, BLONDiE used some ways such as

Firstly, records relating to both block 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 like blocks or smart contracts.

Blockchain stores just a binary form of each contract with metadata. To interact with the contract Application Binary(ABI) Interface specification is required. This specification is in the form of JSON and is created when a smart contract is compiled and stored in the blockchain. The ABI determines all functions of contracts and descriptions about input, and output parameter for each contract[1].

```
{ "badge_abi": [
    {
      "constant": false,
      "inputs": [ { "name": "imageurl", "type": "string" } ],
      "name": "changeImageURL",
      "outputs": [],
      "payable": false,
      "type": "function"
    },
    {
      "constant": false,
      "inputs": [ { "name": "tag", "type": "string" } ],
      "name": "removeTag",
      "outputs": [ { "name": "success", "type": "bool" } ],
      "payable": false,
      "type": "function"
    },
    ...
```

Figure 2.6: Smart contract of ABI

Chapter 3

DALICC

3.1 DALICC

According to *Tassilo*, DALICC stands for Data Licenses Clearance Center. It is a software framework that helps people to legally secure data from a third party and help to detect licensing conflicts and reduce the costs of rights clearance by enabling automated clearance right in the creation of derivative works [12].

3.1.1 DALICC Requirements

The following requirements would be addressed by the DALICC framework:

- *Tackling license heterogeneity:*
  It is possible to combine the various contents which have the same license with different names. But that would be much more difficult to license the resultant 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 the equivalent licenses. It guides the user to possible conflicts of various combined licenses.

- *Tackling REL heterogeneity:*
  Combing licenses are simple if they are expressed through the same RED. 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 exiting 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 if the meaning of the different terms in semantics is aligned. These problems result from indicating the classes, instances, and properties which can not be handled just by mapping.
  This is where DALICC comes to play and helps the user with a workflow that defines the usage context, then gathers additional information to detect conflicts and ambiguous concepts. Based on this information, DALICC makes reasoning over the set of licenses and infers the instruction to the user on how to process with license processing.

- *Legal validity of representations and machine recommendations:*
  According to Anna Fensel[15], 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 legislation according to the country of origin of a jurisdiction (i.e. German Urheberrecht vs. US copyright), the resolution of problems that are derived from multilingual, and the consideration of existing case law in the resolution of licensing conflicts.[15]
  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 and checked the semantic precision derived from different languages and adjusted them[15].

3.1.2 DALICC Software Architecture

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

- **License composer** is a tool that allowed 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 ORDL policies and the letters 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 license composer.
- **License negotiator** as a main component in the DALICC framework that checks the license compatibility and supports license conflicts by detecting equivalence licenses having various names [15].

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 provide by end users themselves. Finally, it involved some kind of data manipulation.
DApp uses the ethereum as 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.

4.1 DApp and Ethereum blockchain

The benefit of using the Ethereum blockchain in DApps are as follows:

- 1- The user can see what is going on before submitting any data.
- 2- Once the user has interaction, no one can temper 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.

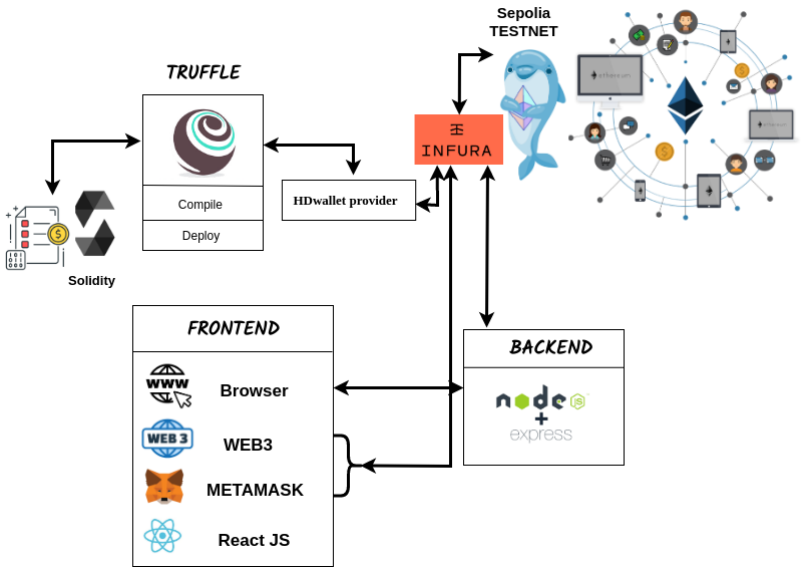


Figure 4.1: DApp infrastructure

4.2.1 Contract and Ontology specification

Here, it is explained some terms related to smart contracts and the ontology of our application. And we needed them as the primary requirement to build up this dApp.

- **Definition 1 (Owner)** is the address of deployed contract or transaction to the blockchain. That is the first person who interacts with the contract. In this case, the owner can change her/his license of the file.

- **Definition 2 (Non-owner)** is the another user that make not action. In this case, can retrieve license information related to specific files.

- **Definition 3 (Licensor)** is the address currently interact with smart contract. In this case, can license his/her 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 ontology, then make a query on produced data from this mapping. This process consists of three following requirements:

  - *Transaction Schema* are actually some attributes resulted from web3.eth functions that EhtOn data properties would be mapped to.
  - *Transaction Triple template* defines for the relationship between concepts(block or transaction) and properties. It is known as 'Subject predicts Object' that Subject is the web3.eth properties, predict known as 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 above parameters as input.

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

4.2.2 Technology Usage

DApp

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.
DApp is similar to centralized apps, as they use frontend and backend. But the backend's app is supported by a centralized server or database. And Backend's DApp runs on a decentralized p2p network and is supported by smart contracts which store on the Ethereum blockchain.

Decentralized application characteristic:

Open Source: all the required are decided on a census 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. *message* is the additional input parameter in text type[2].
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 Managing*: truffle helps to Manage artifacts of smart contracts used in dApp and supports library linking, deploying, and some 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 built contracts [1].

- `React. js` is an open-source JavaScript framework which 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

[1]https://moralis.io/truffle-explained-what-is-the-truffle-suite

create more responsive web application [2].

- `crypto. js` is the JavaScript library that performs data encryption and decryption. It is a collection of standard algorithms including SHA3-256 [3].

- `Web3.js` helps developer to connect to Ethereum blockchain. It is a collection of libraries that allows developers to perform some actions like sending ether, checking data from smart contracts, and creating smart contracts [4].

- `axios` provides the HTTP requests from the browser and handles request/response data [5].

- `HDwallet provider` is a package to help the developer to connect to the network. It is easy to configure the connection network to the Ethereum blockchain through *infura*. This provider is used by Truffle when we deploy the contract. In addition, meta mask providers are also used when we want to interact with the contract in the browser.
  HDwallet provider provides a custom URL: 'http://127.0.0.1:7545'. This will spawn a development blockchain locally on port 7545 [6].

- `Express.js` is node.js framework for authority dApp. It provides HTTP methods (GET, POST) to call functions for particular URL routes [7].
  When we run 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* [8].

[2] https://blog.hubspot.com/website/react-js
[3] https://github.com/jakubzapletal/crypto-js
[4] https://www.datastax.com/guides/what-is-web3.js
[5] https://axios-http.com/docs/intro
[6] https://github.com/trufflesuite/truffle-hdwallet-provider
[7] https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js
[8] https://blog.risingstack.com/fs-module-in-node-js/

Semantic Web Tools

- *EthOn Ontology* is semantic view of ethereum blockchain. It encompasses different classes and relations to cover different concepts of Ethereum like blocks, transactions, and messages to formalize RDF triple. We used classes and properties related to transaction concepts as a template to model our transaction to DALICC [9].

- *SPARQL query* is defined in the section 2.1.4.

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

Ethereum Tools

- *Infura*, According to the internet definition, a web3 backend and infrastructure-as-a-Service (IaaS) provider offers a range of tools to help the developer to build dApp to connect to the Ethereum blockchain.

- *Metamask Wallet* Metamask is a wallet used to interact with the Ethereum blockchain. It allows users to connect network through a browser extension or mobile app. Metamask wallet, including all accounts, each account has its private key [10].

- *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 main net.

- *Testnets* are the test blockchain networks that perform similarly to the main blockchain (mainnet). This allows developers to execute their contracts on test blockchain freely [11].

[9] https://axios-http.com/docs/intro
[10] https://originstamp.com/blog/metamask-what-is-it-and-how-does-it-work/
[11] https://blog.infura.io/post/introducing-infuras-eth-testnet-faucet-get-0-5-eth-daily-to-test-your-dapps

Infura's new testnet faucet is Sepolia ETH which provides the most reliable and high-volume faucets for developers.

- *Web3.eth* the 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. In our case, some functions including: *getBlock*, *getTransaction* and *getTransactionReceipt* are used to provide some more details which are mapped to Ethan ontology objects properties. These retrieved properties would be used later in triple.

- - *knowledge graph* provided to indicate classes and relations. The idea is to use a graph-based data model to clarify survived transaction, their classes, and relations in much more detail.

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 here/his data or retrieve license information. In the next paragraph, we will more focused on authenticated users as the main challenge in decentralized apps and data licensing systems by users.

Authenticated user

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

- *login 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

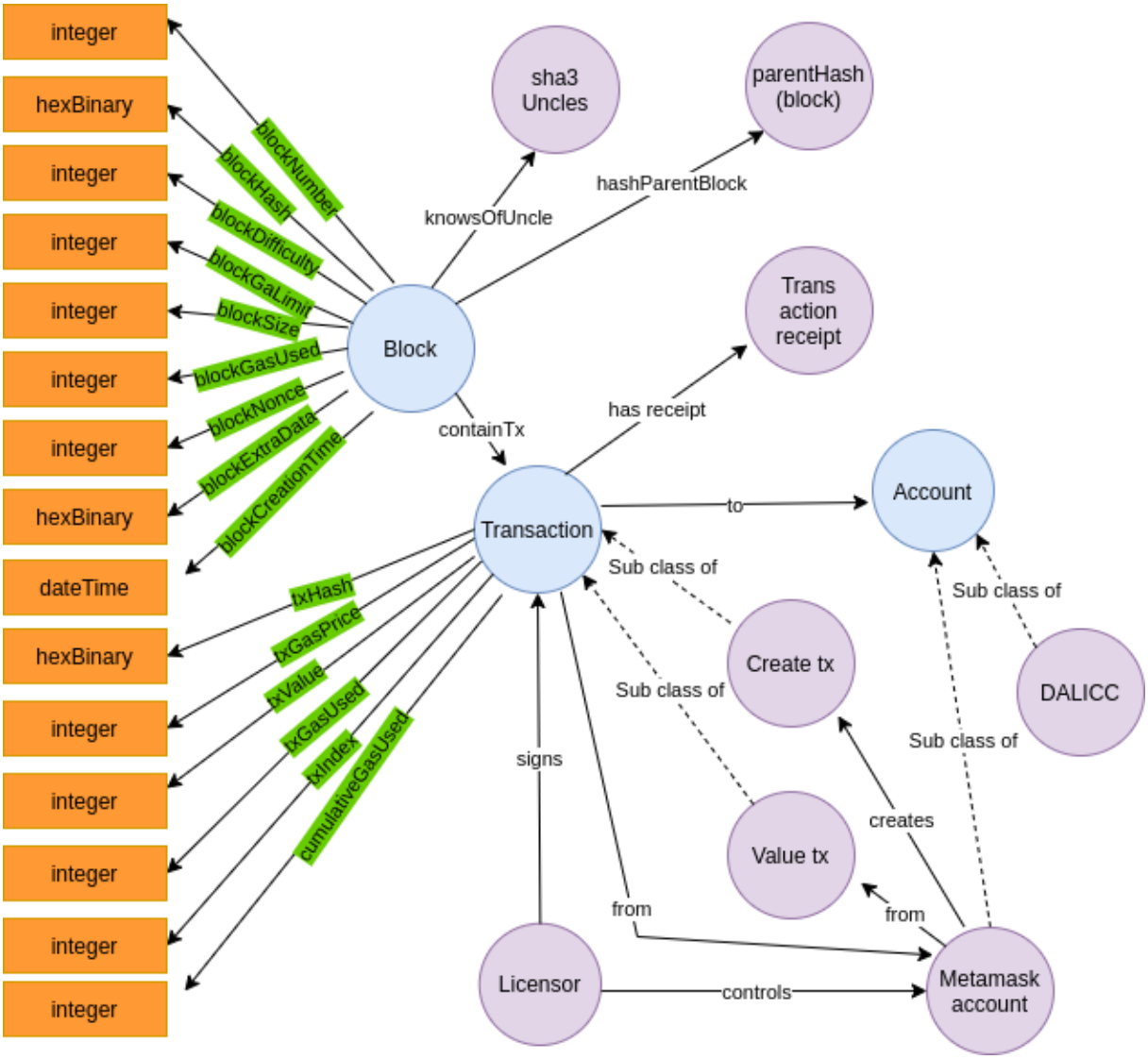


Figure 4.2: Transaction illustration

Metamsk, users can submit transactions or check the stored data on the blockchain.
- *Using smart contract:* The address of the metamask is passed to smart contract as a licensor, then it will use this address as an owner to license his/her file.

Licensing in smart contract as follow:

License information represents three elements: licensor address that is passed by meta-

mask, license of data, and the URI related to this license.

Storage license information: Each smart contract that runs on the Ethereum blockchain would be maintained state 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 contracts 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.

4.3.2 Frontend

Frontend of this project build on React.js, HTML, and CSS. The React js used in this front-end for the user interface.

User roadmap

The licensing Data from the user interface includes three phases:

- The user gets asked to select to license type from which to be loaded from Dalicc Library via Axios and get a request in frontend. Then, after selecting data in the next field, the user should check the file, 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 his/her 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 gets asked to log in to Metamask for the authentication process and this address would be passed to smart contract as owner of this license. The licensing process will be done by receiving the hash value of licensed data.

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

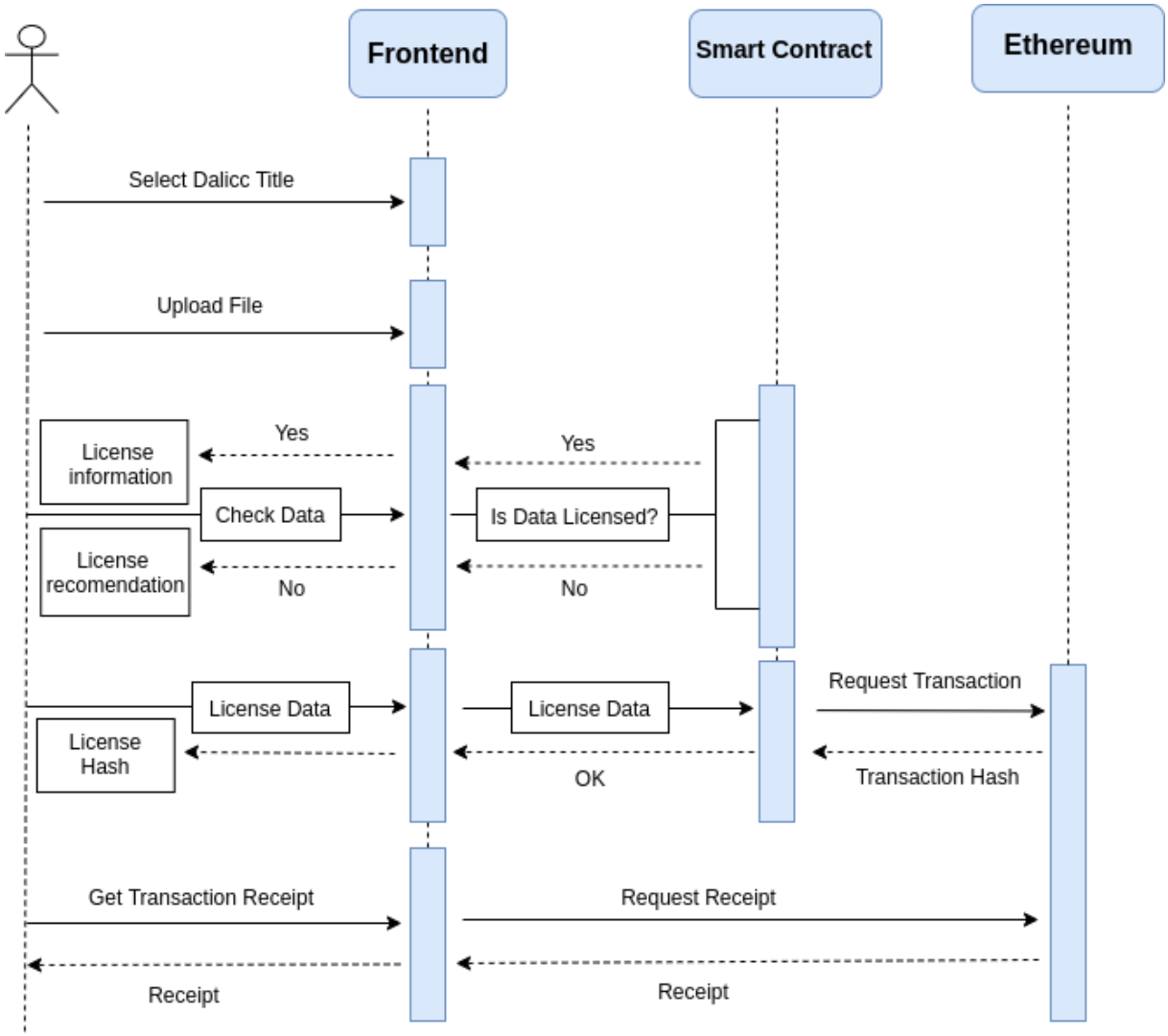


Figure 4.3: DApp Architecture

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 fronted and backend is taken over by Java Script library *web3.js*. For this purpose, web3 provides this connection either with HDWallet-provider to connect to the test network (Sepolia in our case) or Ethereum provider.

Interaction with with DALICC
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 web.eth, then send transaction details in RDF format to the server to store into a file. The produced turtle file will be resulted into a readable format and would be returned to frontend by HTTP GET request from frontend.

4.3.3 DApp Architecture

This application encompasses some components:

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

4.4 Implementation

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

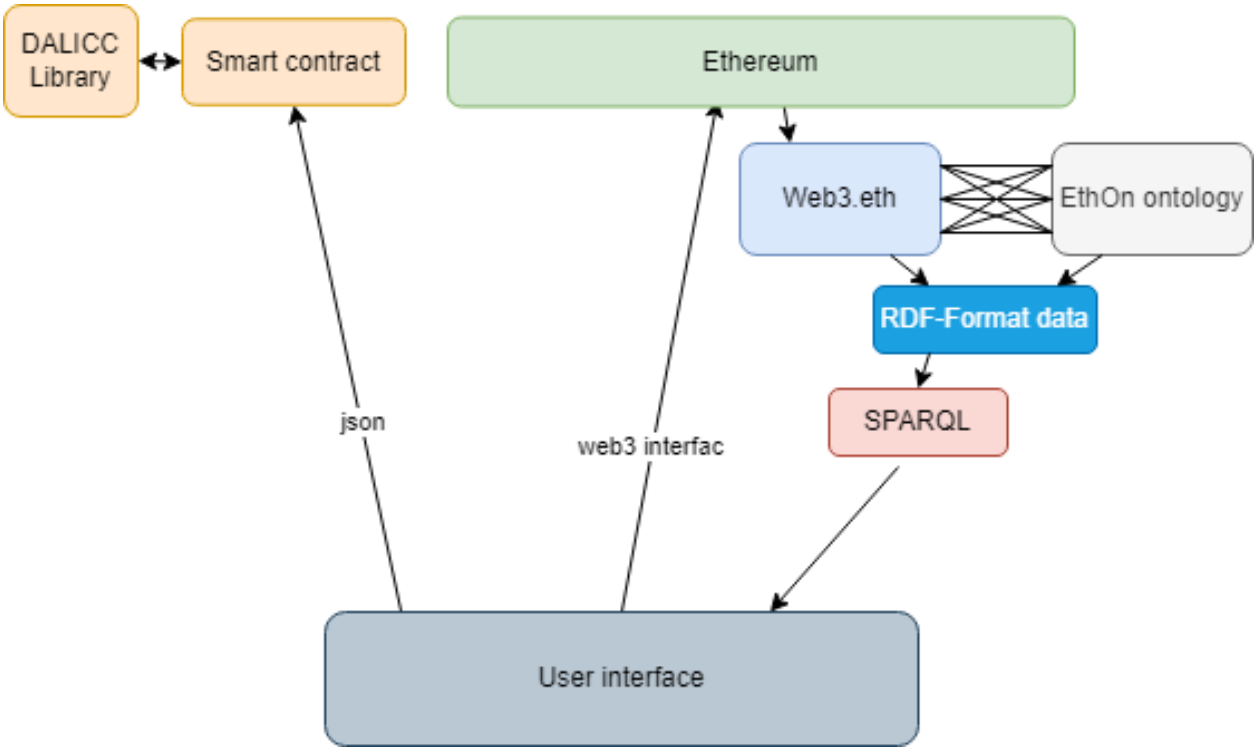


Figure 4.4: DApp Architecture

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 to call the function and the owner as the constructor to be usable in every contract which is called to. This contract provides a function that helps us to restrict access to some function in another contract that would be called later by them.

- *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 licensing data

or retrieves 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*
  In this contract, the hash value and licensor address would be stored here. License contracts represent only one license and associate license contract for this license. The license contract should have been created by the license manager.

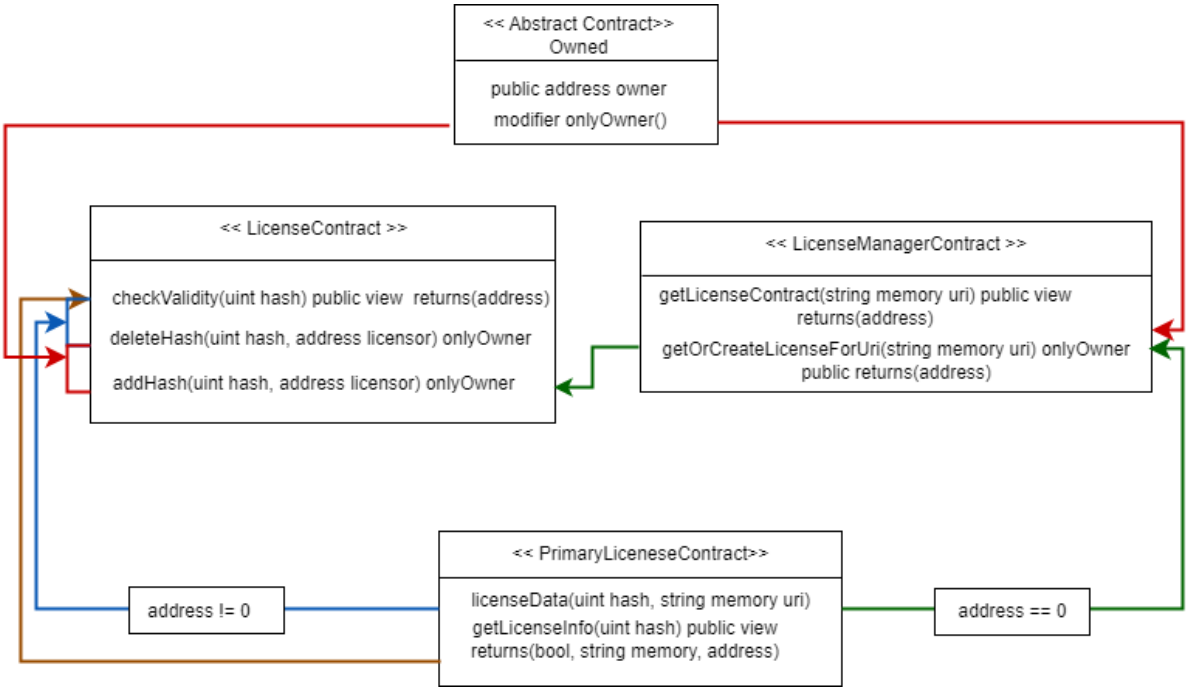


Figure 4.5: Smart contract visualization

How Smart contract works?

- **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* take over the process:

License Data

To license data, the function *licenseData* is used which declares with 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 follow:

This function checks if the target address is the zero account, which means the transaction creates a new contract: if no, function *deleteHash*, otherwise *getOrCreateLicenseForUri* would be called. How does licensing data work?

- *licenseData* function check 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 just 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 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 also will be used later as the owner of this contract.

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

  Definition *addHash*: This function is accessible just 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 PrimaryLicenseContarct.

- **Change License data**
  This is doable just by the licensor in the same way as license data.

- **Retrieve license information**
  To retrieve license information 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 is returned the address otherwise, returns a null address and string.

4.4.2 Forntend Workflow

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

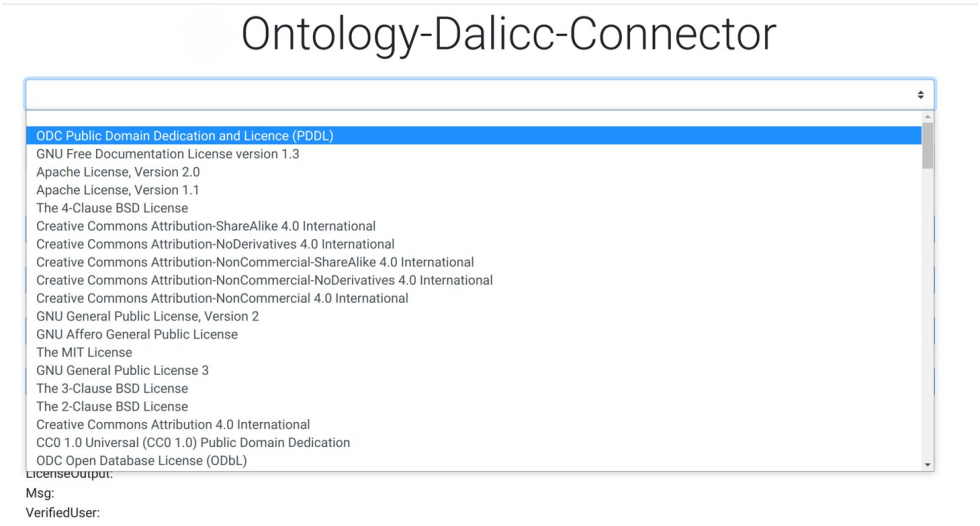


Figure 4.6: Define Type from DALICC library

- Define Content: In this step, the user should choose the file or data that wants to be licensed. Subsequently, the hash `SHA3-256` of the selected file is calculated which is used to retrieve license information later.

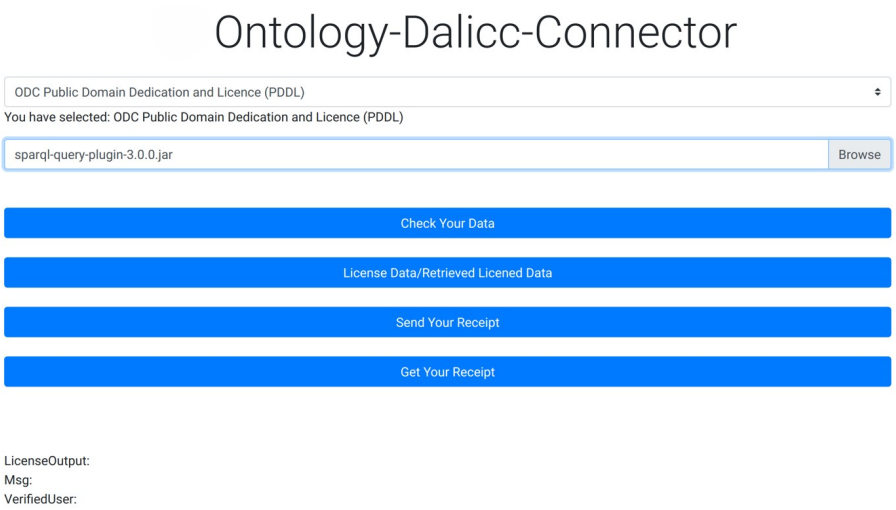


Figure 4.7: Define file from local device

- Check Your Data: In this step, the user should check the selected data: if it is licensed before or not. He/ she receives just a message either confirmation 'License is detected' or rejection 'License is not detected'. He or she should go further to obtain more information.

- 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 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.

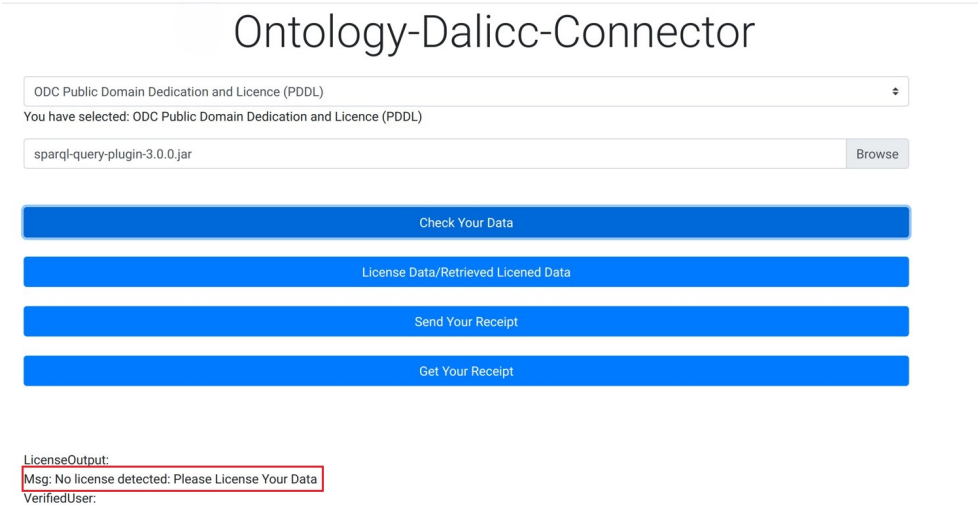


Figure 4.8: Check if file is already licensed?

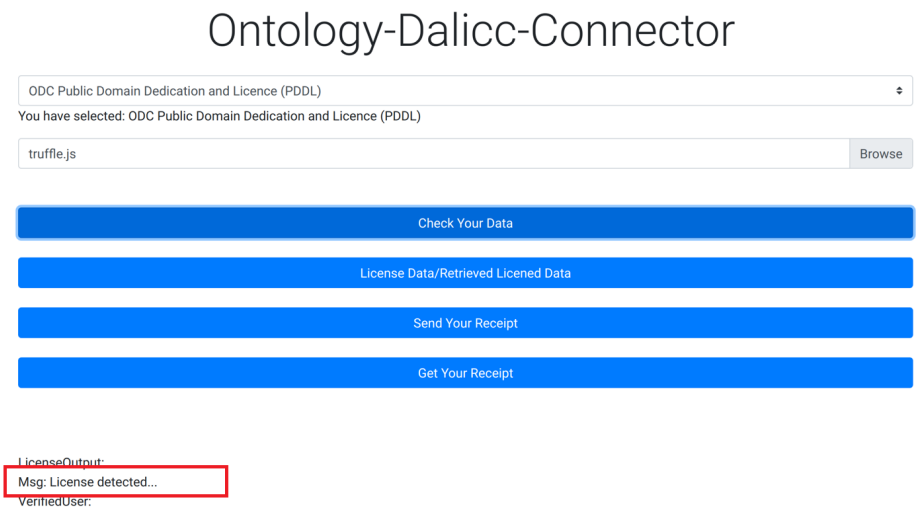


Figure 4.9: message for licensed data

- Start licensing he/her file, if he/she received 'No license detected' from the last step. To license data, the user gets 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 front end, depending on the successful transaction or he/she receives the error message for failed transaction on the front end.

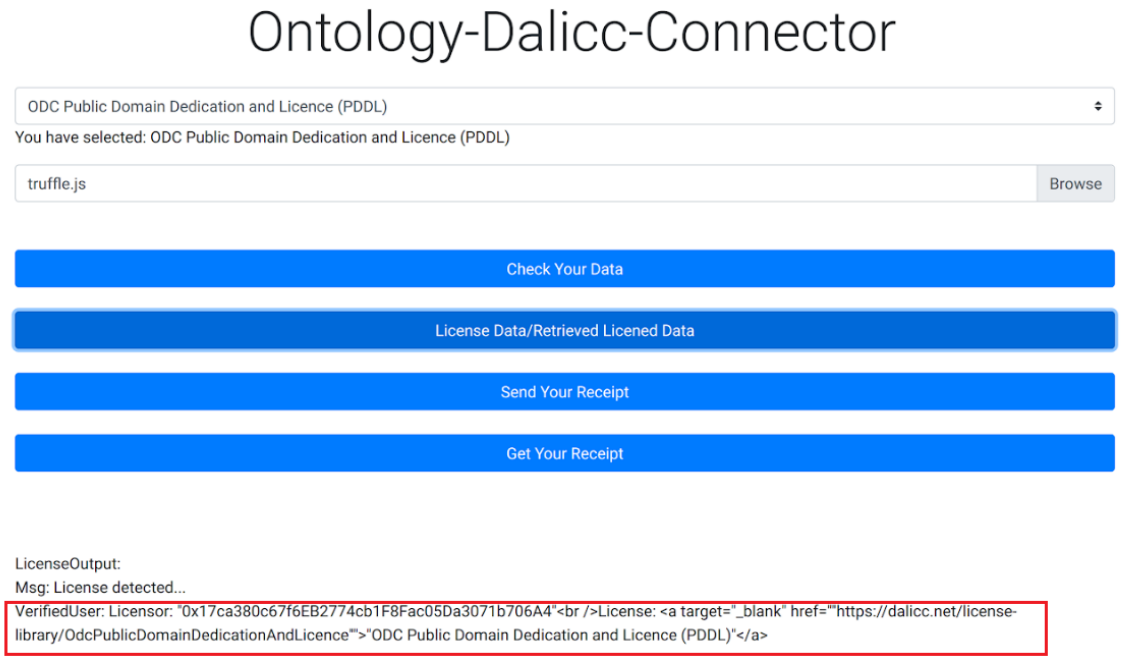


Figure 4.10: license information for already licensed data

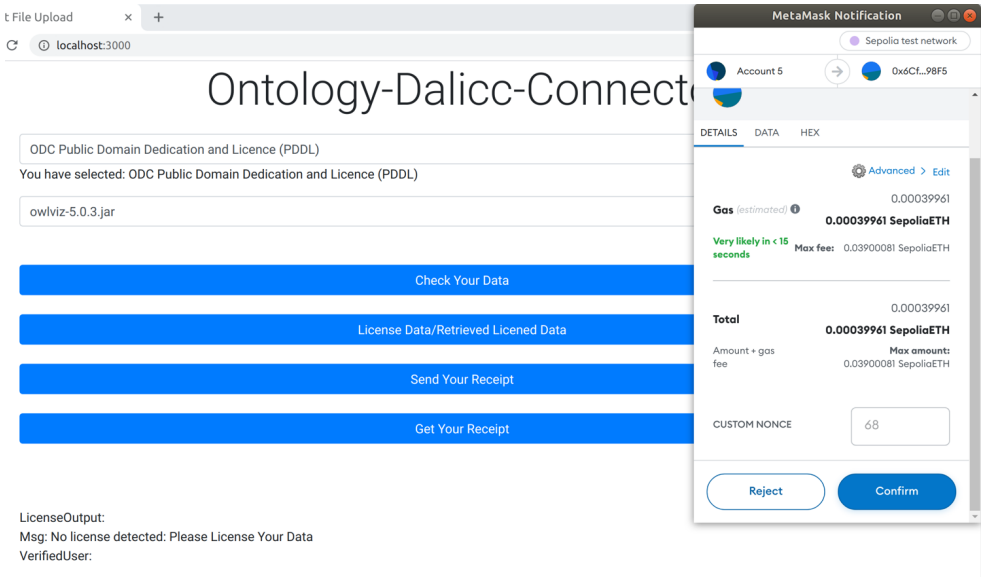


Figure 4.11: licensing data process

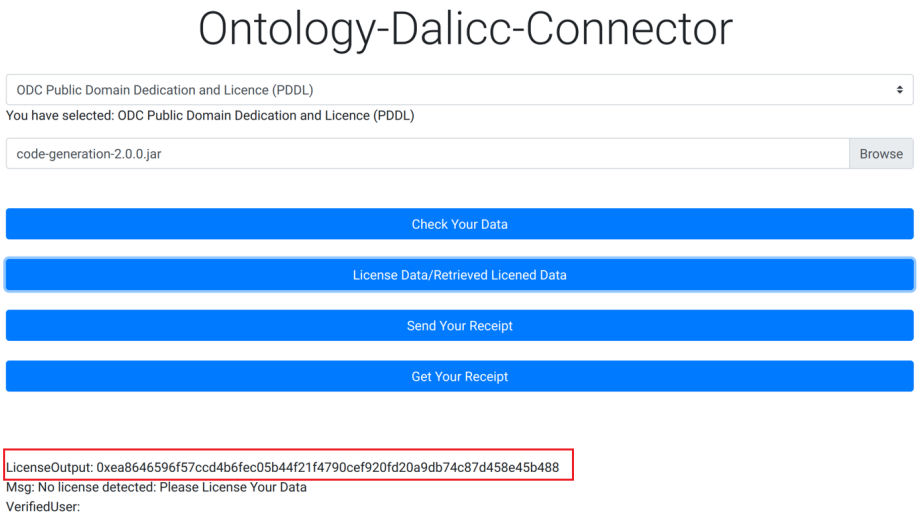


Figure 4.12: license hash after confirming transaction

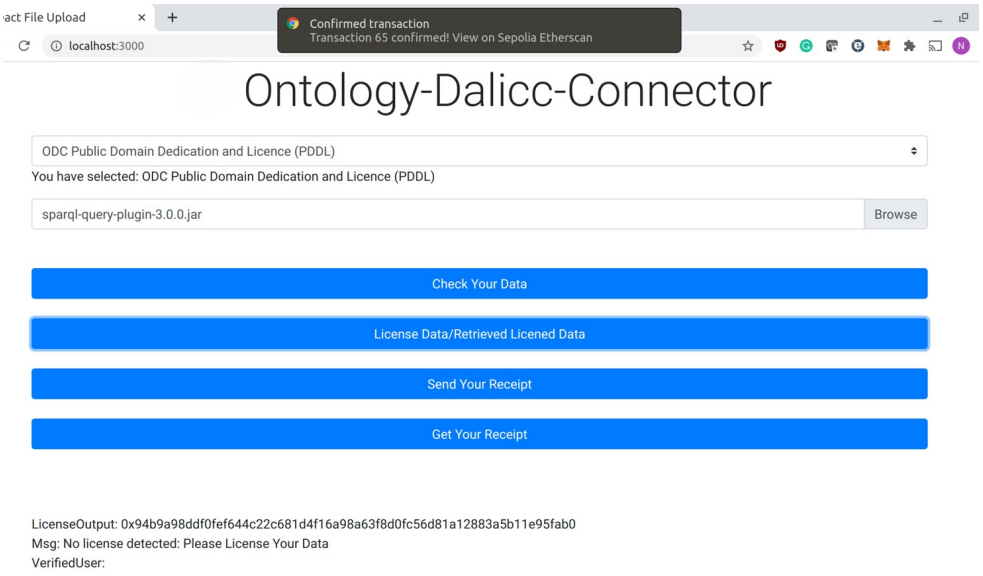


Figure 4.13: transaction confirmation

- 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 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.

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

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...	0xd883010b02846765746888676f312e31392e35856c696e7578	0xd8060d92a316bd366691046a7dba7df97ac1643e9ae0945dd4f9b...
68	0	30000000	1	0x27c62575539cd78f865cb3a507b6700c3d2f000592409 5bcbaee0...	0xd883010b02846765746888676f312e32302e32856c696e7578	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...	0xd883010b02846765746888676f312e32302e32856c696e7578	0x335917d91e720dbe6cdb58d6fc000bab981a64b374804f7858620...
44	0	30000000	3	0x34d80e463124a683a69783eb350109daaad19ed28d57f5f205ebe...	0xd883010b01846765746888676f312e32302e31856c696e7578	0x5a77190c920b04239129283969e44d3f6701939f6e0ee82e4285a...
42	0	30000000	1	0x340952651fb42abf7003a483a4d2e87371091572a9c42ff906e262...	0xd883010b02846765746888676f312e32302e31856c696e7578	0x45a1b437517f5e744b7eefc9542b4565f20ec0e5f1cb68a488b23b...
67	0	30000000	2	0xea8646596f57ccd4b6fec05b44f21f4790cef920fd20a9db74c87d4...	0xd883010b02846765746888676f312e31392e35856c696e7578	0xd8060d92a316bd366691046a7dba7df97ac1643e9ae0945dd4f9b...
68	0	30000000	1	0x27c62575539cd78f865cb3a507b6700c3d2f000592409 5bcbaee0...	0xd883010b02846765746888676f312e32302e32856c696e7578	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…	0x964b90012a3e069a5f1bdb55bfe321f70ac8f13210a1b0b85039bf…	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…	0xb46cc41037c6758476 8ee20672a11cebe87deebdf4f20c75cf4910…	139062	295232519…	0xe5ec022ce34de8e2e88bc1805cc847983743 5b3bbbe0c8b823cbe…
3288397	18087739	125363	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0xe5f4dfef64cea780158354f56b12c88662fd034954ac5b267e3c9ca…	936900	295232519…	0xc96ea05c5ef4f806db084d6ae8007f9611d98ff9277d8784ec9edd5…
3288506	6920779	68752	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a14…	0x964b90012a3e069a5f1bdb55bfe321f70ac8f13210a1b0b85039bf…	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 licenses from the DALICC library with smart contracts and some problems that are still open in the literature special when considering the few works that integrate blockchain with 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 Ethon 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 as Blocks, Accounts, Transactions, Contract Messages, and the relation between the instances of these classes. therefore, licensing system has been created which keeps the integrity of license information and also provided a semantic view of such deployment environment using Ethon ontology concepts.
To do this, the user is guided to start licensing and get its 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;
    }
}
```

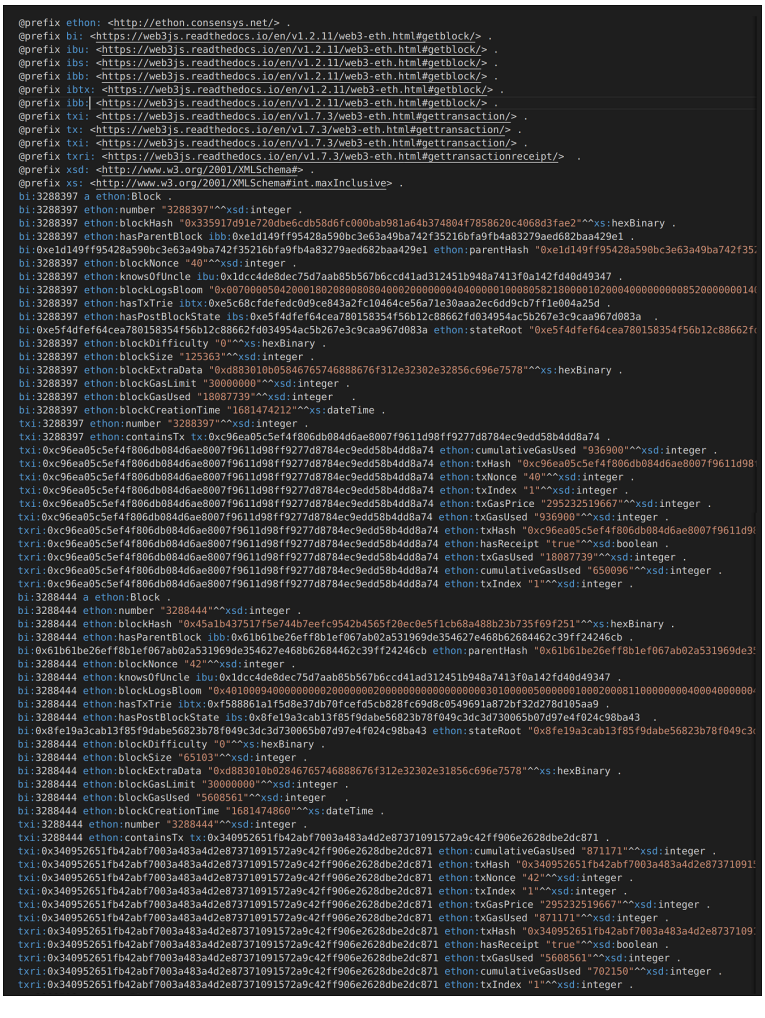
Appendix G

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 "0x0070000504200018020800808040002000000404000001000805821800010200040000000008520000000140
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 "0x4010009400000000020000002000000000000000000030100000500000010002000811000000004000400000
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 "0x340952651fb42abf7003a483a4d2e873710911
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] Allan Third, J. D. (2017). Linked data indexing of distributed ledgers. In *WWW 17 Companion Proceedings of the 26th International Conference on World Wide Web Companion*, volume 8.

[2] Fips (2015). Sha-3 standard: Permutation-based hash and extendable-output functions. 37.

[3] Group, W. B. (2017). *Distributed Ledger Technology(DLT) and Blockchain*. World Bank Group.

[4] Gupta, M. (2018). *Blockchain for dummies*. John Wiley.

[5] Konstantinos Christidis, M. D. (2016). Blockchains and smart contracts for the internet of things.

[6] Markos, K. (2019). Indexes for blockchain data.

[7] Matthew English, S. A. and Domingue, J. (2016). Block Chain Technologies and The Semantic Web: A Framework for Symbiotic Development. 15.

[8] Max Luke, Anna Dimitrova, S. J. L. Z. P. (2018). Blockchain in electricity: a critical review of progress to date. 38.

[9] Mirek Sopek, PrzemysÅĆaw GrÄĚdzki, W. K. D. K. R. T. R. T. (2018). Www '18 companion proceedings of the the web conference 2018. In *GraphChain âĂŞ A Distributed Database with Explicit Semantics and Chained RDF Graphs*, volume 8.

[10] Pablo Lamela Seijas, Simon Thompson, D. M. (2016). Scripting smart contracts for distributed ledger technology. 30.

[11] Payrott, S. (2017). introduction to ethereum and smart contracts. 68.

[12] Pellegrini, T. (2017). Detecting licensing conflicts with dalicc. 5.

[13] P.K. Paul1, P.S. Aithal, R. S. and Ghosh, S. (2021). Blockchain in electricity: a critical review of progress to date. 13.

[14] Sukrit Kalra, Seep Goel, M. D. S. S. (2018). Zeus: Analyzing safety of smart contracts. 15.

[15] Tassilo Pellegrini, Simon Steyskal, O. P. A. F. (2018). Automated rights clearance using semantic web technologies. 16.

[Ugarte] Ugarte, H. A more pragmatic web 3.0: Linked blockchain data.

[17] Ugarte, J. L. R. (2018). Distributed ledger technology. 11.

[18] Umar Rashid, Allan Third, J. D. (2018). Web service for semantic negotiation of smart contracts. 6.