# 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 licesing in this new technology and deploying smart contract, we divide this research into 3 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 use case: supply chain management system and how smart contract facilitates supply chain process and represented different models of ontology in this field, then implemented Etherum ontology concepts to model in semantic blockchain using RDF triples and SPRQL query.*

# Contents

# List of Figures

## 0.1 Introduction

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 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.

In this paper, supply chain management is regarded as use case where blokchain is fit for some reasons. During product life cycle in every step, data can be documented in blockchain. Blockchain technology can contribute to record single asset as it flows through supply chain node, track orders, payment, product and track digital asset. Blockchain can contribute through distributed nature in sharing information about product process, delivery between from supplier to customers. In today's world, supply chain is complicated structure with multiple involved participants with amount of activities. Security and organizational issue cause to improve the need to build blockchain based supply chain management. In spite of some features of supply chain, blockchain also offers some advantages by indexing, registering products, increase transparency and trust of participants. Besides elimination of third party which allow for growing number of participants, it increases innovation by deploying smart contract with low fee transactions, without cost of third party.

The literature provides some supply chain management ontologies for range of activities and industries. Many studies claim the benefit of ontology in supply chain industries which are suggested multiple models to supply chain ontology another application of these ontologies by some Enterprise Modeler program model ontology. This reports different ontologies models in supply chain management fields. Then it works on how semantic web service can be applicable in this field. Existing supply chain ontology has several gaps with respect to domain accuracy, consistency and development approach. But we attempted to semantify blockchain using ontologies and some semantic techniques.Then we purposed proof of concept developed in the concept of supply chain

management and semantic blockchain.

As this is important to gain advantages, we used Ethereum ontology which provide some specifications of phenomenon, adapt semantic principles for developing semantic blockchain and mapped these concepts using RDF triple and model based on Sparql query to produce blockchain models based on semantic web ontologies.

# Chapter 1

# Smart Contract and Distributed Ledger Technology

## 1.1 Ledger

Ledger is a principal book or computer file for recording transactions measured in terms of unit account with specific balance for each account[22].

### 1.1.1 Distributed Vs. Undistributed

the difference between decentralized and distributed is made by Baran(1964). decentralized means there is no single point to make a decision. It contains central hierarchy nodes. Each node makes up a decision for own self and the system gathers all responses as resulting behavior. In a distributed system, the process spread across all nodes and there are no central nodes. The main difference is that a decentralized database is the collection of independent databases, while distributed is the single database that is spread multiple inter-connected computers 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"[28]. Based on this definition Blockchain technology adheres to both definitions, as it appears as a single system to its users and performs a task in a network. Thus, Blockchain is a form of a distributed computing system.[22].

## 1.2 Distributed Ledger Technology(DLT)

DLT is the method of keeping a distributed ledger on networks of a computer. DLT uses a consensus technique for adding a new node(participant) over the network. Similarly, This is the digital record that shared across participant and the records held by each node or participant. The term 'blockchain' is the most well-known type of DLT and are used by many best-known instances of DLT.
A distributed ledger can be either permissionless or with permission, regarding be private or public. Most of the distributed ledger is permissionless and public means anyone can easily join to network and see all entries. But in financial fields, distributed ledgers are restricted to members of each participant and they are 'private' networks. Furthermore, in financial fields exist also commercial sensitivity about the privacy of data related to each participant. In another word, no one wants that data to be seen by the other participant. therefore, participants can see their data and transaction on the network.
Recently, blockchain as a type of distributed ledger become the most popular and widely used in diverse fields. one of the outstanding usages of blockchain is Ethereum which extends initial bitcoin blockchain with features such as smart contracts that enable to distribute of data on the blockchain securely. Also, there is an increasing demand to integrate data stored in distributed ledger with other external sources. And also integrate smart contracts with the service available on the web. This is where linked data comes to play to provide sufficient access to data and smart contracts also stored on Ehereum blockchain via semantic web technology stack[3].

## 1.3 Blockchain

Blockchains are the distributed records for digital events and they are structured as a chain of linked data stored in individual database or computer over network[3]. Blockchain is organized into blocks. Each block is identified by cryptography hash that refers to a hash of the previous block. this creates links between blocks. Thus, it creates the chain of blocks wherein Blocks have held a copy of the blockchain structure. The initial block created manually, is known as *genesis block* and the other node will add to the blockchain after a process of consensus between nodes. The distributed consensus method allows adding a new block or item into the blockchain that is verified as legitimate. This process would be done by some computational work called Proof Of Work

(POW) or mining. All blocks in blockchain hold a small amount of data which need to be secure before distributing over all participating computer over the network and are visible by having the public key for all participant but not modifiable. These data get timestamp to provide the time of adding that block.

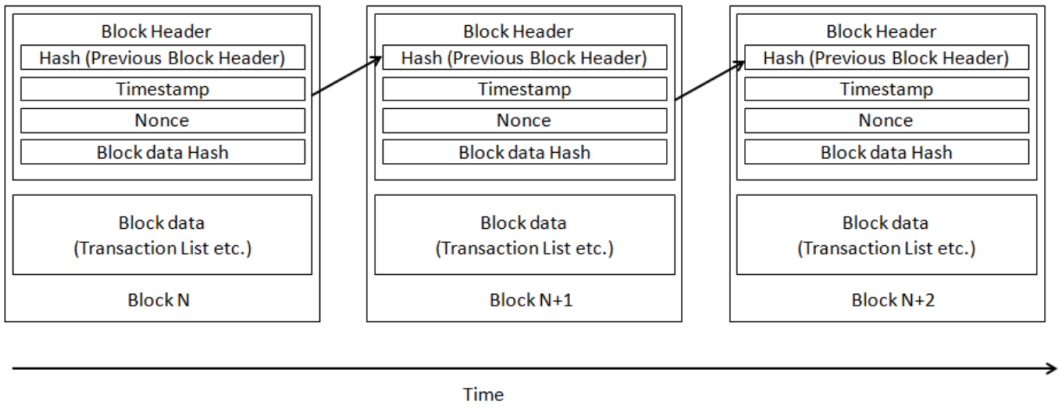


Figure 1.1: Generic chain of blocks

Blockchain has four main concepts:

*P2P network:* nodes can interact with each other using the private and public key. Private keys are transaction signature and public keys are the address of transactions that can be reachable on the network.

*Open distributed ledger*: It is like a book containing all transactions that each node has a copy of data and there is no central entity and anyone can see assets and how much assets has each node. Thus, they can decide about the validity of transactions.

*Synchronization*: As all node has one copy of the ledger. Therefore, synchronization should be done throughout networks whenever a new item or one new transaction added to the network and finally consensus will need to validate these transactions.

*Mining:* In the distributed ledger, all nodes can not receive transaction simultaneously. To add a new item in the blockchain, the consensus of all nodes are needed and it prevents every node to add a transaction into the blockchain. Miners(buyer) are who attempt to validate the transaction to add to the blockchain.

### 1.3.1 Advantage Of Blockchain

There are several distributed system based on consensus, but the one outstanding feature that makes blockchain more prominent then the others is that:
Only single record stores in each participant computer. This provides transparency to the transaction.
- Storing whole records overall networks of participating computers mitigate the probability of losing infrastructure. - One new item is confirmed and added to the blockchain, can not be altered. - Events are stored and accessible to any participant but required public key to be reachable[30].
- Blockchain is the sole technology that fulfills such properties:*(a)trustless*: There is no need to verify involved identities in a transaction. *(ii) Permissionless*: there is neither permission nor controlling for participants in the network. *(iii)censorship resistant*: Anyone can trade on blockchain and just cryptography algorithm governs the operation that entities(participants) can trust it. this feature and above reason make the blockchain as powerful and most secure technology in commercial events over network[2].

### 1.3.2 Limitation of Blockchain

There are some limitations and need to cope with before blockchain can be more applicable: the proof of work that generates blocks are wasteful. Although, it helps us to remain integrity in blockchain and prevent form attacks, as blockchain gets longer this problem deteriorates.
Also, when blockchain grows, processing speed affects adversely. Because it needs more time to verify transactions. However, blockchain relies on a consensus of the majority to ensure security in blockchain. there is a probability of being violated by this security. Requiring the consensus cause some malicious attackers can tamper the blockchain by achieving %51 of consensus and introduce their version into blockchain and validate it. To overcome these limitations, proof of stack(PoS) used over proof of work. In POS, miners are not compulsory to solve a computational puzzle. Instead, they would be selected based on their wealth or stack in the crypto token. This will decrease the probability of a %51 attack and wasting the resources.

### 1.3.3 Type of Blockchian

**Permissionless/ public Blockchain** This blockchain allows anyone to join to network and create consensuses such as Bitcoin and Ethereum. In a permissionless blockchain, any miner can create consensus mechanisms such as proof of work, proof of stack to validate the transaction. but as this mechanism is decentralized, it has a low rate of validity function[31].

**Permissioned/ Private Blockchain**: This blockchain, the only restricted participant has the right to validate the transaction. Therefore, it provides better privacy and scaleability. Unlike permissionless blockchain, this blockchain does not have mining computation to reach the consensus because all participants are known in this network[31].
**Wallet** contains the ether which will be used to execute transactions[36].

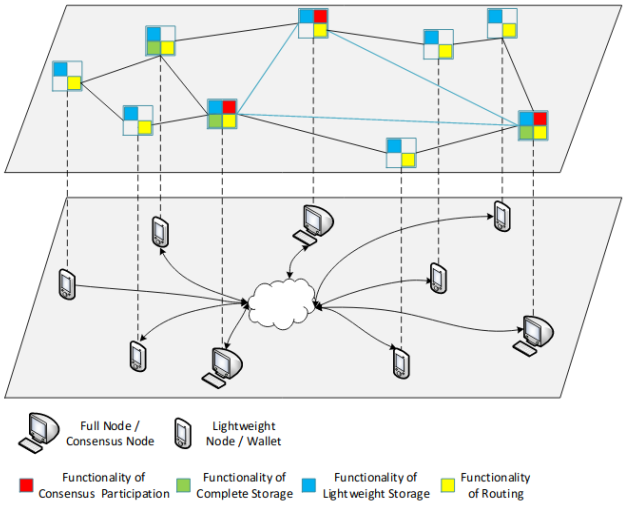


Figure 1.2: Permissionless blockchain network. The P2P links between consensus nodes are shown in blue.][60]?????

### 1.3.4 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 valid block and the miner of the valid block also receive $\frac{1}{32}$ extra ether for each uncle block[36].

### 1.3.5 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. Miner in pool mine together and rewards will split to all members in pool[36].

### 1.3.6 Ether

Is the form of payment and as fuel for Ethereum. The base fo 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 into blockchain and miner just receives 2-3 ether[36].

### 1.3.7 Merkler Tree

all transactions are stored in tree structure wherein leaves contain transactions and the internal node contains the hash of its subtree and a single root also contains the hash of two children and represents the top of the tree. The purpose of bottom-up hashing is that if an attacker attempts to create fake transactions into the bottom of the tree. This will change the node above and subsequently change the root. Thus altering the hash of block causing to register new block. Only the sequence of hash form root the leaf called the Merkler tree[Buterin].

Merkler tree helps the blockcchain by providing tree-structure of transactions. The structure can be described the branches $b$ and main branch is root $b_1...n$ and is connected by

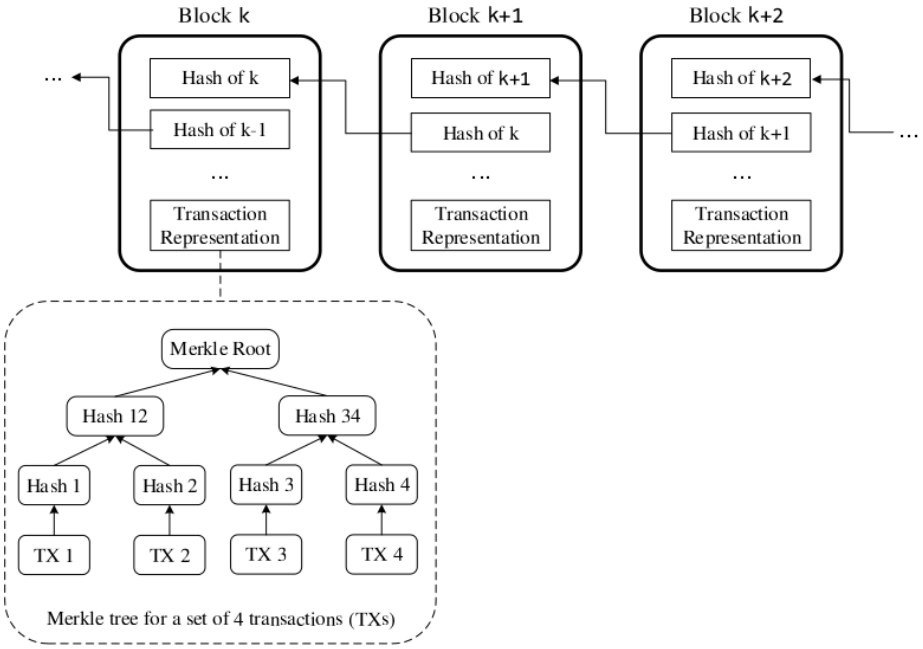


Figure 1.3: Illustration of chain of blocks and markler tree in single block [60]

its concatenations hash $b_{1..\frac{n}{2}}, b_{\frac{n}{2}+1...n}$, this is:

$$b_{1,...,n} = h(b_{1,...,\frac{n}{2}} + b_{\frac{n}{2}+1,...,n})$$

And branches are :

$$(b_{1,...,\frac{n}{2}}, b_{\frac{n}{2}+1,...,n}) = (h(b_{1,...,\frac{n}{4}} + b_{\frac{n}{4}+1,...,\frac{n}{2}}), h(b_{\frac{n}{2}+1,...,\frac{3n}{4}} + b_{\frac{3n}{4}+1,...,n}))$$

Then , Binary branching continue down:

$$b_1 = h(T_1), ..., b_n = h(T_n)$$

Where T is the transaction in blockchain. Generally, it takes $\log_2(n)$ branches to check transaction is $h(T) \in b_1, ..., n$[7].

### 1.3.8 Hash Function

It is a mathematical method to apply cryptographic (secret writing) function. A hash function is deterministic means for input to produce the same output each time. Altering

data generate different output. According to Wikipedia, the security level of the hash function has different properties:

- Pre-image resistance: It is not easy, for initial input with $h$ hash value find the output value $m$ , in a way that $h = hash(m)$
- Second pre-image resistance: It is not easy, for input $m_1$ find another input $m_2$ in way that $hash(m_1) = hash(m_2)$ and both have same result.
- Collision Resistance : It is difficult to find two different input $(m_1, m_2)$ that have same output in way that$hash(m_1) = hash(m_2)$.

The hash function used in blockchain and it is the most secure hashing algorithm is SHA256 which that provides the combination for a given input with 256-bit length. Using SHA256 in blockchain makes to be impossible to duplicate hash because there is a diverse combination of this input with this length and it requires a huge amount of computational works. That means there are $2^{256}$ hash values[37].

| Input value | SHA256, message hash |
|---|---|
| 1 | 6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b |
| 2 | d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35 |
| Blockchain | 625da44e4eaf58d61cf048d168aa6f5e492dea166d8bb54ec06c30de07db57e1 |

### 1.3.9 Blocks

Blockchain consists of blocks to record a series of transactions. Blocks are like containers that every on see metadata but only an owner has a key. Each block contains a list of transactions which once the block is accepted by the network, by some consensus algorithm, will be broadcasted to the network and included in the blockchain[**?** ].

### 1.3.10 Header

It is like metadata on the top of the block which includes a reference to the previous block, timestamp containing the time of creation, Merkle root which is an efficient data verification structure, nonce[7].

### 1.3.11 Nonce

it stands for *number used only once*. It is integer number and along with block data, number, and previous hash use as input in the SHA256 function to generate the current block hash. We used this nonce to vary the hash of the current block and without modifying the data inside the block. By the usage of the nonce, a miner can generate a valid hash, adding first own block into blockchain and get the reward[37].

### 1.3.12 Merkler Root

All transactions contained in the block can be hashed together and create a one-line hash text which is the concatenation of all transactions in the block. This is the root hash of the Merkler tree that gives a short representation of each transaction in a block????.

### 1.3.13 Transaction

transfer data (asset) between users, an transaction contains information such:
- The recipients which message to send
- Signature of sender
- A mount of ether to transfer
- Structure value, the maximum number of computational step that transaction is allowed to do
- Gas price, fees that should payâĂę[36].

### 1.3.14 Consensus Algorithm

The consensus algorithm is used to get the agreement of all participants about what block appends to the chain. By the use of this algorithm, real participant and malicious can be recognized. A blockchain is a consensus, if block $b_t$ is before block$b_{t+1}$ while no other participant can append block in reverse order. In this section, I will go through some important consensus algorithm which is better suited for blockchain[7]:

#### 1.3.14.1 Proof Of Stack

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 chances of a node mining the next block depending on node balance. In private networks, however, where the participants are

known, costly consensus mechanisms such as proof-of-work are not required. This partially removes the need for mining and give wide ranges of consensus protocol for picking node[17].

#### 1.3.14.2 Proof Of Work(POW)

It is a mechanism that ensures consensus is done without any central control. With POW miners compete to complete its transaction first into blockchain and get rewards(e.g: Bitcoin, Ether).
Miners connected to blockchain and accomplish task validating transactions to add new block by solving a cryptographic puzzle and anybody who complete own task sooner can add own block first in blockchain[29].

## 1.4 Bitcoin

The basic goal of blockchain technology is to ensure people form trustworthiness and legitimacy of transactions. Blockchain initially used to enhance commercial transactions through a currency called Bitcoin.
Bitcoin is the digital records or cryptocurrency that is accepted by users involved in the transaction. Bitcoin is the financial use case of this powerful technology[2]. Bitcoin is the list of blocks of transactions. Each block in the blockchain is identified by the hash algorithm on the top of the block. Bitcoin is introduced by a consensus mechanism of blockchain, it is a well-known implementation of decentralized cryptocurrency.
Bitcoin is the limit of the block, wherein each block is verified by a hash algorithm using SHA256 cryptography on the top of the block. Each block encompasses the hash of its parent (previous block) in the own header that refers to it. It forms a blocklist wherein each block refers to the previous block in the list name genesis block. Altering in block implies creating a new block on the top. Since each block contains the hash of its parent creating a new block is expensive and needs proof of work that the miners allow the add new block[29].

## 1.5 Ethereum

It is another cryptocurrency similar to Bitcoin that built on the top of the blockchain. the participant publishes the transaction on the network that is then divided into a node

(called the miner) and add to the blockchain using a consensus mechanism. The state of the system refers to the sate of account that can be an external account related to the user of the system(that contains information about balance) or contract account that obtain 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 transfer the Ether but calling the contract account to execute the code of that contract and may perform a transaction or change the storage of that account[14].

### 1.5.1 Gas

Transaction in Etheruem 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 transaction rolled back but consumed gas will not be returned.
To enable easier calculations, Ether also has some sub-denominations[36]:

- Wei - $10^0$
- Szabo - $10^12$
- Finney - $10^15$
- Ether - $10^18$

### 1.5.2 Account

There are two types of accounts in Ethereum:
- *Normally controlled* is an account controlled by the private key. if Person has the private key can send message and ether from this account.
- *Contract* is an account controlled by code. It is a normal account with an extra option of containing code. Ethereum blockchain starts firing transactions from an account in which this transaction is the response of receiving transactions by an account[36].

### 1.5.3 Message

As already said blockchain fire transaction 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[36].

### 1.5.4 Ethereumt Virtual Machine(EVM)

It handles the state of contracts and it builds on stack-based language with some instruction like opcode. A contract is a collection of opcode statements that are executed on EVM. EVM can be assumed as a decentralized computer which all smart contract run. It is a network of the smaller interconnected machine but sounds to be a big computer. Transactions executed smart contract on each node on the network and each node gather transactions sent from users to block to append in blockchain to collect rewards.
To ensure resource handling of the EVM, every instruction the EVM executes costs gas. Operations with more computational resources cost more gas than operations with less computational resources. Therefore, using gas is beneficial due to encourages developers to write quality applications and avoiding wasteful code. When it comes to paying for gas, a transaction fee is in small amounts of Ether, and the token with which miners are rewarded for executing transactions and producing blocks[38].

### 1.5.5 Solidity

os high-level truing complete language with Java script similar syntax. The contract is similar to classes in an object-oriented language. The contract contains the fields as persistent storage of contracts and methods to be invoked by internal and external transactions. For interacting with another contract, either need to create a new instance of this contract or make a transaction to a known contract address.
In particular, Solidity provides for accessing the transaction and block information like: *msg:sender* or *msg:value* to access the amount of *wei* transferred by transaction that invoke the method. Solidity uses some fuctions to rransfer money ro another contracts suhc as *call* and *send*. A value trnsfer using this function to internal call transaction which implies calling contract also execute code or may fail to execute due to insufficient gas. Another contract is *fallback* function that get execute via *call* and *send* function was preformed [14].

## 1.6 smart contract

A smart contract in computer science is the piece of code that is designed to execute certain terms by the predefined condition. These terms are embedded and preformed on a distributed ledger. As compared to a traditional contract that includes the third party to execute terms of condition, the smart contract has low transaction fees and is more profitable without needing to the third party.

*A smart contract is a self-executed and self-enforced agreement.*

Sometimes, smart contracts and DLT are remembered as the same thing, but not. They are different technologies that are complementary to each other. However, there are several platforms in DLT on which smart contracts can execute. By existing computer and capability of executing code, why have not smart contracts developed?

This question shows the smart contract and DLT and the relationship between each other. A computer is capable of executing an event such as payment of contact when pre-defined conditions met. But, that would have meant that both parties require to have programmed on their computer. subsequently, the version of codes or using the program may differ that would be another issue.

What DLT has done, is that to bind the parties to each other by embedded code in a distributed ledger. More importantly, DLT ensures both parties have secure transactions and the contract will execute automatically. This is the exact description of smart contracts as 'self-executed' and 'self-enforced'. With the advent of distributed ledger, needing an efficient query of diverse data and indexing entries become more important. Within the blockchain, a smart contract is actually coded inside the block and it invokes by receiving a transaction or message and send the transaction in the response of transaction that has received then it can read, write or create the contract.

A smart contract is an independent factor a behalf of the user to perform some operations as long to users' goals. These goals are programmed inside the contracts as code. Each contact controlled own Ether, key as long-lasting storage to follow the constant variables [36].

Let us clarify more by one example: Consider blockchain network where *Bob, Alice, and Carol* are participant and there exist *2 assets X and Y. Box* uses the smart contract that defines three functions: *'deposit'*: store unit of X into a contract, *'trade'* send back one unit if X instead of five units of Y and *'withdraw'* to roll back all asset into the contract.

*Bob* starts activating smart contract by calling the *'deposit'* function and moving 3 unit of X asset into a contract and record it in the blockchain. *Alice* has 12 unit of Y, and sending a transaction and 'trade' 10 unit of Y and get back 2 unit of Y and recorded into blockchain as well. *Bob* signed transaction to *'withdraw'* function. Contract check signature and *'withdraw'* is called by owner, then transfer all deposits 1 unit of x and 10 unit of Y to Bob[17].

There are some features of smart contact out of this example:

- Contract has its states and control over them. These states can be updated by executing the contract. The blockchain keeps a record of all previous states of a contract because overwriting the previous state would involve overwriting records earlier in the blockchain.
- Contract allows us to do business logic in code.
- Contract is deterministic means for the same input produce the same output.
- Smart contracts can be used to implement dynamic data storage.
- Correct smart contract describes all possible results.
- Data explains Relationship between participants.
- Smart contract is triggered by receiving a transaction and sent the transaction afterward.
- Since contract stores on the blockchain, that can be visible by every participant on the network.
- Each participant can trace contract operation due to *sign* message using cryptographically verification.

### 1.6.1 Security in Smart contract

A survey on the smart contract shows that the Bitcoin and Ethereum focused mostly on financial contracts. That's why the smart contract must execute and perform correctly. A security issue in such a structure is to be used for a malicious purpose such as DAO hack or some incidents in Bitcoin. Such incidents caused a hard fork of blockchain to mitigate the malicious transaction. Atzei et al. list 12 vulnerabilities that are assigned by Solidity, EVM, and blockchain. Such vulnerabilities can be addressed by secure smart contracts[38].

### 1.6.2 Vulnerabilities in Ethereum Smart Contracts

Atzei et al. in this section categorized the vulnerabilities in smart contracts into three classes(solidity, Ethereum virtual machine, blockchain:

| Level | Vulnerabilities |
|---|---|
| Solidity | Call to the unknown |
| | Gasless send |
| | Exception disorders |
| | Type casts |
| | Reentrancy |
| | Keeping secrets |
| EVM | Immutable bugs |
| | Ether lost in trasfer |
| | Stack size limit |
| Blockchain | Unpredictable state |
| | Generating randomness |
| | Time constraints |

**- Call to Unknown:** One invokes the function and transfer the ether to the counterpart. If there is no signature in the given address, then the callback function is executed.

**- Exception disorder:** this has occurred in a situation such as execution run out of gas, call stack limit, and *throw* command.

**Gasless send: send** function , transfer ether to contract. This function compiled the same way of *call* function without signature and returns *out of gas* exception.
Assume *call* has no signature, so it invokes *callee's* fallback function. However, the upper bound of the unit of gas is 2300 that is available to *callee* and is executable.

**Type casts** compiler in solidity may be faced to some errors such as assigning an integer to type string

**Reentrancy** the nature of transaction cause the programmer to believe that invoking non-recursive function can not *re-enter* before termination. But it is not always true, because callback function may allow the attacker to *re-enter* caller function which cause the unexpected behavior or invocation loops and consume all gases.

**Keeping secrets** Fields of contract can be public and private. However, using a private field can not guarantee its secrecy. Because the executing contract will publish transactions on the blockchain. using suitable cryptographic techniques can remove this

problem.

**Immutable bugs** AS contract publishes on a blockchain, that can not be changed anymore, if there is an error on contract, it can not patch it. there is no way to predict the error of contract and terminate implementation.

**Ether lost in transfer** when the ether associated with any user or address. ether will lose forever. programmer must be sure to send ether to a valid user address.

**Stack size limit** whenever contract call another contract *call stack* increases one frame. *call catack* is bounded 1204 frames: when the stack exceeds this limit size. A further invocation will give an exception.

**Unpredictable state** when users send transactions on the blockchain to call smart contracts. Users can not be sure if the contract transaction time and transactions received by the user are in the same state. Because other transactions have changed the contract state.

**Generating randomness** many contracts generate random numbers where all miners chose a unique initialization seed.

**Time constraints** or timestamp is used to determine which actions are mandatory in the state.

# 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 that initial block 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 the untrusted external sources. Ethereum smart contracts have an account that can normally store, update, or make a function with the input and output. The concept*accounts* are widely used in this study refer to an agent such as human and the concept *balance* refers to cryptocurrency in blockchains.
As already said, smart contracts are time-ordered where data are stored into blocks. therefore it 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 worlds for more of 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 account, blocks related to distributed level, and data can be stored or retrieved here. In the functional level, smart contracts contain a lot of functional interfaces that depict the other functionality of platforms such as Ethereum [3].

### 2.1.1 Why do we use ontology for Blockchain?

Generally, Blockchain is the distributed database that replicated over all nodes as a cloud computing architecture. These databases are distributed across multiple organizations.

That's why standard interpretation is needed that the data would 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 comes to play to ensure a common interpretation of data of shared database among different enterprises. blockchain as a modeling form used a different type of ontology:
***informal/semi ontology*** to facilitate search and enhance 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. On the other word, blockchain modeling based on formal ontology can help the development of smart contract 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 human. On the other hand, enables interlinking with other link data to convey deductions and formal reasoning[23].
Vocabulary used within ontology increase the transparency of transaction in a way that by describing the transaction in the context of linked data comfort 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 structure using vocabularies like schema.org to interlinked between different data and resources and it helps the semantic query of data. When information represents in linked data, querying about related information can be easily discovered. It is built up to standard web technologies such as
**- URIs**(Uniform Resource identifier) as names.
**- HTTP** to search for names.
**- (SPARQL, RDF)** when a user search for something, provides related information.
**- Link** to other URLs to provide more information[33].

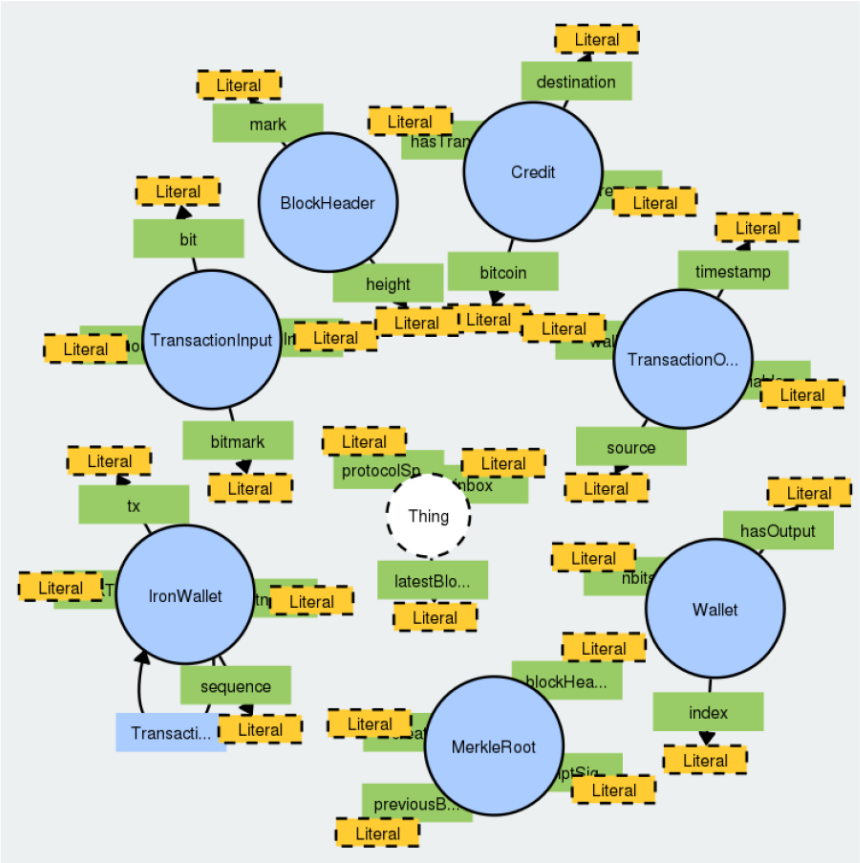


Figure 2.1: Illustration of Ontology diagram[23]

### 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 as a subject that predicts an object is called a triple. i) Subjects that RDF expressions describe them. ii) predict is specific properties, attribute, or relation to describe a resource. iii) The object is the name of property or value. We can build a graph based on these three objects.

### 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.

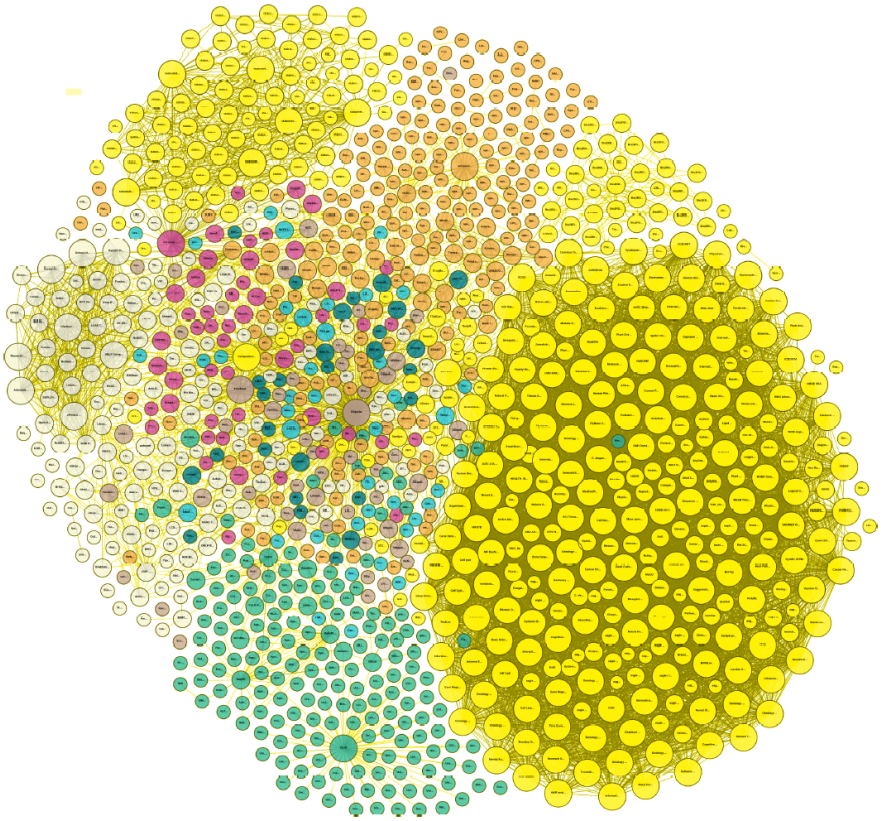

Figure 2.2: Linked data Diagram[33]

### 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.

## 2.2 Vocabularies

### 2.2.1 Vocabulary in Distributed Ledger

To generate linked data, it requires to use a standard ontology or vocabulary to explain the blockchain concepts. Interfaces between distributed ledgers and the Semantic Web are still on an early stage. Some systems define such vocabulary such as FlexLedger,

Ethan, BLONDiE[3].

***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 having concrete ontology for itself.
It is striking to say that the FlexLedger is not suitable to implement in some graph model 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 [27].

***EthOn*** is an OWl ontology that describes blockchain classes such as *"blocks, accounts, message" ,"state"*and relations such *"has parent block"*[34].

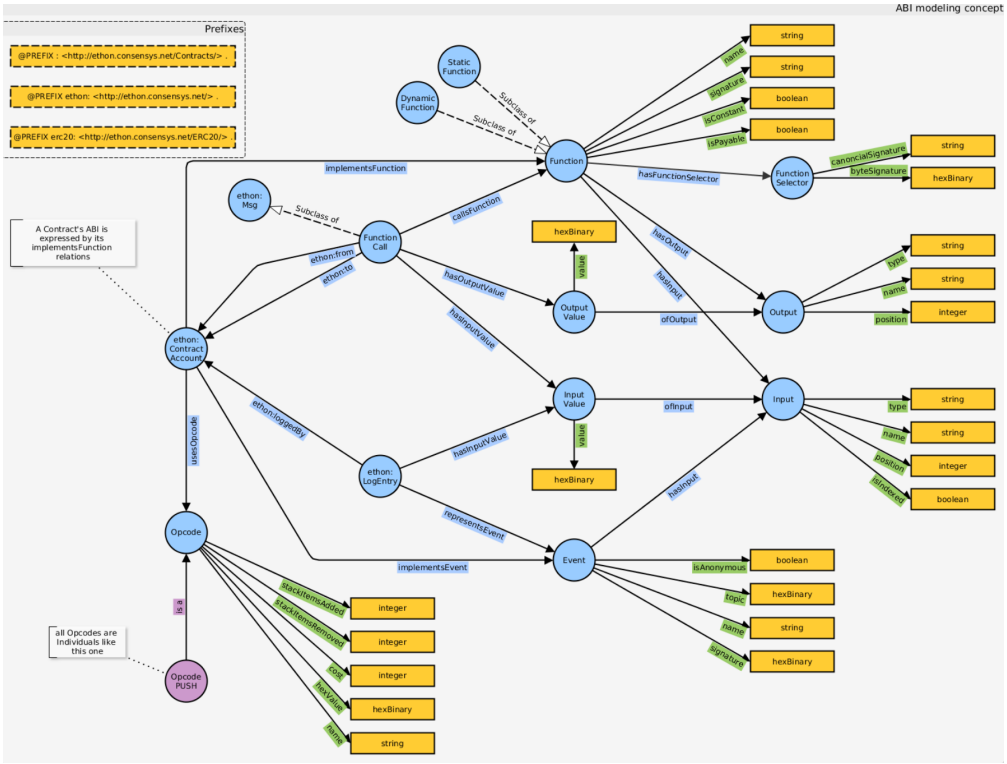


Figure 2.3: EthOn Contract Model(blue arrow is object properties, green arrow is data properties, purple circle is instance and blue one is class)[34]

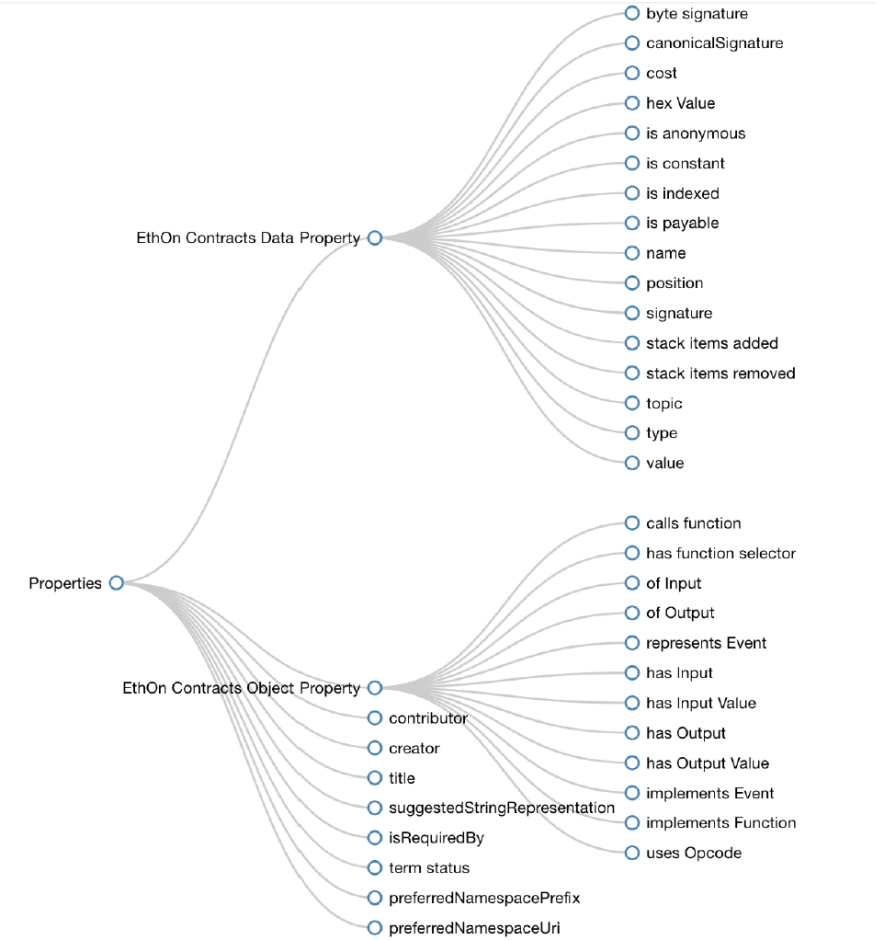


Figure 2.4: EthOn classes[34]

***Blockchain Ontology with Dynamic Extensibility*** BLONDiE is another OWL ontology for describing the Blockchain structure like EthOn. But, it is more generic then EthOn. For example, EthOn and BLONDiE both defined some terms such as 'account', 'block', 'transaction', and some attributes such as 'transaction payload' or 'miner address'. BLONDiE defines some other concept for different Blockchain such as 'BitcoinBlockHeader' and 'EthereumBlockHeader' as subclasses of 'BlockHeader'. At the moment, BLONDiE supports two cryptocurrencies like bitcoin and Ethereum where all links and relationships between objects and attributes represent in RDF(Resource Description Framework)[3].

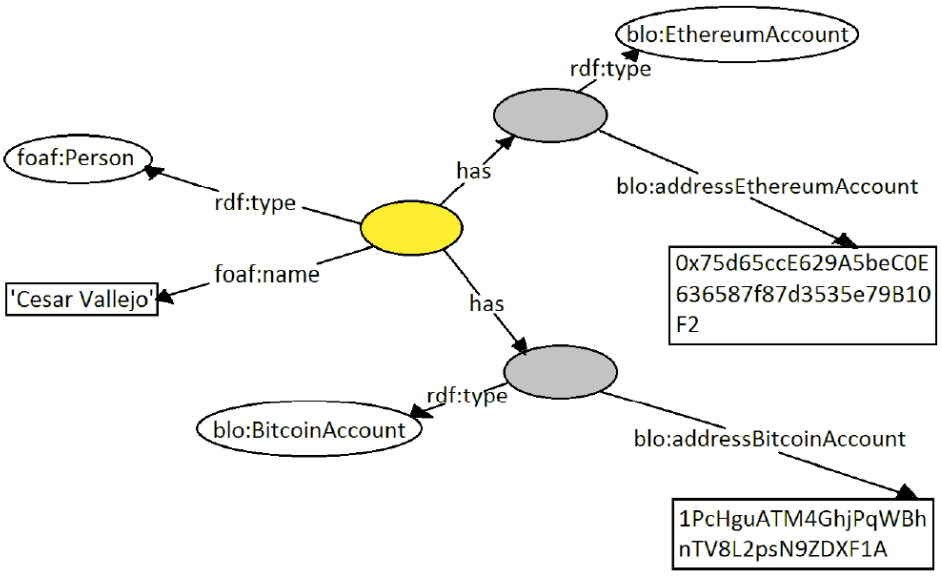


Figure 2.5: BLONDiE usage example[33]

### 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, so the semantic and vocabularies that are applicable for 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 too. It seems that the Combination of a distributed ledger with smart contract and web service due to profitability becomes common[3].

### 2.2.3 Ontology-based smart contract

Kim and Laskowski[15] propose ontology can help in the development of smart contract which can be executed on the blockchain. This approach can come up with proof of concept using Etheruem or other traceability ontology[33].

### 2.2.4 Decentralized Storage and RDF

The main goal of data over the web is to make a data machine-readable format on the web. It causes to interpret data in suitable formats to be useful in a way that is human readable too.

JSON-RPC is a remote procedure call protocol in JSON. This protocol allows sending data to the server without needing to the response. Ethereum blockchain which has properties such as pay fee with Gas. there are different ways of storing data in an RDF triple The most wallet in Ethereum in JSON format that is easily convertible to RDF and front-end is made up of HTML technology that we can use it to embed data on the website interface as Microdata, JSON-LD. The main focus of research is the storage of data on blockchain itself. There exist limitations of storage in Ethereum blockchain and fees. that's why it is suitable to use compact RDF serialization. There are some methods to store data on Ethereum blockchain that are summarized in the table below[33]:

```
{
"id":1,
"jsonrpc":"2.0",
"result": {
    "number": "0x1b4", // 436
    "hash": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331",
    "parentHash": "0x9646252be9520f6e71339a8df9c55e4d7619deeb018d2a3f2d21fc165dde5eb5",
    "nonce": "0xe04d296d2460cfb8472af2c5fd05b5a214109c25688d3704aed5484f9a7792f2",
    "sha3Uncles": "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
    "logsBloom": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331",
    "transactionsRoot": "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
    "stateRoot": "0xd5855eb08b3387c0af375e9cdb6acfc05eb8f519e419b874b6ff2ffda7ed1dff",
    "miner": "0x4e65fda2159562a496f9f3522f89122a3088497a",
    "difficulty": "0x027f07", // 163591
    "totalDifficulty":  "0x027f07", // 163591
    "extraData": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "size":  "0x027f07", // 163591
    "gasLimit": "0x9f759", // 653145
    "gasUsed": "0x9f759", // 653145
    "timestamp": "0x54e34e8e" // 1424182926
    "transactions": [{...},{ ... }]
    "uncles": ["0x1606e5...", "0xd5145a9..."]
  }
}
```

Figure 2.6: Ethereum structure

| Way | Short explanation | Advantages | Disadvantages |
|---|---|---|---|
| Transaction Data Property | Property existing in each transaction on Ethereum | - Not fixed size<br>- Cannot be modified | - Expensive<br>- Stored on hexadecimal format<br>- Is not SPV friendly |
| Contract Storage | Contract state flexible database. Key-value store | - Not fixed size<br>- Easily accesible | - Expensive<br>- Information is modifiable |
| Event Logs | Historical raw data | - Cheap | - Not accesible for smart contracts.<br>- Data generated by the Smart Contract |
| External Storage | Storing it externally and keeping the identifier using one of the above methods | - Unlimited size | - Not guaranteed that data will not be removed |

Figure 2.7: Storage of RDF data on Ethreuem summery[33]

### 2.2.5 Semantic Blockchain

By 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 blockchain in a novel idea and the way of how to apply this technology in blockchain and smart contract is also an as controversial issue.
There are some **definition of semantic blockchain:**
*Semantic blockchain is the representation of stored data on distributed ledger using linked data.*
*Semtcic blockchain is the applying semantic web standard on the blockchain that these standards are based on RDF.*

### 2.2.6 semantify Blockchain process

Semantic blockchain or semantic distributed ledger affects the industrial world and subsequently, the result leads to start developing new application and framework to combine two worlds: thee are some way to sanctify blockchain: -Mapping the basic blockchain 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 data set in blockchain and

then share RDF on the blockchain. - Create semantic blockchain that internal data exchange protocol is based on RDF[33].

### 2.2.7 Semantic Ontology Mapping

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 block and transactions both, have been completed with an attribute for hashing to provide a direct mapping between contents of index and address of entities on the blockchain, Secondly, the transaction is strengthened with link some entities like blocks or smart contract and input to originating contract.
Blockchain stores just a binary form of each contract with metadata. it requires to have the relevant Application Binary Interface (ABI) specification in JSON form to interact with such a contract. This specification is created when the smart contract is compiled and stored in the blockchain and also smart contracts will index on the blockchain in binary form. To interact with contract Application Binary(ABI) Interface specification is needed. This specification is in the form of JSON and created when a smart contract is compiled and stored in the blockchain. The ABI determines all functions of contracts and descriptions about input, output parameter for each contract[3].

By the presence of address and ABI, it is obvious to generate RDF to index smart contracts. Allan Third at el.[3]model smart contract as follow:
- A contract as *msm:Service*
- A function as *msm:operation* with (*msm:hasInput* or *hasOutPut*) related to suitable *msm:MessageContent*
- A msm:Service records contains the blockchain address as an attribute to keep the index into blockchain.

### 2.2.8 Minimal Service Model

The MSM defines a service that has Operations. Operations have input, output, and default MessageContent that may include mandatory or optional MessageParts. Mes-

```
{ "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 2.8: Smart contract of ABI

sagePart provides support for finer-grained input/output discovery, as available in OWL-S and WSMO[34].

We present one example that Rashid et al. used to clarify more the meaning of MSM in the blockchain: In this example, educational data is used to store in the blockchain using Ethereum *web3* library.
it is considered that every block adds to blockchain and retrieve transaction within the block. If the transaction contains a smart contract. Then it retrieves the contract address using Ethereum API. Then, it saves the smart contract address, Application Binary Interface(ABI) as triple in RDF. ABI describes the name of smart contracts and way of calling them. Auther, then saved each method in ABI as an RDF triple based on MSM ontology as follow[34]:

| **Smart Contract Methods** | **MSM** |
|---|---|
| Smart Contract ABI | msm:service |
| Function | msm:opreation |
| Input | msm:messagecontent |
| Output | msm:messagecontent |
| Gas | msm:gas(from EthOn Contract DataProperty 'cost') |

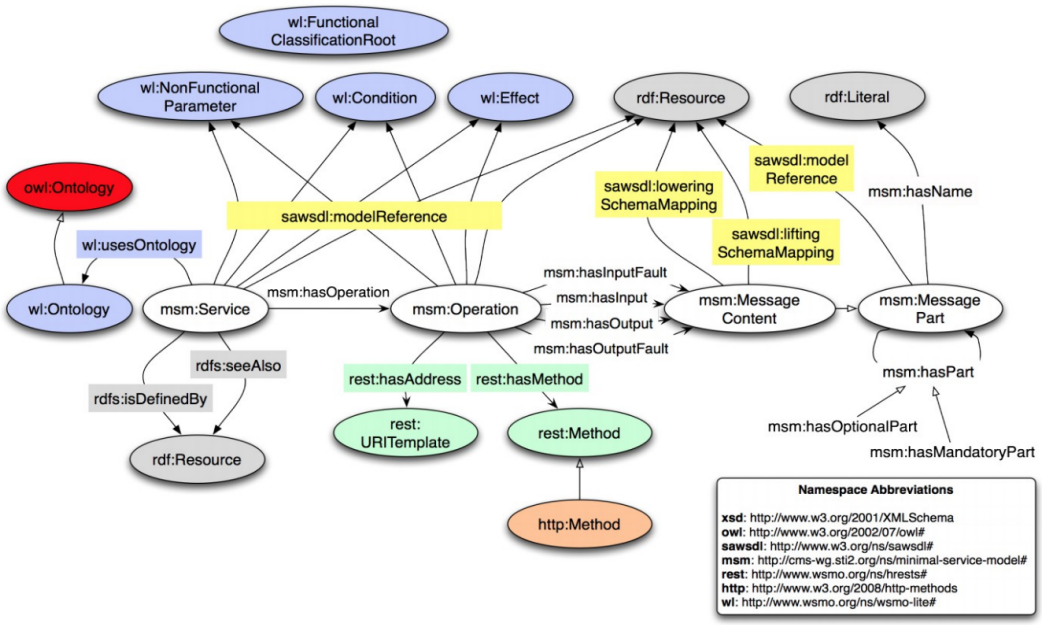


Figure 2.9: Illustration of Minimal Service Model Ontology[34]

### 2.2.9 Web 3.0

[33] The evolution and interaction of people on the Internet emerged on a classification of this revolutionary technology. Early websites formed what is now known as Web 1.0 the âĂIJweb of documentsâĂİ, documents serving as information portals with basic ability to link websites. Later the Web 2.0 emerged as the âĂIJweb of peopleâĂİ, it introduced users to collaborate with content creation and alteration. For the coming Web 3.0 there is a debate about what is the proper definition of its characteristics. For some group, the Web 3.0 is powered by the Semantic Web, where people can access to linked information fast and easy. But now, with the emergence of a decentralized web powered by Blockchain technology and since it enables unmediated transactions, there is a new focus on the Web 3.0 based on the trustful nature of the blockchain. It is the âĂIJread-write-own webâĂİ. Here, the user owns and participate in owning the protocol. It is both peer to peer and machine to machine. And it applies to people, companies, and autonomous entities [14]. For instance, the term Web 3.0 is used by Ethereum in a different context than the suggested by Berners-Lee. It is proposed as

the separation of content from the presentation by removing the need to have servers at all [15]. Stephen Tual, EthereumâĂŹs CCOâĂŹs, defines that what makes Ethereum different than Web 2.0 is that âĂIJthere are no web-servers, and therefore no middleman to take commissions, steal your data or offer it to the NSA, and of course nothing to DDoSâĂİ.

## 2.3 Retrieve Information form semantic blockchain

Reasoning and knowledge representation in the semantic web is based on Description Logic(DL). Description logic has some elements such: classes and properties, the link between a different object, etc. Ontology is the study about the objects and relations between these objects or entities. Each DL has a set of constructors to combine the concept and roles. Concepts used in inclusion and definition axiom that model knowledge domain. These axioms called terminological box(TBox) concerning ontology individual axioms from Assertion Box (ABox). ABox and TBox create Knowledge Base(KB).
This study extends in the keeping with the retrieving the most relevant resources for a given query by the requester, where both query and resources are ontology-based and matched called semantic matchmaking. This process leads to two approach *full match*, *no match*. For request $R$ and resource check whether all features in request exist in $S$. The classic inference service is capable of to Boolean match approach. But, It is not enough because fully matched is rare. Non-standard inference determines the semantic ranking of resources concerning request and logic-based results. If request and resource are not compatible:
- contradiction concept which part $R$ is not compatible with $S$. Consider $G$ is the part that has conflict and $K$ as a part where $S$ and $R$ are compatible.
- Concept Abduction: $R$ and $S$ are compatible but $S$ not match fully with $R$. consider $H$ as a Hypothesis that represents what is request in $R$ and not exist in $S$. If $R$ and $S$ are not matched. Use contraction to represent $K$ part(compatible part) and $H$ hypothesis to suggest part to reach full match.
Moreover, we used the number of an element $G$ and $H$ in penalty function computation to define the semantic distance matrix. This matrix can be used to rank resources concerning request[25].

### 2.3.1 Semantic Blockchain Architecture

This method purposed a framework based on the semantic discovery layer built upon basic blockchain. It is striking to see the main features of such a blockchain as follow:

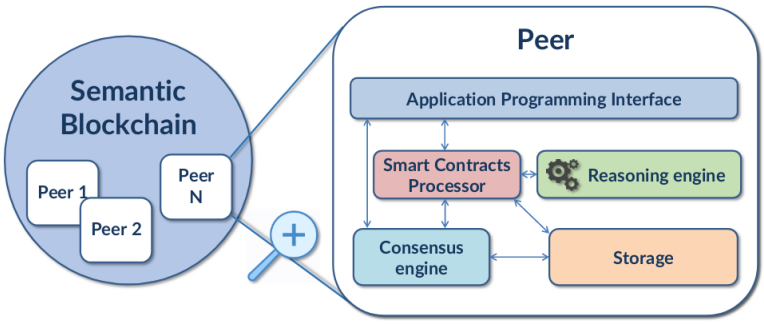


Figure 2.10: Framework Architecture [25]

**Peer** agent is identified as by public key and its accounts. Each agent can enforce semantic discovery to transfer assets between annotations that are stored in the blockchain. And each peer is integrated of matchmaking and reasoning engine for semantic discovery.

**Asset** represents the resources and service instances based on domain ontology.
**Smart Contracts** are the integration of matchmaking and reasoning engine.
**Consensus Engine** allows to validate transaction in blockchain.
**Storage** built up *markle tree* out of transactions including semantic ones to efficient detection of erroneous change in transaction[25].

### 2.3.2 Making smarter Contract: Putting Semantic in Consensus Protocol

In the semantic blockchain, EthOn or BLONDiE concepts and vocabulary to facilitate integrity and develop resource discovery. EthOn concept used W3CRDF schema and web ontology language to describe blockchain contract concepts. By the use of these concepts, it is possible to compare requests with different recourse concerning semantic annotation of shared domain ontology.
Smart contract semantic is represented by a service layer that gives a correct description of discovery outcome. Thus, it raises the trust of the discovery process for users. In the

semantic blockchain, Baqa et al.[12] used domain ontology to maps to EthOn contract. These concepts used for OWL-S ontology, indexing, and invoking smart contracts on Blockchain via URIs[12]. Semantic blockchain also performs operations such as registration, discovery, selection, and execution that are implemented as a smart contract.
Ruta at el.[25] focused on semantic mismatching as a significant feature of semantic blockchain concerning basic blockchain. This allows computing the semantic distance between resources and queries with the same ontology. The logic-base matrix enforces the semantic ranking of the element of the query[25].
**A: Resource Registration:** As already mentioned, a Smart contract designed using OWL specification. It used concepts and entities defined as a class in OWl language and relationship between entities defined as object properties concerning ontology. To construct smart contract semantic requires a framework to design such contracts.
Multiple resources stored on the blockchain and Each domain is related to a different ontology that provides vocabularies for annotating resources. Resources also are categorized by attributes such as URI of reference ontology to retrieve the resource, semantic annotation in OWL that describes resources, resource price.
To register resources on the blockchain, a user is required to register an account with public address and related private key to call the smart contract as a parameter. In ontology, a user describes as class corresponding to an account along with other attributes such as an address, private key, and other details.
To achieve this, A new ontology called *EthereumContractConcepts* is implemented containing *EthereumContract* data property. A *ContratcAccount* consist of license number, date, smart contract owner and state. After all, a smart contract is developed with vocabulary as domain-based-ontology[12].

**B: Smart Contract** EthOn and OWl concepts used to build semantic web service for smart contracts. OWL-S ontology is used that makes functionalities such discover, invoke, and monitor by providing some additional vocabulary along with smart contract concepts to facilitate query to finding a smart contract. The OWL-S ontology provides three types of description about service as follow:

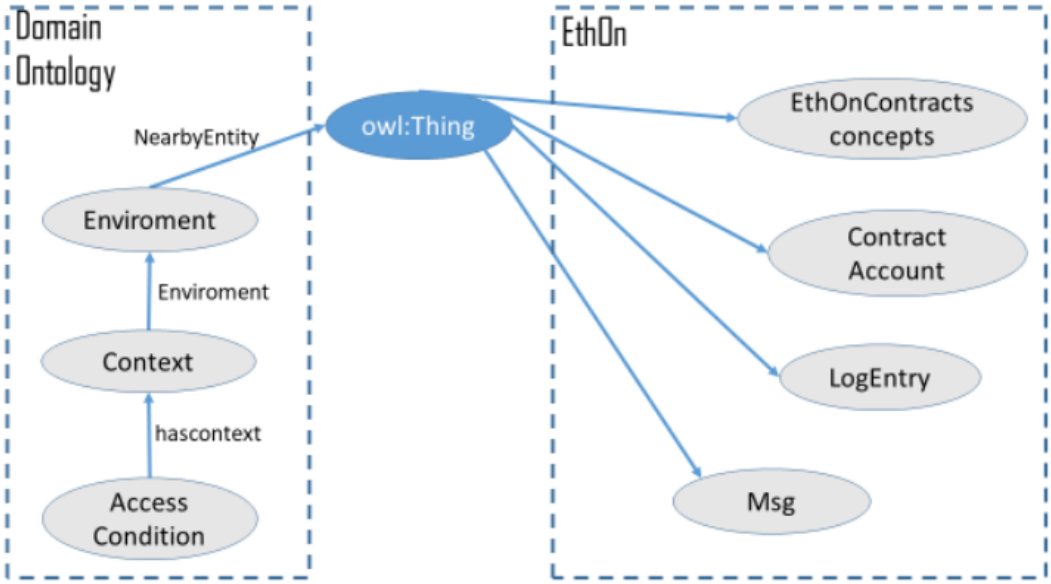


Figure 2.11: Domain ontology [12]

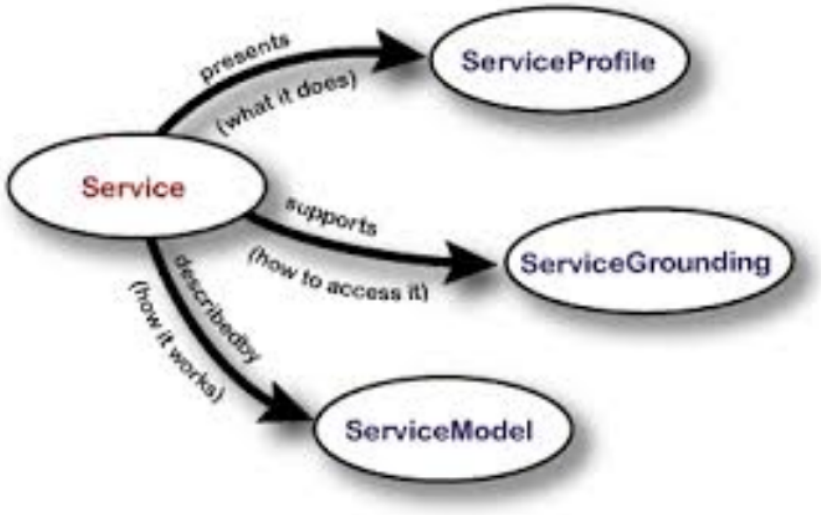


Figure 2.12: Service-based smart contract

By the use of EthOn concepts, we can monitor added block to the blockchain. When a transaction has a smart contract, it is retrieved from the store contract address via API, then the address of smart contract and Application Binary Interface(ABI) as a triple in the RDF store. ABI consist of the smart contract method and way of calling it. Methods are stored in ABI as an RDF triple concerning OWL-S ontology. The table below shows the OWL-S vocabulary to describe smart contract[12].

TABLE I
EXTENDED OWL-S ONTOLOGY FOR SMART CONTRACTS

| Smart Contract Methods | OWL-S |
|---|---|
| Smart Contract ABI | owls:service |
| Input | owls:ServiceModel |
| Output | owls:ServiceModel |
| Function | owls:ServiceProfile |

Figure 2.13: OWLS ontology for smart contract

**B:Semantic Discovery** To search for a smart contract or an item, The requester sends the request to $n$ nodes randomly specifying:
- URI of domain ontology to determine resource and vocabulary of both resource and request which to be retrieved.
- Semantic annotation of smart contracts in OWL language.
- Maximum Price $p_max$ that requester pays.
- Minimum semantic $s_min$ threshold in $[0, 1]$ interval, 1 is related to full match and 0 is the mismatch. - maximum result $r_max$ is to be returned. - Address of requester.

Baqa et al. here used the *gossip* approach to propagate the request. The nodes which received the request perform 2 operations at the same time: First, preform semantic matchmaking of own resource with the request and provides a list of max result $r_max$ satisfying both semantic relevance score $s_min$ and cost $p_i <= p_{max}$, is returned.
Second, send the request to other $n$ nodes randomly, the other nodes perform in the same way until reach the $m$ thresholds. the request continue until reach the $\sum_{i=1}^{m} n^i$

with $n$ and $n$ parameters. Then do not forward the request and perform matchmaking locally.
Afterward, the nods send back the result to the main requester at the specified address[12].

**C: Explanation** In this process, the requester sends the request containing: Semantic is the annotation of request and URI od related resource. The receiver node response matchmaking result that encompasses the semantic dependency score in [0,1]interval and the concept expression of G (conflicting part between R and S), K(the compatible part between R and S), and H(the hypothesis concept to reach the matchmaking). This process is optional and needed when the requester needs the explanation of matchmaking results.

**C: Resource Selection:** After receiving all results, the requester selects the best resource and smart contract, sending a message to the resource owner and contextually payment. the receiver responses with the proper resource representation relied on the meaning of uri[4]. The resource discovery and retrieval interaction are shown bellow and Each associated transaction is recorded on the blockchain[25].

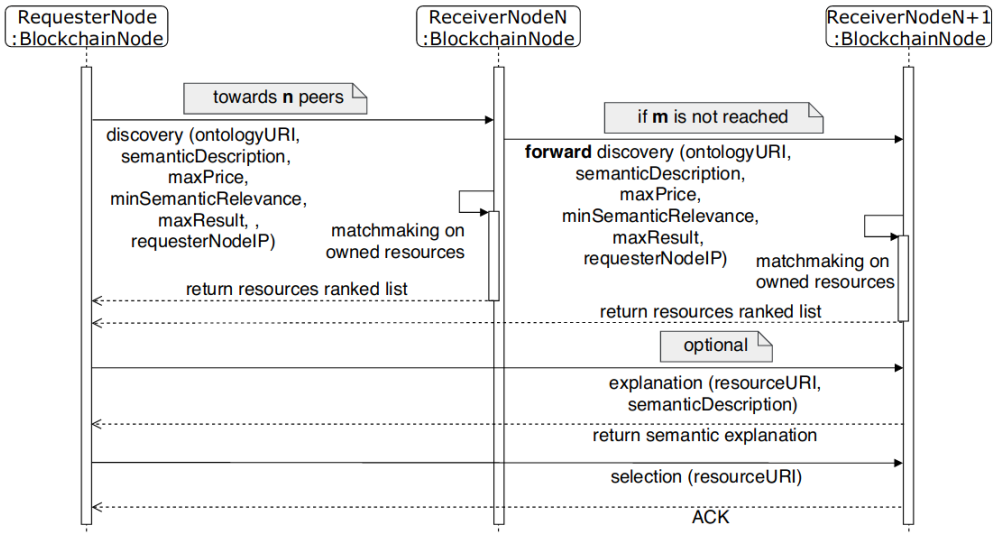


Figure 2.14: Resource Discovery[25]

### 2.3.3 Indexng of Smart Contract

As already mentioned, the distributed ledger does not have a central registry due to its structure. The contract in this distributed ledger is not directly queried. and blockchain is the time-ordered structure where data exist on multiple blocks. In such a distributed structure, developing the indexing process for the smart contract is necessary. The indexing system provides the ability to analyze and search service on blockchain and displays the outside. There is a different level of indexing such as low level to index basic entities such as account, transaction, block and the top level for more functional operations is indexed.

### 2.3.4 EthOn

Is an ontology that describes blockchain concepts such as transaction, account, block using w3RDF schema and ontology web language OWL to describe the relation of these objects on the blockchain. To model data and be understandable for resource, EthOn and some semantic such as ”hasParentBlock”. EthOn in at infancy and should develop more to have smart contract concepts, functions, events, input, and output, etc. EthOn accepts consider just some relation between smart contract and Ethereum framework. Some multiple tools and languages can annotate a smart contract. In this study, It is focused on OWL-S ontology due to some significant features such as flexibility, wide description of service, and no- restriction... The main goal is to develop to support the smart contract too for semantic web service context. For reaching this aim, the web graph chain is PI and a smart contract can be represented as an executable function in a distributed environment. The semantic web describes message, service, and entities in a machine-readable format that can support logical reasoning based on semantic web service description. It helps to map between different ontologies that facilitate the transformation of the concepts among ontologies[12].

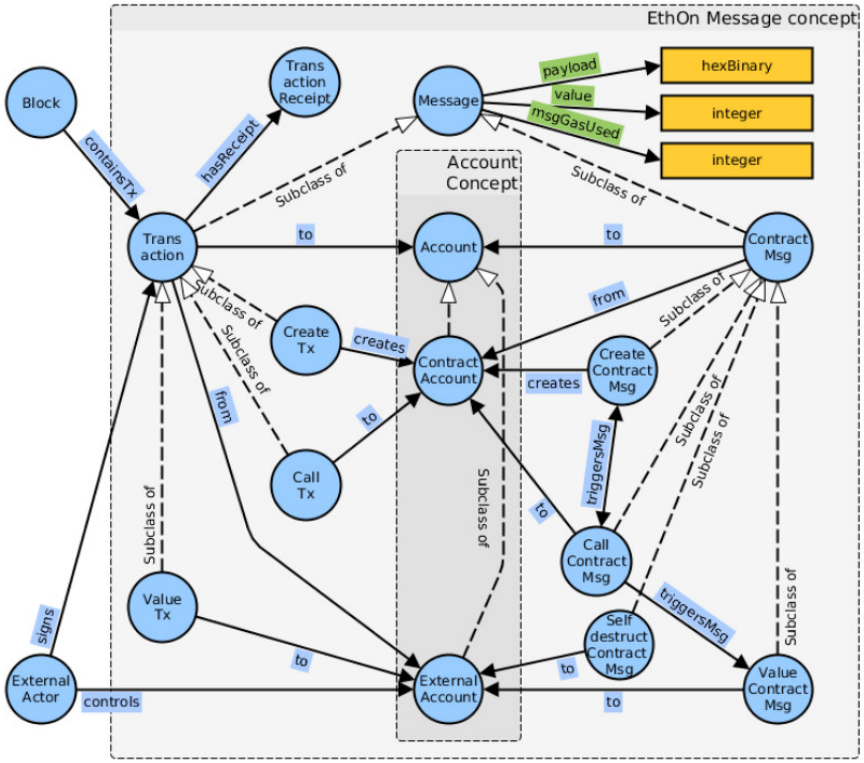


Figure 2.15: EthOn message concept[12]

# Chapter 3

# Implementation: Dapp

## 3.1 DALICC

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

### 3.1.1 DALICC Requirements

The following requirements would be addressed by DALICC framework:

- *Tackling license heterogeneity:*
  It is possible to combine the various contents which having the same license with different names. But that would be much difficult to license the resultant of the combined contents. To solve this issue, DALICC provides set of machine-readable representation of licenses that allow 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 express through the same RED. But, it is difficult to compare licenses have been represented by different RELs.
  DALICC resolve this problem, by representing RELs based on semantic web, map the terms to each other. It will represent exiting RELs based on W3C-approved standards, thus allow 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 semantic are aligned. This 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 user with a workflow that define the usage context, then gathering additional information to detect conflicts and ambiguous concepts. Based on this information, DALICC make reasoning over the set of licenses and infers the instruction to user how to process with license processing.

- *Legal validity of representations and machine recommendations:*
  According to Anna Fensel[**?** ], The semantic complexity of licensing issues means that the semantics of RELs must be clearly aligned within the specific application scenario. This includes a correct interpretation of the various national legislations according to the country of origin of a jurisdiction (i.e. German Urheberrecht vs. US copyright), the resolution of problems that are derived from multilinguality and the consideration of existing case law in the resolution of licensing conflicts.[**?** ]
  To solve this problem, DALICC will check the legal validity of machine-readable license and compatibility of reasoning engine output with law. DALICC output will be testes against law and checked the semantic precision of derived from different language and adjusted them.

### 3.1.2 DALICC Software Architecture

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

- **License composer** is tools that allowed the license to be created from set of questions which are mapped to ODRL, ccREL and the DALICC vocabularies and concepts.
- **License library** is a repository that represents licenses in machine-readable format , the former as ORDL policies and the letters as plain text.

- **License annotator** allows to attache license to dataset, either by choosing available licenses in license library or create new license using license composer.
- **License negotiator** as main component in DALICC framework that checks the license compatibility and support license conflicts by detecting equivalence license having various names.

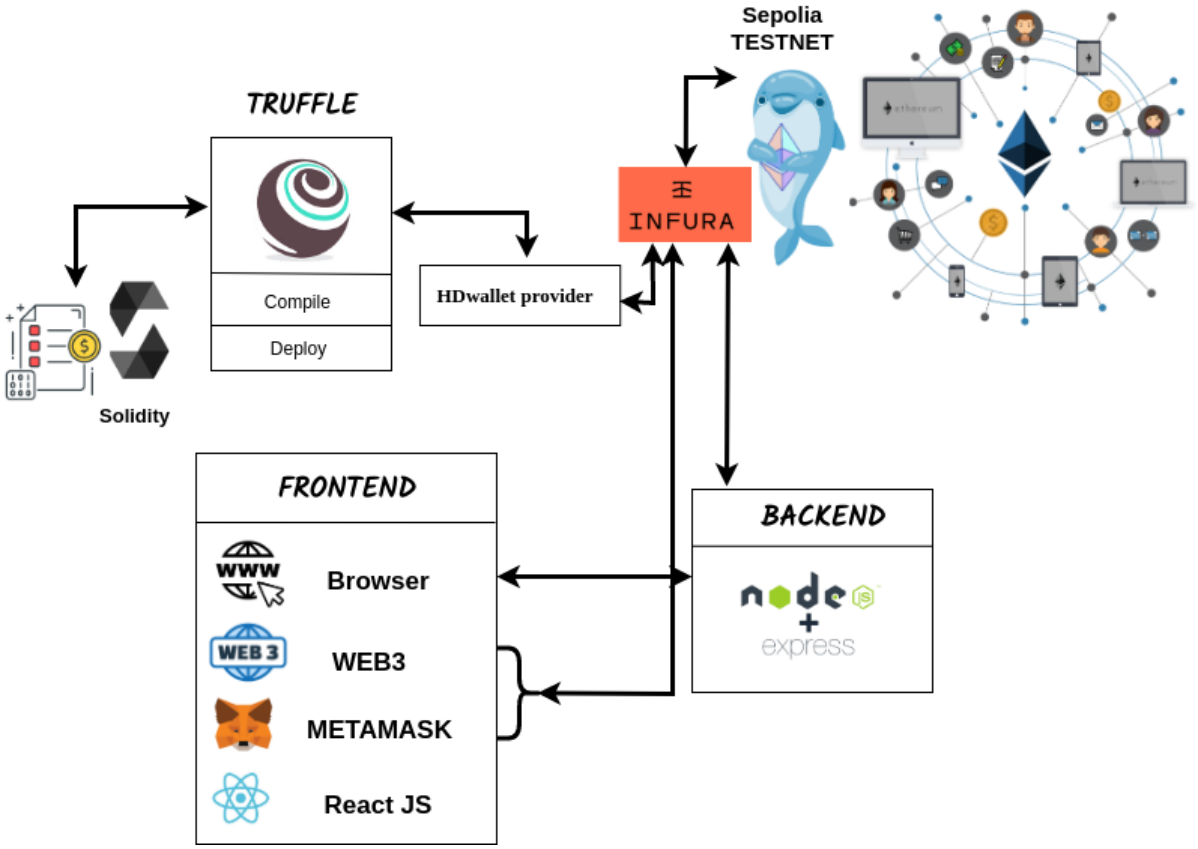


Figure 3.1: DApp infrastructure

## 3.2 Project Concepts

In this section, we focused on requirements that we need to address them in our application. And some terms that we mostly heard of them in smart contract section.
**Triple** use as template in this project. it contains different

concepts likr class and properties associated with project transaction graph.
**SPARQL** is language query that runs over tripled produced by ...

**Http Request**

### 3.2.1 Contract and Ontology specification

Here, it is explained some terms related to smart contract and ontology of our application. And we needed them as 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 msg sender to interact with contract. In this caseed is able to change her/his license of the file.

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

- **Definition 3 (Licensor)** is the address currently interact with smart contract. In this case is able to license his/her file.

- **Definition 4 (Web3 to Ontology mappings)**do not have authority to change stored data on ethereum blockchain, one idea is that to create template to have semantic view of blockchain. In order to reach this purpose, we need mapping between stored transaction data in the blockchain and transaction concepts defined as ontology, then make query on produced data from this mapping. This process consist 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 create mapping between 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 to resources explicitly. Prefixes are declared on the top of SPARQL query and our triple template, so that the statements can reference to.

### 3.2.2 Technology Usage

**DApp**

It is type of open source smart contract application that runs on decentralized peer-to-peer network rather then centralized server. It allows users to transparently execute transaction on distributed network.

**Charactaristic:**

- *Open Source*: all the required are decided on consuse all available users on network.
- *Decentrlized Storage:* data is stored on decentrlized blocks.
- *Validation:* AS application runs on blockchain. They offer validation of data using cryptographic token which is needed of network.

DApp is similar to centralized app, as they use frontend and backend. But backend's app is supported by centralized server or database. And Backensd's DApp runs on decentralized p2p network and is supported by smart contract which stores on ethereum blockchain.

**Ethereum**

it is explained in previous section.

**Solidity**

it is explained in previous section.

**SHA3-256**

he process of SHA-3 standarazation by NIST was completed in Auguts 2015. SHA3 includes four functions with the different length of 224, 256, 384, 512 bits. In all thesis cases, the capacity is twices the output length.

The function blow indicates tha implentation of four functions in SHA-3 statndards.

SHA3[n]*[massage]* which($n$) is the output length of 224, 256, 384 or 521 bits.
*message* is the additional input parameter in text type.

**Java Script Tools**

- `Truffle` is the smart contract development tools and testing network for blockchain application and supports developer who are looking to build etherem dApp, etc. Truffle offers some different features:
  - *Smart contract Managing*: truffle helps to Manage artifacts of smart contract used in dApp and supports library linking, deploying and some other ethereum dApps.
  - *Contract testing system*: Truffle helps developer to construct smart contract testing system for all own contracts.
  - *Network management*: it helped developer by managing its own artifacts.
  - *Truffle console*: allows the developer to access all truffle commands and built contracts [1].

- `React.js` is open-source JavaScript framework and used as frontend. In react, developer build web application by using reusable components that are individual which when assembled together form a application's whole user interface.
  React has advantage of providing feature that combine spedd of JavaScript with more efficient method of managing DOM to render web pages faster and create more responsive web application [2].

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

- `Web3.js` helps developer to connect to ethereum blockchain. *web3.js* is collection of libraries that allows developer to preform some actions like sending ether, checking data from smart contract and creating smart contract [4].

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

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

- `HDwallet provider` is package to helps developer to connect to network. It is easy to configure connection network to ethereum blockchain through *infura*. This provider is used by truffle when we deploy the contract however metamask providers are used when we want to interact with the contract in the browser.
  In our case, we migrate smart contract to Sepolia network. HDwallet provider provide custom rpc url: 'http://127.0.0.1:7545'. This will spawn a development blockchain locally on port 7545 [6].

- `Express.js` It is node.js framework for authority dApp. It provides HTTP methods (GET, POST) to call function for particular URL route [7].
  When we run dApp, we have HTTP server locating on port 3000.

- `FS` provides some functionality to interact with file system, mostly used function like: *readFileSync*, *writeFileSync* and *appendFileSync* [8].

**Semantic Web Tools**

- *EthOn Ontology* is semantic view of ethereum blockchain. It encompasses different classes and relations to cover different concept of ethereum like blocks, transaction, message to formalize RDF triple. We used class and properties related to transaction concept as template to modeling our transaction to DALICC [9].

- *SPARQL*
  It is defined in the previous section.

[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/
[9] https://axios-http.com/docs/intro

- *Sparql* is utility of Appache Jena that run queries on remote sparql endpoint or RDF files that would be lo located on local computer or web. We used the command as follow:
  *sparql –data rdfFile –query sparqlFile*

**Ethereum Tools**

- *Infura*, According to internet definition,a web3 backend and infrastructure-as-a-Service (IaaS) provider that offers range of tools to help developer to build dApp to connect to ethereuem blockchain.
- *Metamask Wallet* metamask is wallet used to interact with ethereum blockchain. It allows to user to connect network through browser extension or mobile app. Metamaskwallet, including all accounts, each account has own privat key [10].
- *Faucet(ETH faucet)* is the platform that give some test tokens to user to test smart contract or sending transaction. For out purpose, Sepolia faucet is used to sending transaction.
- *EthOn Ontology* is semantical view of ethereum blockchain. It encompasses diffrenet classes and relations to cover other aspect of ethereum like blocks, transaction, message to formalize RDF triple. we used class and properties related to transaction concept as template to modeling our transaction to DALICC.
- *Web3.eth* the package that allows to interact with etheruem blockchain. It contains many functions to provide more information about executes smart contract or transaction on blockchain. In our case, some functions including: *getBlock*, *getTransaction* and *getTransactionReceipt* are used to provide some more details which are similar to ethOn ontology. these retrieved properties would be used later in triple.
- - *knowledge graph* provided to indicate classes and relations. the idea being to use graph-based data model to clarify survived transaction in much more details.

[10]https://originstamp.com/blog/metamask-what-is-it-and-how-does-it-work/

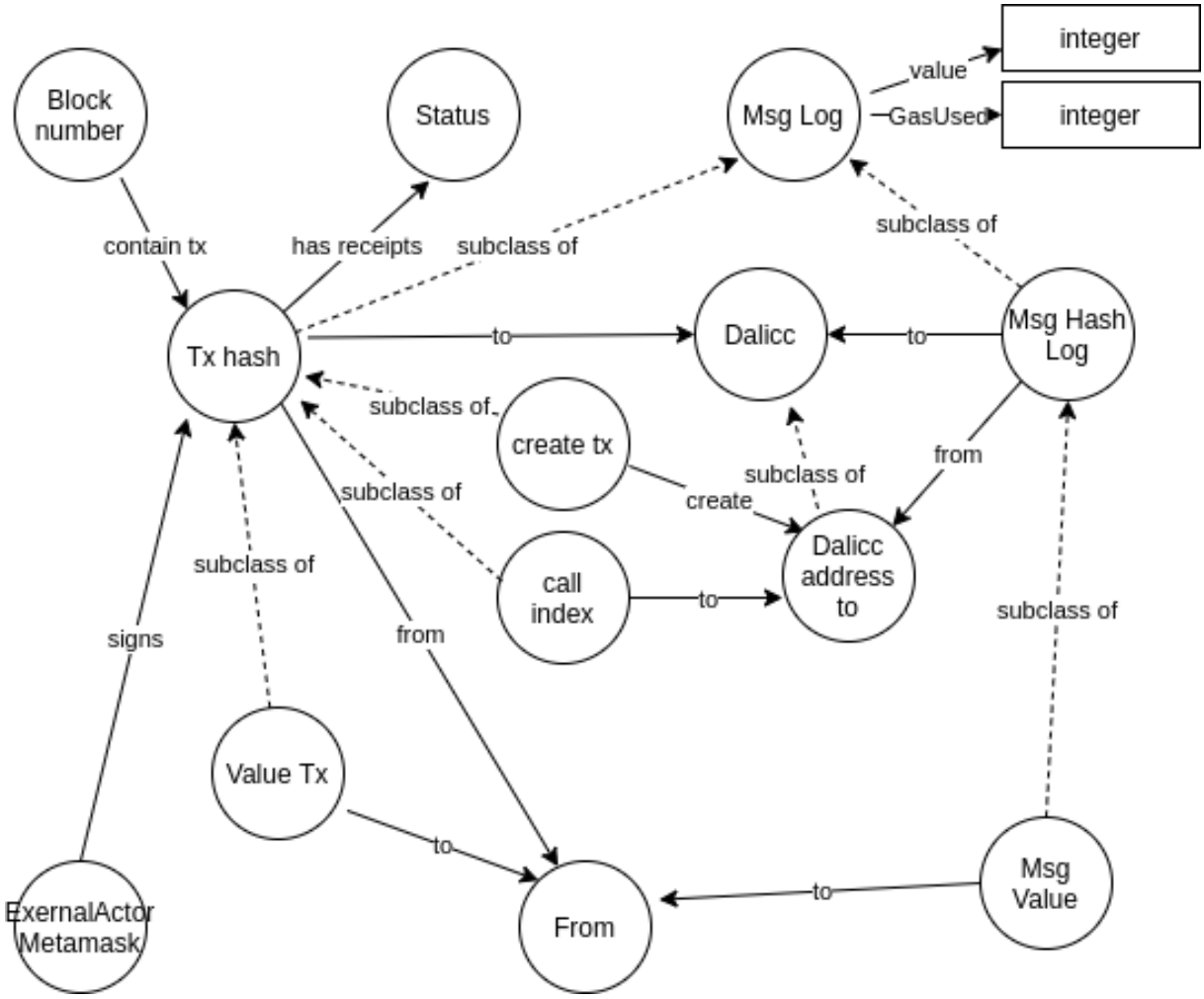


Figure 3.2: Transaction illustration

## 3.3 Project Architecture

### 3.3.1 Backend

The backend of this project rely on 2 main components: First one is the ethereum blockachain using smart contract to create platform.
Second one is DALICC license library is used so that user can license here/his data. In the next part, we will more focused on authenticated user as main challenge in decentralized app and data licensing system by user.

**Authenticated user**

Authentication of users on Ethereum for validation purpose is the essential feature when

building dApp. In this dApp, there are 2 ways of authentication:
- *login to Metamask:* Metamask is the most popular cryptocurrency wallet(ethereum account) to support ethereum blockchain. additional, metamsk is bridge for web3 authentication to ethereum-based dApp. This feature would be used to offer authentication in ethereum.
- *Using smart contract:* the address of metamask passed to smart contract licensor, then it will used this address as owner to license his/her file.

**Licensing in smart contract**

**License information** consists of three elements: licensor address that is passed by metamask. license of TODO

**Storage license information**. Each smart contract runs on ethereun blockchain would be maintain state in its permanent storage. TODO

**Retrieve license information**

As mentioned earlier, only a smart contract can change the data in its memory. Thus, a smart contract system is validating the license requests. The JavaScript library web3.js serves as an interface to the Ethereum Network for the frontend of the web application. This allows the frontend to access the smart contract and thereby retrieve license information as well as change them.TODO

### 3.3.2 Front-end

Frontend of this project build on React.js, HTML, CSS. The React js used in this frontend for user interface.

**User roadmap**

The liceseing Data from user interface includes thress phases:

- The user get asked to select to license type from which be loaded from Dalicc Library via axios get request in frontend. Then, after selecting data in next field, user should check whether there if license for selected file or data and will receive either confirmation message 'Your Data is Licensed before' or rejection message 'there is not license for this file'.

- The user should go further be pressing 'License Data/ retrieve license' button to get license information for confirmation message of the last phase or license his or her file.
  license information contains some information like license type, license URI retrieved from Dalicc library and address of owner of this license. For licensing data, user get asked to login to Metamask for authentication process and this account address would be passed to smart contract as owner of this license. The licensing process will be done by receiving hash data of licensed data.

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

**Interaction with backend**

Since the backend code(smart contract) of the dApp is on decentralised network, we focused on the interaction between smart contract and thereum network. The communication between 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 test network (Sepolia in our case) or ethereum provider TODO.

**interaction with with DALICC**

In order to retrieve license, an HTTP GET request is sent to DALICC library endpoint and returns the license which encompasses to elements: license title and license URI. The user can choose just DALICC title as license then URI of associated title would be stored subsequently in smart contract fro further processing. **Produce Transaction Details**

After commit transaction in blockchain, user can get transaction details via web.eth, then send transaction details in RDF format to server to store into a file. The produced turtle file will be resulted into a readble format and would be return to frontend by http GET resuest from frontend.

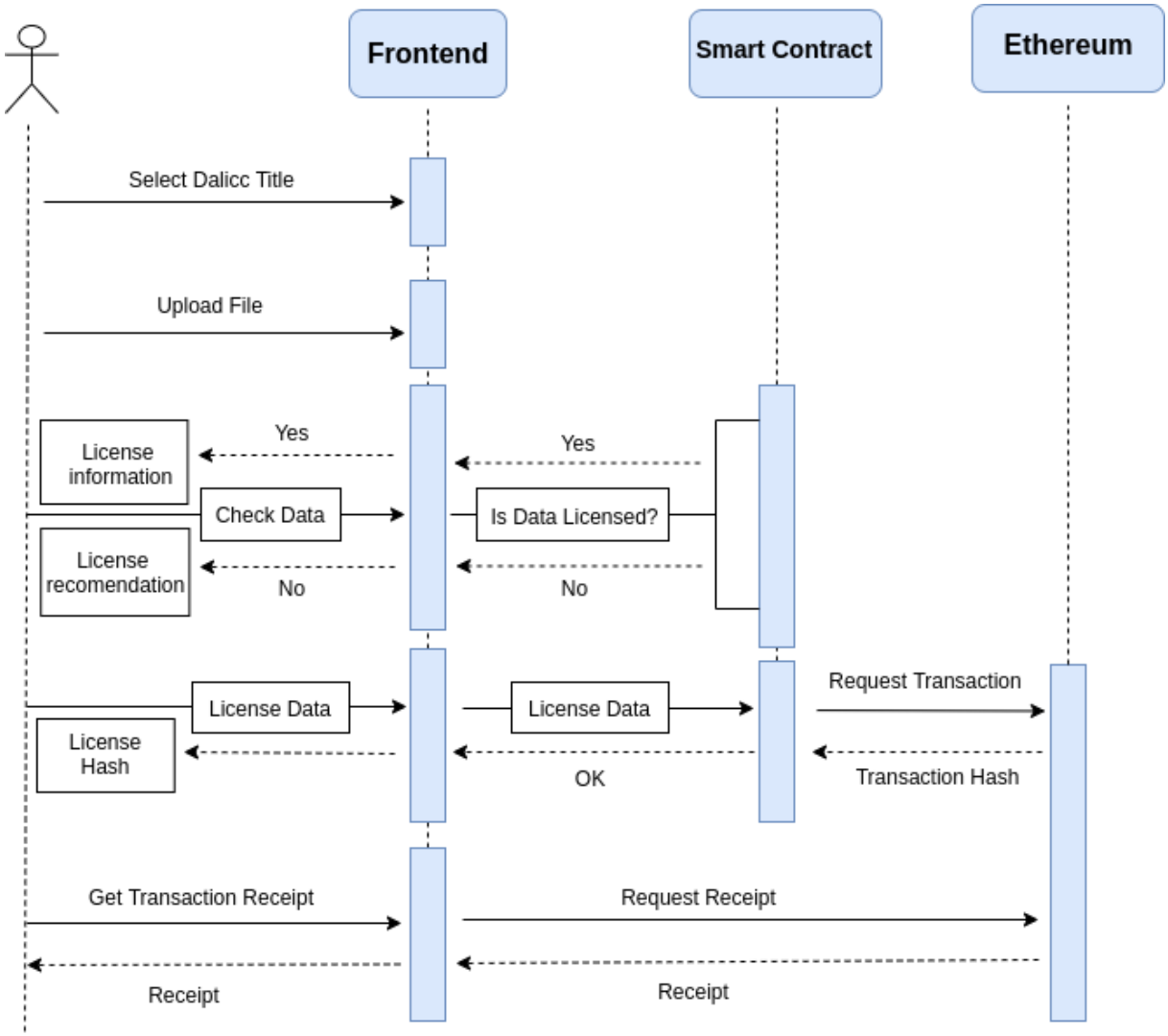


Figure 3.3: DApp Architecture

### 3.3.3 Schematic Thema

This application encompasses some components:

- User interface
- Smart contract to communicate with DALICC library
- Ethereuem network to support transaction
- Transaction receipt

- EthOn ontology
- SPARQL query

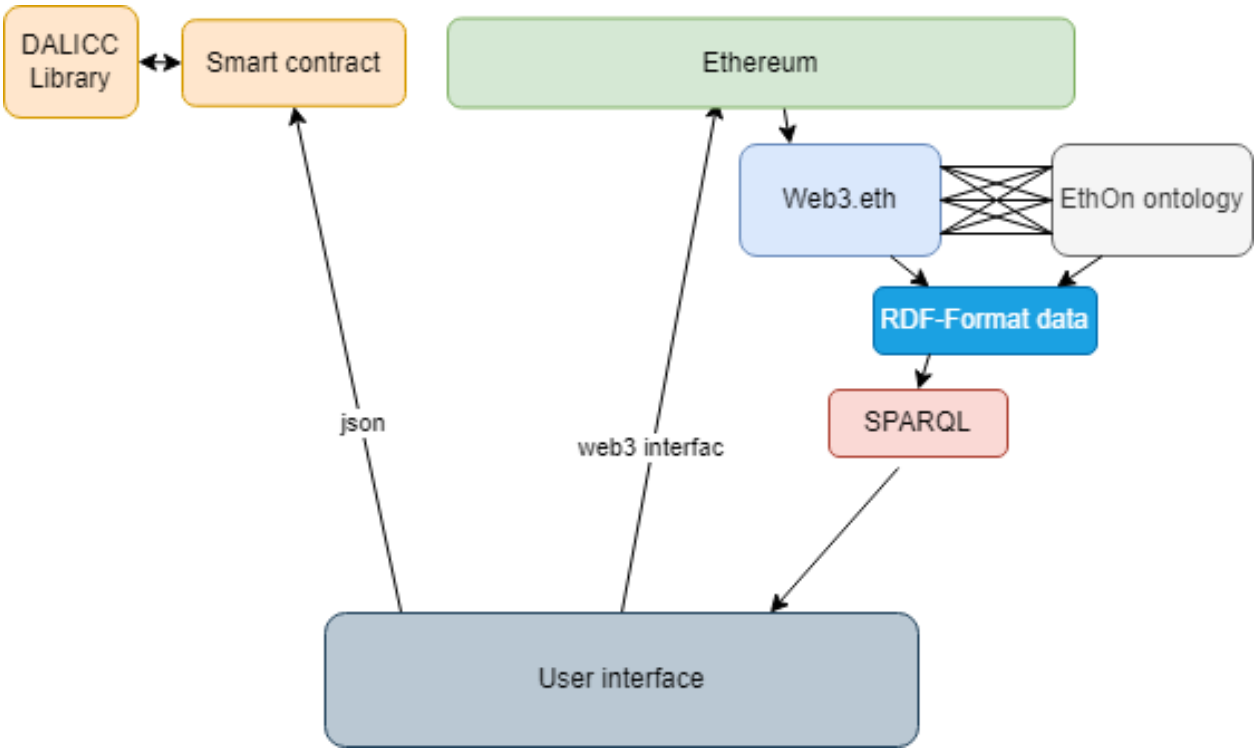


Figure 3.4: DApp Architecture

## 3.4 Implementation

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

### 3.4.1 smart contract logic

AS mentioned before, the backend code of this application on the ethereum network. in this section we will contemplate more on the each functionality of contract in this dApp.

- *Owned contract*
  This smart contract contains function to prevent non owner user to call function

and owner as constructor to be usable in every contract which is called to.
*onlyOwner*
This function 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 communicate with two other contracts to represent public interface of licensing system. It provide functions to licesing data or retrieve license information. It contains two other functions as follow:

- *LicenseManager contract*
  THis smart contract is responsible for saving address of license or create one for new file.

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

How Smart contract works?

- **Licensing Date** To license data, two parameters are needed. The first one is the hash of related data and second one the address of URI of the license. This smart contract is a *caller* contract *License Data*
  To license data, function *licenseData* is used which cecalred with two paramerters: first one is the hash of selected data, and the second one is the URI of the selected license. By pressing the 'License Data/retrieve license' on frontend, the function *licenseData* would be triggered and preform the steps as follow:
  This function checks if the target address is the zero-account, means the transaction create new contract:

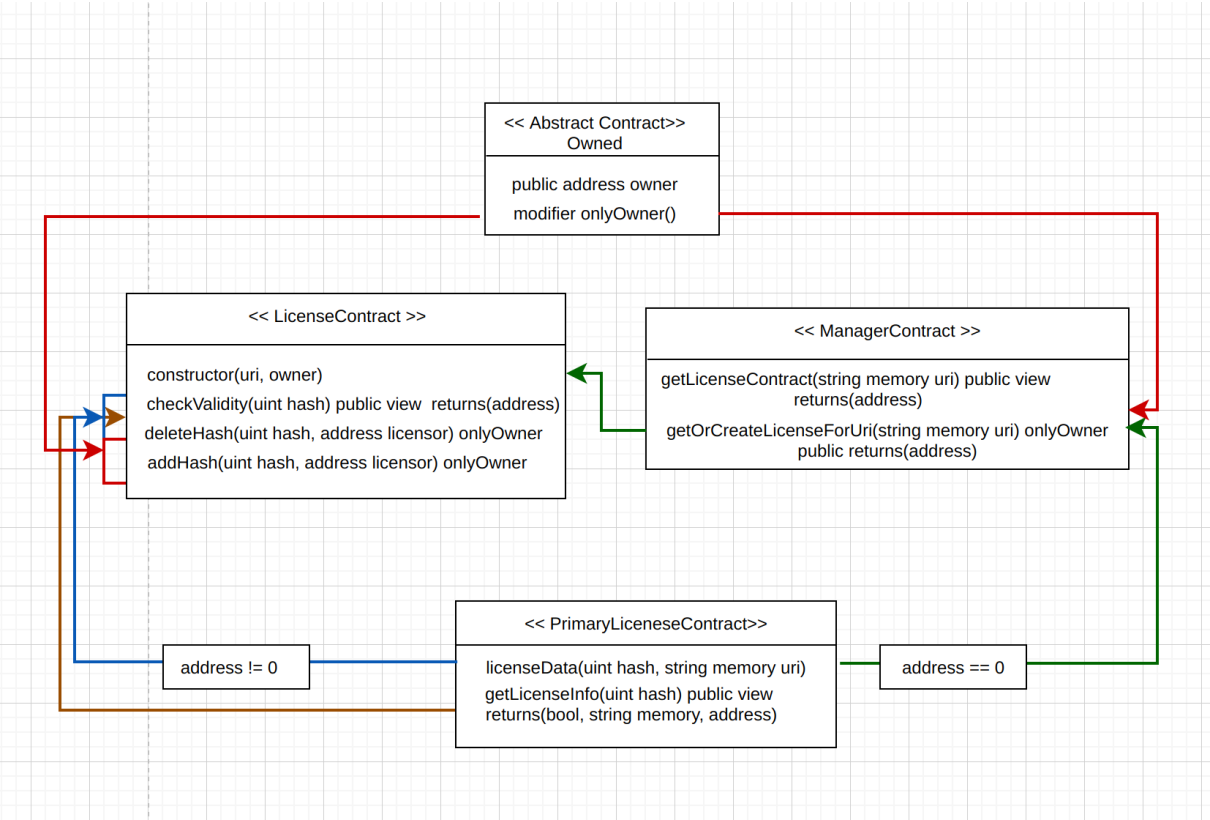


Figure 3.5: Smart contract visualization

if no, *deleteHash*,
otherwise *getOrCreateLicenseForUri*.

- *licenseData* function check if the specified hash 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 owner (function caller), means the caller of this function must be owner and the passed address should be licensor. Then the link between 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 represent this URI.
  **Definition** *getOrCreateLicenseForUri*: This function checks if the caller of

this function is owner, then exists a license contract for given URI.If so, the address of this license contract would returned. Otherwise, new licesn contarct will be created and the address of contract will be stored and returned. The caller address of the caller also will be used later as owner of this contract.

  - *addHash* function of the License contract is called with having two parameters: fist 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 the link between hash value and 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 licesnor with the same way like license data.

- **Retrieve license information** To retriev licnese finromation function *getLicenseInfo* is called having hash of data as parameter and check if there is some license information related to this hash, then is returned the address and , otherwise, return null address and string.

### 3.4.2 Forntend Workflow

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

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

- Check Your Data: In this step, user should check the selected data: if it is licensed beofre or not. HE/ she receives just message either confirmation 'License detected' or rejection 'NO license is detected', he or she should go further to obtain more information.

- License data / retrieve License information: Here, user receives the result of that last step, by pressing the hash value of selectd data `SHA3-256` is calculated and passed to smart contract using *web.js* interface:
  - License information of the selected file or data which user received the 'license detected' message from last step. TODO more about license information - Start licensing hie /her file, if he / she received 'No license detected' from last step. In order to license data, user get asked to log into `Metamask` to pass this account as address of licensor to smart contract. After afew second user should receive some transaction hash, if the transaction is done successful, otherwise he/she receive the error message on frontend.

- Send transaction: After receiving hash of transaction in frontend , user go further to get receipt of this licensing having all details about transaction. in order to have this receipt user send transcation receipt which is not readable to server for more processing on this raw result, coverting to RDF format data and make query it to produce more readable RDF-based-data.

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

# Chapter 4

# Conclusion

In this work, we attempted to analyze smart contract and their integration with semantic web data licesing. To achieve this goal, first we tried to analyze blockchain where smart contract reside on it as program code. Afterwards, we showed that how blockchain could be applied as computational paradigm for semantic web and linked data. Then we focused on bitcoin, Etheruem as main usage of blockahin and related vulnerabilities in Ethereum. Furthermore, we have presented the first initial effort to use this construct in useful ways by extending blockchain with semantic web in supply chain management. As the main goal of this work is to show ssemantic web principle can be use in making smarter smart contract and blockchain network. First, we started presenting our work done in ontology, Ethereum ontology (EthOn) describes blockchain structure and related information such as classes, object properties and etc. Later, we used this concept as RDF triple format which will feed by dataset and will lead into representing our data based on query.

# Appendix A

# Smart Contract code in solidity

```
pragma solidity ^0.4.18;

// construct contract that update and track goods throughout ShipmentTracking
// and automatically execute payment

contract GoodTracking {
uint[] conLoc; // array containing latitude & longitude
uint conTime;
uint conPayment;   // in token
address sender;
mapping(address => uint) balances;
mapping(string => good) goods;
mapping (address => uint) totalShipped; // total number of all shipments
mapping (address => uint) successShipped; // Shipment successfully are completed

// events to display messages when certain transactions are executed
event Success(
string _msg,
string packageId,
uint[] location,
uint timestamp,
address sender
);

event Payment(
string _msg,
address _from,
address _to,
uint _amount
);

event Failure(
```

```
string _msg
);

// function to amount of tokens from one account to another
function arrivedToken(address _from, address _to, uint _amount)
public returns (bool success) {
if (balances[_from] < _amount) {

Failure('Insufficient funds to send payment');
return false;

}
balances[_from] -= _amount;
balances[_to] += _amount;
Payment('Payment sent', _from, _to, _amount);
return true;
}

// function to show token balance
function getBalance(address _account) public returns (uint _balance) {

return balances[_account];
}

struct good{
string item;
uint  quantity;
uint[] location;
uint  timestamp;
address sender;
}

//function for displaying details of sent shipment

function sendGood(
string  packageId,
uint  _quanitity,
uint[]  _location,
string  _item,
uint _departure
)public returns(bool success){

goods[packageId].item = _item;
goods[packageId].quantity = _quanitity;
goods[packageId].location = _location;
Success('Item shipped', packageId,  _location,
_departure = now, msg.sender);
return true;
}
```

```
// function for displaying details of received shipment

function arrivedGood(
string _item,
string packageId,
uint _quantity,
uint[] _location,
uint _arrival
) public returns (bool success)  {

if (sha3(goods[packageId].item) == sha3(_item) && goods[packageId].quantity == _quant
Success('Item shipped', packageId, _location, _arrival = now, msg.sender);
}

if( _arrival <= _arrival + conTime && _location[0] == conLoc[0] &&
_location[1] == conLoc[1]){

arrivedToken(sender, goods[packageId].sender, conPayment);
} else {
Failure('Payment not triggered as criteria not met');
}

return true;
}
// Gets all the details for each shipment

function getDetails(string packageId)
public constant returns( string, uint, uint[],uint,  address)

{
return (
goods[packageId].item,
goods[packageId].quantity,
goods[packageId].location,
goods[packageId].timestamp,
goods[packageId].sender
);
}

// function to display number of successfully shipments and total shipments
function ckeckSuccess(address _sender) public returns (uint, uint)

    {
        return (successShipped[_sender], totalShipped[_sender]);
      }

}
```

# Appendix B

# Tripleize application

```
const fs = require('fs');
const parse = require('csv-parse');
const readline = require('readline');
const Handlebars = require('handlebars');
const Stream = require('stream');

fs.readFile('../templates/transactions.tmp', 'utf8', function (err,data) {
if (err) {
throw new Error(err);
} else {
rendering(data);
}

});

function rendering(source) {
const template = Handlebars.compile(source);
const readStream = fs.createReadStream('../data/input/transactions.csv');
//const writeStream = fs.createWriteStream( "../data/output/blocks.ttl", { encoding:
let rl = readline.createInterface({
input: readStream,
output: new Stream,
terminal: false,
historySize: 0
});

rl.on('line', function(line) {
// process line here
parse(line, (err, data) => {
if(err) {
console.log("Error: " + err);
throw new Error(err);
```

```
} else {
tripleize(data, template);
// writeStream.write(line);
}
});

}).on ('close', function() {
console.log('Good done!');
process.exit(0);
});

}

function tripleize (data, template){

let properties ={
tx_hash: (data[0][0]),
tx_block_number: data[0][1],
tx_block_header: data[0][2],
tx_block_timestamp: data[0][3],
tx_from: data[0][4],
tx_to: data[0][5],
tx_address: data[0][6],
tx_gas_used: data[0][8],
tx_value: data[0][9],
};

let result = template(properties);
var header = fs.readFileSync('../data/input/headers.ttl', 'utf8');
//console.log(header +result);
fs.writeFileSync('../data/output/output.ttl', header + result, { encoding: "utf8"});
}
```

# Bibliography


[Ada] Bc approach scm in bim environment. In Adam Fitria wijaya, T. H.-h. and Taysheng, J., editors, *Proceedings of the 24th International Conference of the Association for Computer-Aided Architectural Design Research in Asia*. ACM.

[2] Alfonso Panarello, Nachiket Tapas, G. M. F. L. and Puliafito, A. (2018). Blockchain and iot integration: A systematic survey.

[3] Allan Third, J. D. (2017). Linked data indexing of distributed ledgers. In *Companion Proceedings of the 26th International Conference on World Wide Web Companion*, volume 8.

[4] Antono Lucas Soares, A. L. A. and de Sousa, J. P. (1999). Distributed planning and control systems for thevirtual enterprise: organizational requirementsand development life-cycle. page 18.

[Buterin] Buterin, V. A next generation smart contract and decentralized application platform.

[6] Christidis, K. and Sevetsikioti, M. (2016). Blockchains and smart contracts forthe internet of things. page 12.

[7] Criback, K. (2018). Micro payment, viable technical platforms and models for a bank to provide payments on micro amounts.

[8] Dujak, D. and Sajter, D. (2018). *Blockchain Applications in Supply Chain*.

[9] Edvard Tijan, Sasha Aksentijevic, K. I. and Jardas, M. (2019). Blockchain technology implementation in logistics. page 13.

[10] Guido Geertsa, D. L. (2014). The eaglet ontology for highly visible supply chains. page 53.

[11] Guo, Y. and Liang, C. (2016). Blockchain application and outlook in the banking industry. page 12.

[12] Hamza Baqa, Nguyen B. Truongy, N. C. G. M. L. F. L. G. (2019). Semantic smart contracts for blockchain-based services in the internet of things. 5.

[13] Hanqing Wu, Jiannong Cao, Y. Y. C. L. T. S. J. B. T. Y. L. X. W. Y. D. (2019). Data management in supply chain using blockchain: Challenges and a case study. page 9.

[14] Ilya Grishchenko, M. M. and Schneidewind, C. (2018). *A Semantic Framework for the Security Analysis of Ethereum smart contracts*.

[15] Kim, H. M. and Laskowski, M. (2018). Towards an ontology-driven blockchain design for supply chain provenance.

[16] Kim, J.-S. and Shin, N. (2019). The impact of blockchain technology application on supply chain partnership and performance. page 17.

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

[18] Kouhizadeh, M. and Sarkis, J. (2018). Blockchain practices, potentials, and perspectives in greening supply chains. page 16.

[19] Law, A. (2018). Smart contracts and their application in supply chain management. page 89.

[20] Lin and Harding (2007). A manufacturing system engineering ontology model on the semantic web for inter-enterprise collaboration. page 32.

[21] Lin, A. M. W. and Madni, C. (2000). Ideon: An extensible ontology for designing, integrating, and managing collaborative distributed enterprises. page 14.

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

[23] Matthew English1, S. A. and Domingue2, J. (2016). Block chain technologies and the semantic web: A framework for symbiotic development. page 15.

[24] Md Nazmul Islam, Vinay Patil, S. K. (2018). On ic traceability via blockchain. page 4.

[25] Michele Ruta, Floriano Scioscia, S. I. G. C. A. P. and Sciascio, E. D. (2018). A blockchain infrastructure for the semantic web of things. page 12.

[26] Mike Uschold, Martin King, S. M. Y. Z. (1996). Enterprise ontology. page 57.

[27] Mirek Sopek, Przemyslaw Gradzki, W. K. D. K. R. T. R. T. (2018). Companion proceedings of the the web conference 2018. In *GraphChain âĂŞ A Distributed Database with Explicit Semantics and Chained RDF Graphs*, volume 8.

[28] Ozsu, T. and Valduriez, P. (1999). Principles of distributed database systems.

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

[30] Sharples, M. and Domingue, J. (2016). *Adaptive and Adaptable Learning*.

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

[32] Tonci Grubic, I.-S. F. (2010). Supply chain ontology: Review, analysis and synthesis. page 11.

[33] Ugarte, H. (2017). A more pragmatic web 3.0: Linked blockchain data. 15.

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

[35] Ureten1, S. and Ilter, K. (2007). Supply chain management ontology: Toward an ontology-based scm model.

[36] Wesley Egbertsen, Gerdinand Hardeman, M. v. d. H. G. v. d. K. and van Rijsewijk, A. (2016). Replacing paper contracts with ethereum smart contracts. 35.

[37] Wood, D. G. (2012). Ethereum: A secure decentralized generalized transaction ledger. page 32.

[38] WÃűhrer, M. and Zdun, U. (2018). Smart contracts: Security patterns in the ethereum ecosystem and solidity. 7.

[39] Yan Ye, Dong Yang, Z. J. L. T. (2008). An ontology-based architecture for implementing semantic integration of supply chain management. page 19.