



*Università degli Studi Di Urbino Carlo Bo*

Tesi di Laurea

*Corso di Informatica Applicata*

# **Implementation of a Smart Contract based System for Traceability in Agrifood**

Autore:  
Matteo Serafini  
Università degli Studi Di Urbino Carlo Bo  
[m.serafini12@campus.uniurb.it](mailto:m.serafini12@campus.uniurb.it)

Relatore:  
Stefano Ferretti  
Dip. di Scienze Pure ed Applicate  
Università degli Studi Di Urbino Carlo Bo  
[stefano.ferretti@uniurb.it](mailto:stefano.ferretti@uniurb.it)

Co-Relatore:  
Mirko Zichichi  
Law, Science and Technology Joint  
Doctorate  
[mirko.zichichi2@unibo.it](mailto:mirko.zichichi2@unibo.it)

Sessione Invernale 2021/2022

# Index

I.	Introduction	4
II.	Background	6
	a. Distributed Ledger Technology	6
	b. Decentralized Application	6
	c. Smart Contract	7
	d. Ethereum	7
	e. Scalability, Security, Decentralization, Consumption	8
	i. Blockchain limitations	9
	ii. DAG solution	11
	a. IOTA	12
	b. EVM	14
III.	Agrifood Supply Chain Scenario	15
IV.	Architecture	17
	a. Methodology	17
	b. UML diagram	18
V.	Implementation	19
	a. Development Tools	19
	i. Ganache	19
	ii. Metamask	20

b.	Smart Contracts in Solidity	21
i.	Producer	21
ii.	Resource	23
VI.	Deployment in IOTA 2.0	26
i.	Wasp node configuration	27
ii.	Wasp-cli configuration	28
iii.	Chain initialization	29
VII.	Evaluations	30
a.	Tests	31
b.	Results	33
VIII.	Conclusions	36
IX.	References	37

# Introduction

In recent years, with the steady advance of globalization, traceability has become recognised as an essential tool for guaranteeing food safety and food quality. Definition: Food traceability is “*the ability to follow the movement of a feed or food through specified stage(s) of production, processing and distribution*” [1].

In the Agrifood sector, ensuring food safety and security is a critical and necessary prerogative. A proper traceability system is needed in order to prevent and eventually trace any problem that may incur, be it a compromised product, a falsified origin or others.

A traceability system should be secure, transparent and openly viewable while respecting the privacy of its parties. With decentralized traceability systems in place, cutting down fraudulent documentation and minimizing risks to both producers and consumers, fairer trading, people’s and environment’s wellbeing can be assured more easily [2].

Distributed ledger technologies (DLT), due to the inherent trust and inalterability they provide, are becoming a first choice to implement such systems [3][4][5][6]. The mainstream DLT often are blockchain technologies like Bitcoin and Ethereum. Another type of distributed ledger technology is based directed acyclic graph (DAG) and offers a more scalable and environmentally friendly solution to the blockchains. An example of a DAG based technology is IOTA [7].

In this paper we will briefly describe the differences in the model, the architecture, the development and deployment process of Smart Contracts with the Ethereum[8] and IOTA platforms; as well as focusing on the feasibility of Smart Contract employment for secure, transparent and

available supply chain food traceability systems. In the agri food industry traceable assets all have a location of origin, they are at first a primary resource and later they are transformed into a product, zero or more times, before reaching the end customer; we will later analyze the actors, entities and events that take part in these stages of the supply chain.

The thesis is structured as such:

- Introduction
- In the second chapter we describe and explain the meaning of the technical terms we will encounter in this paper as well as a quick presentation of the architecture of the Ethereum and IOTA Distributed Ledger Technologies.
- In the third chapter we analyze Agrifood Supply Chain Scenarios in general
- In the fourth chapter we describe the architecture of the sample DApp for agri food traceability
- In the fifth chapter we will dive into a sample implementation of a distributed supply chain food traceability system through the use of smart-contracts written in the Solidity programming language.
- In the sixth chapter we will describe the attempt to port such a system from the Ethereum platform to the IOTA 2.0 platform [9].
- The seventh chapter contains tests and evaluation of the results obtained.
- The eighth chapter contains the conclusions found, some consideration of the difficulties encountered and possible subsequent developments.

Keywords — Smart-Contract, Blockchain, Tangle, DApp, Agrifood Supply Chain Traceability, Ethereum, IOTA, Solidity

# Background

## **Distributed Ledger Technology**

Distributed Ledger Technology (DLT) is a technology providing an open environment with no singular authority for registering, sharing, and synchronizing transactions on digital devices. DLT executes this function making use of several computer science disciplines such as distributed systems, cryptography, data structures, or consensus algorithms. It supplies many highly desirable features such as decentralization, openness, immutability, transparency, traceability, security, availability, etc.).

## **Decentralized Application**

A decentralized application, or DApp, is a computer application that is stored and executed on a distributed system, that is on a network of nodes, with no node taking over as a supervisor. DApps can operate autonomously, through the use of Smart Contracts. The decentralized and distributed nature of DLT on peer-to-peer networks allows for it to be transparent, deterministic and redundant, enhancing security openness and availability.

## **Smart Contract**

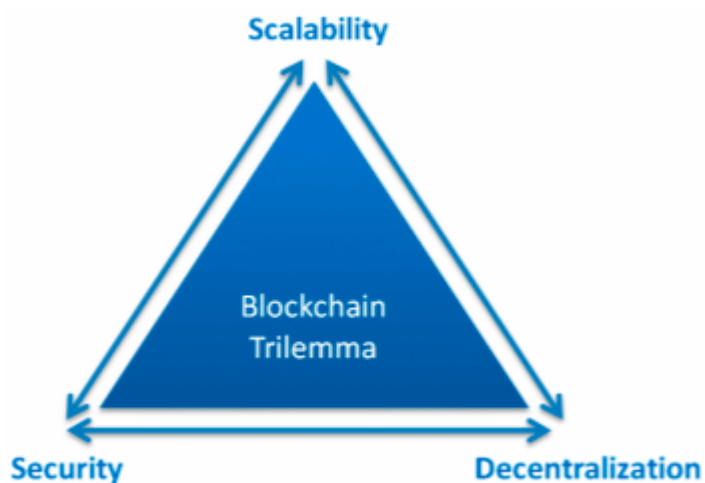
The concept of Smart Contract was first proposed by Nick Szabo in 1995 [10]: *“a smart contract is a set of commitments defined in digital form, including the agreement that the participants in the contract can implement these commitments”*. Smart Contract is a computer protocol designed to verify or execute contracts in an information-based manner. It allows trusted transactions without a third party and these transactions are traceable and irreversible, Smart Contracts are immutable as no one can tamper with the code of the contract, and distributed; because of their storage inside of a distributed system Smart contracts have been developed and used in several fields [12]; their purpose is to provide better security, transparency and reduce other transaction costs related to traditional contracts. In the case of this paper I focus on agri-food supply chains Smart Contracts.

## **Ethereum**

Blockchain is the most famous and stable example of a DLT. In the blockchain as each transaction occurs, it is recorded as a “block” of data and each block is connected to the ones before and after it, similarly to a linked list. The blocks store the exact time and sequence of transactions, the transactions are blocked together in an irreversible chain, this prevents blocks from being altered or being inserted between two already existing ones. Each additional block strengthens the verification of the previous one and as a consequence of the entire blockchain. This makes the blockchain tamper-evident, delivering the key strength of immutability. Ethereum proposes to utilize blockchain technology not only for maintaining a decentralized payment network but also for storing computer code that can be used to power tamper-evident decentralized financial contracts and applications. In Ethereum there is a mandatory fee for each transaction regardless of the actual price of the assets being exchanged. This is due to the architecture of the blockchain and its validating algorithms. This fee in jergon is called gas.

## Scalability, Security, Decentralization, Consumption

Scalability in DLTs can be described as the ability to support growth in the form of the frequency of transactions. This means that a highly scalable blockchain would be able to adapt or at least not suffer from an increase of its adoption. The blockchain trilemma tells us that greater scalability is possible, but security, decentralization, or both will suffer as a consequence [13]. Scalability is a strong prerogative for blockchain networks to compete with other modern or even legacy centralized platforms.

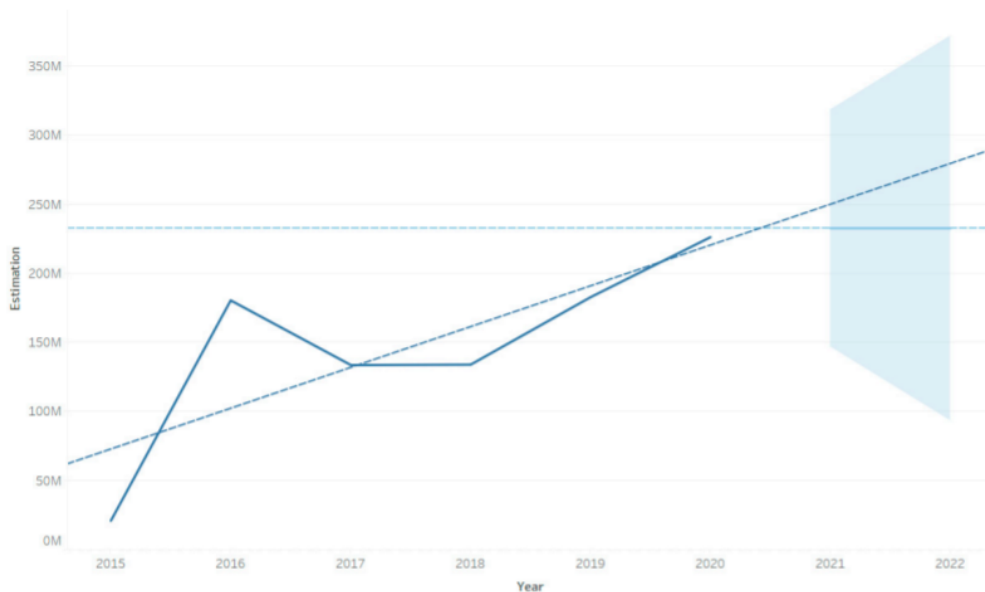


### *Blockchain Limitations*

For every new transaction, each node adds information regarding the transaction in the ledger, consequently the ever growing transaction history could overload the system. Blockchains could also experience issues in terms of hardware limitation or energy consumption. The Proof of Work algorithm is by nature not resource friendly, moreover as the blockchain network expands further, it is difficult to set up and maintain the



hardware required for operating nodes and validating transactions on long chains. For example, as opposed to when it was first announced in 2009, nowadays to successfully mine Bitcoin blocks it takes expensive and high energy consuming hardware (ASIC or GPU).



**Figure 1.** Estimated energy consumption from Bitcoin mining (annualized TWh). Source: Own elaboration using Tableau Desktop Professional Edition and data extracted in May 2021 from the University of Cambridge.

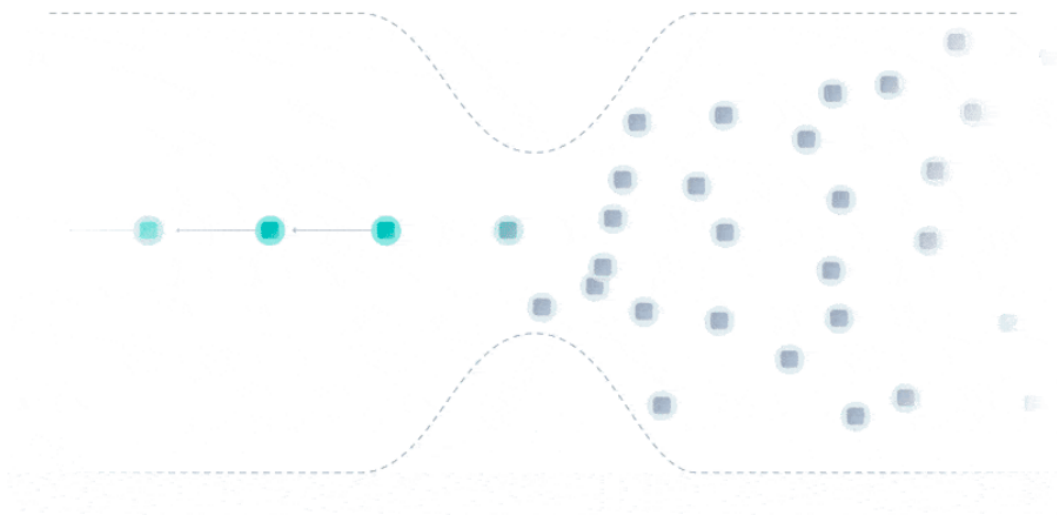
[14]

Another critical factor is the high transaction fees. While in the case of the exchange of high value assets the transaction fees can become negligible, often in the use case of IoT devices or in general whenever tiny and frequent transactions are needed, high value or even any transaction fee would render the system unfeasible.

Transactions in the blockchain have to wait for relatively long periods of time for the validation process, also considering the number of transactions in the queue. In the case of Bitcoin almost 10 minutes are needed for building a new block. A much better rate is offered by Ethereum with the possibility of 15 seconds for building a block [15].

Another reason for resource wastage in the blockchain is the orphan blocks, an orphan block is a block that has been solved within the blockchain network but was not accepted due to a lag within the network itself.

Example of a Blockchain bottleneck:



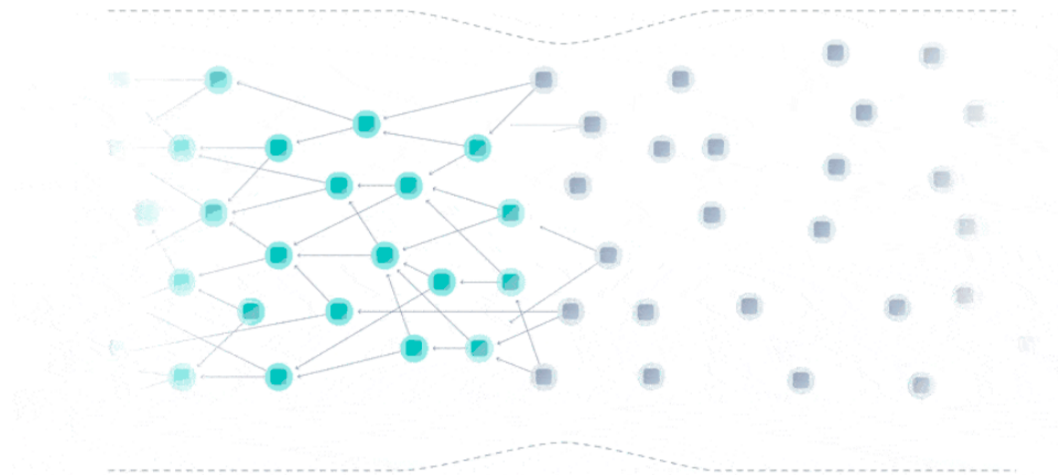
[16]

The blocks are validated relatively slowly and one at a time often forming a queue. Due to block size and block production time limitations heavier congestions can also occur.

### *DAG solution*

One approach to overcome the scalability issues of blockchains is the use of the directed acyclic graph (DAG) data structure. Blockchain and DAGs have some similarities. Both allow users to obtain eventual consensus over the state of a ledger, in a decentralized manner. However, they do differ in their underlying mechanisms, and more importantly, have some key differences in their scaling properties and their potential use cases. In DAG based DLTs unlike in blockchain systems, orphan blocks can be merged back into the system, and are therefore not a waste of resources. This is one of the major advantages of DAGs. Another advantage is the possibility for any user to validate new blocks and attach them to the DAG, while a blockchain transaction must be included in a block strictly by a block producer.

DAG DLT solution to the bottleneck:

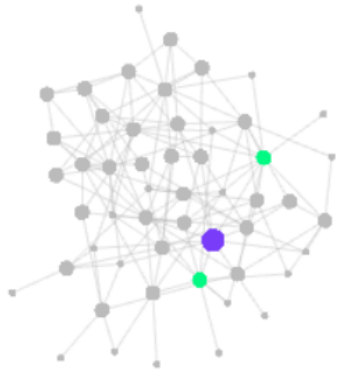


[16]

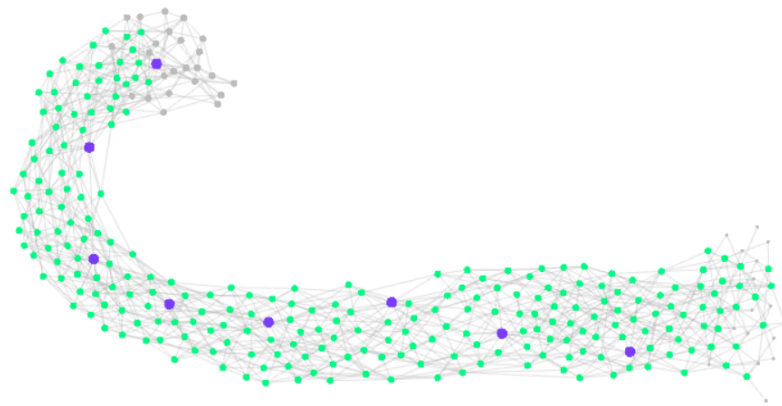
## IOTA

IOTA is a DLT that differs from Ethereum as it is not based on the blockchain but on DAG. IOTA's DAG is called Tangle. From IOTA's wiki *"In the Tangle, there are no block producers, and therefore every user is free to issue new transactions and attach them on different Tangle parts without an entity that acts as middlemen. The Tangle is not a single chain of blocks that follow each other. It is a network of parallel processed transactions (so-called Tips). These parallel transactions form the "front" of the Tangle and offer many different points for newly issued transactions to be attached, which dramatically speeds up the processing of transactions."*[16]. In IOTA's versions previous to the 2.0 there is a supervising node, The Coordinator, that sends signed messages called milestones that nodes trust and use to confirm messages. Messages in the Tangle are considered for confirmation only when they are referenced by a milestone that nodes have validated. To ensure that new messages always have a chance of being confirmed, the Coordinator sends indexed signed milestones regularly. This way ensures that nodes can compare the indexes of their milestones to check whether they are synchronized with the rest of the network. This approach is however temporary as it makes full decentralization of the system not possible. In IOTA v2.0 the coordinator has been removed in accordance with IOTA Foundation's vision of a fully decentralized, feeless and highly scalable distributed ledger technology.

Example of a growing Tangle (DevNet IOTA v2.0):

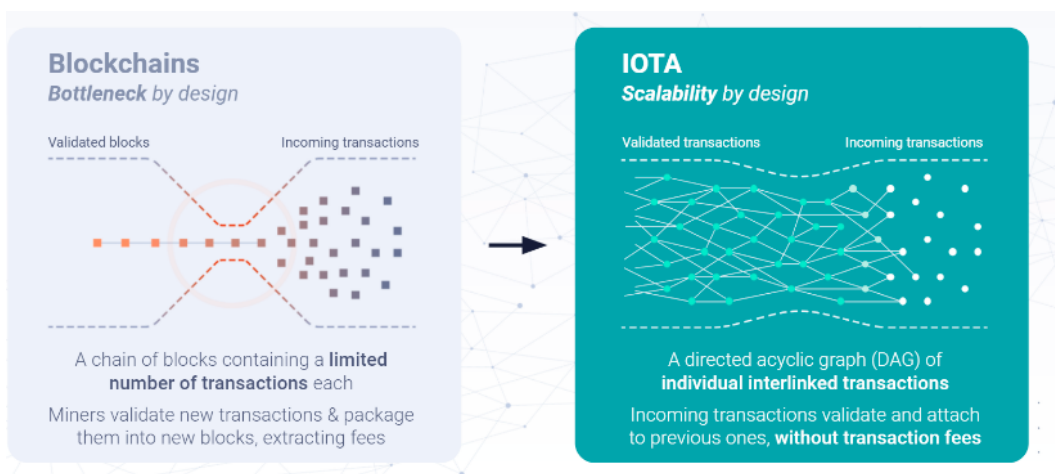


[17]



[17]

Blockchain vs. Tangle:



[16]

## **EVM**

EVM which stands for Ethereum Virtual Machine is currently the environment which most Smart Contract implementations are running on. Solidity is the programming language of choice with EVMs, and has been created for this specific purpose. The main benefit of using EVM/Solidity in the IOTA platform is simply the sheer number of resources available from it from years of development and experimentation on Ethereum. There are many articles, tutorials, examples and tools available for EVM/Solidity, and the IOTA Smart Contracts (ISCP) implementation is fully compatible with them. Any existing SCs that run with Ethereum will probably need no (or very minimal) changes to function on IOTA Smart Contracts platform [16]. In The IOTA platform, an EVM based chain runs as an IOTA SC, so any internal EVM chain from the point of view of the ISCP is just another Smart Contract. Because of this, it is possible to run both Wasm based Smart Contracts and an EVM chain in a single ISCP. In the wasp-cli package tools it is offered an EVM compatible JSON-RPC server, which allows existing tools like metamask to connect to these EVM Chains. To deploy SCs to a new EVM chain is as easy as pointing to the address of the JSON-RPC gateway.

# Agrifood Supply Chain Scenario

In the agri food production domain, we can assume that that virtually all supply chain systems start from one or more primary sources (soil, herd, beehives, lake, sea, etc.), through a set of events they produce a primitive resource (harvest, milk, raw honey, fishes, etc.), then this resource is transformed to a product, zero or possibly several times.

Configuring such complicated systems can be trivial if they are not targeted for a specific purpose, product, legislation system, etc.

As mentioned in this [19 ] paper:

\_The main Actors in such types of systems could be:

- Administrator/Owner
- Producer
- Supplier
- Transformer
- Wholesaler
- Retailer
- End Customer
- Certification Authority
- Professional
- Analysis Lab
- Warehouse
- Device

\_The main entities:

- Address Catalog
- Producer
- Productive Resource

- Product
- Token
- Notarization document

\_The main Events:

- Transformation events
  - Product Merging
  - Product Splitting
  - Product Transformation
- Documentation events
  - Asseveration
  - Token creation
  - Data Registration
  - Notarization Event

The Owner/Administrator main tasks are to oversee the entire system and manage the collection of different actors in the system.

The actors have the responsibility to manage their own collection of resources. Producer and Transformer can for example transform an asset from a primary resource to a final product, or they can transfer the asset to another owner. Actors like the Certifier are not able to hold or transform assets, instead they can only generate events associated with the target resource. The events in the chain are mainly associated with the resource itself since they are the traceable asset in the system. In general, the main data to store in any agrifood system are therefore the catalog of actors managed by the owner, the catalog of resources associated with each actor and the catalog of authorized actors associated with each resource, each element with its respective attributes and state; finally one or more collections of events to be able to retrace the entire supply chain.



# Architecture

Solidity is the language of choice for the implementation of the sample DApp, it is the standard language running on EVMs and it is Object Oriented. Therefore the approach used in the engineering of the DApp is also Object Oriented.

For the sake of simplicity the sample DApp will take in account a much less complex system:

\_Owner of the chain: manages address catalog of producers.

\_Producers: as holders of resources and able to execute transformations

\_Resources: will have a list of authorized parties to prevent fraudulent actions

\_Generic Events : any type of event will be generalized in this one.

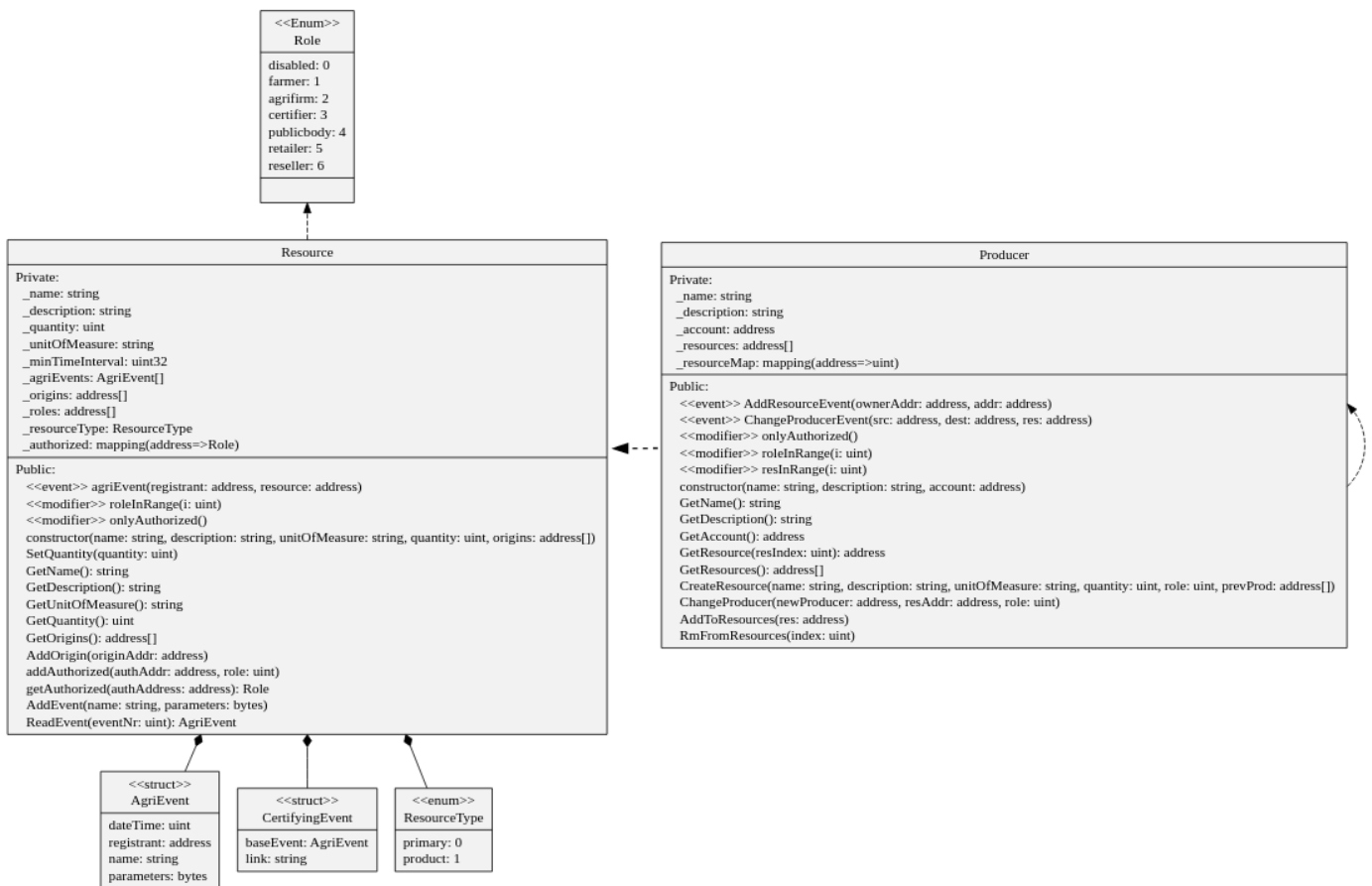
The owner will be the managing account of the chain so it won't need a SC to be represented.

The SCs will be used to represent Producers and Resources.

The Generic Events will be represented as a property of each Resource

We modeled the two Smart Contracts as shown in the following image. The Producer SC in its methods can reference other instances of itself as well as the Resource SCs. We generated a custom complex type to represent agri events, we will describe it in the implementation section.

### UML Diagram

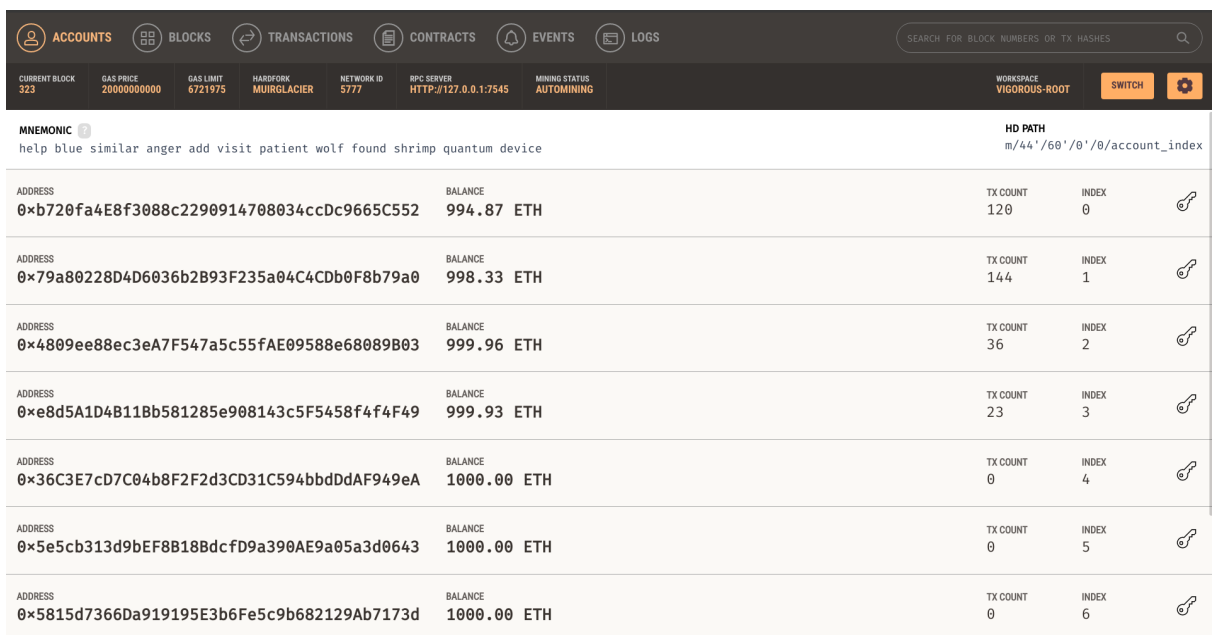


# Implementation

## Development Tools

### Ganache

Ganache allows to spawn a local blockchain which is used during the development of decentralized applications. It provides an easily accessible and tweakable development environment for the testing of Smart Contracts, including a variable number of mock accounts and their respective balances in Ethers.

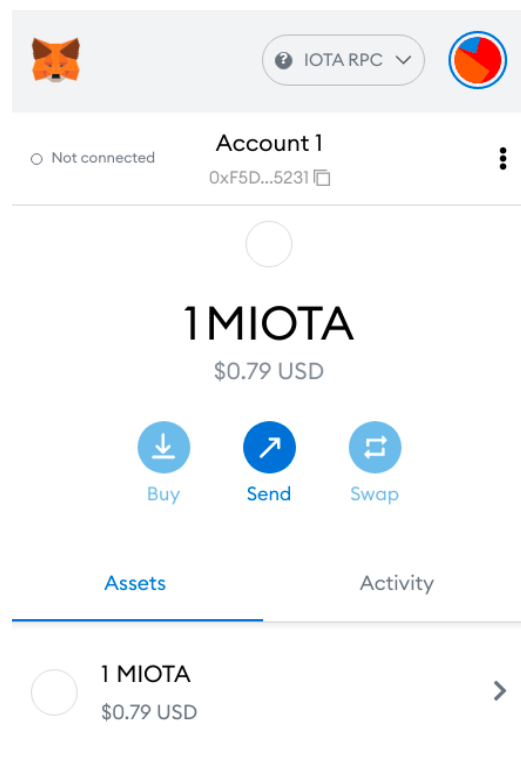


The screenshot shows the Ganache web interface. At the top, there are navigation tabs: ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below the tabs, there is a search bar and a status bar with various metrics like CURRENT BLOCK, GAS PRICE, GAS LIMIT, HARDFORK, NETWORK ID, RPC SERVER, and MINING STATUS. The main content area displays a list of accounts with their mnemonics, HD paths, addresses, balances, transaction counts, and indices.

MNEMONIC	HD PATH
help blue similar anger add visit patient wolf found shrimp quantum device	m/44'/60'/0'/0'/0/account_index
ADDRESS: 0xb720fa4E8f3088c2290914708034ccDc9665C552	BALANCE: 994.87 ETH
	TX COUNT: 120
	INDEX: 0
ADDRESS: 0x79a80228D4D6036b2B93F235a04C4Cb0F8b79a0	BALANCE: 998.33 ETH
	TX COUNT: 144
	INDEX: 1
ADDRESS: 0x4809ee88ec3eA7F547a5c55fAE09588e68089B03	BALANCE: 999.96 ETH
	TX COUNT: 36
	INDEX: 2
ADDRESS: 0xe8d5A1D4B11Bb581285e908143c5F5458f4f4F49	BALANCE: 999.93 ETH
	TX COUNT: 23
	INDEX: 3
ADDRESS: 0x36C3E7cD7C04b8F2F2d3CD31C594bbDdAF949eA	BALANCE: 1000.00 ETH
	TX COUNT: 0
	INDEX: 4
ADDRESS: 0x5e5cb313d9bEF8B18BdcfD9a390AE9a05a3d0643	BALANCE: 1000.00 ETH
	TX COUNT: 0
	INDEX: 5
ADDRESS: 0x5815d7366Da919195E3b6Fe5c9b682129Ab7173d	BALANCE: 1000.00 ETH
	TX COUNT: 0
	INDEX: 6

## Metamask

Metamask can be used in real life scenarios to interact with one's cryptocurrency wallets. However it is useful also in the development and testing phase of DApps, for example it makes it easy to create mock accounts which have been used for the IOTA Smart Contracts testing phase or even to be connected to Ganache. It is employed by users to manage their own accounts and the respective private keys, this allows easy signing of transactions to send to the nodes of the DLT.



## Smart Contracts in Solidity

Here is an example of Solidity source code used to implement the Producer and Resource smart contract. The OpenZeppelin toolkit is used to aid in the development, compilation, deployment and interaction with Smart Contracts [19]; in this case the Ownable object has been used for simplifying the implementation of ownership of a Smart Contract and the library Clones to enable a cheaper creation of Resource Smart Contracts through the use of assembly code.

This is a portion of the code for the Producer SC including several attributes to describe it; the `__account` attribute identifies the wallet address associated with the SC instance; `__resources` and `__resourceMap` are both used to manage the resources catalog for each instance; the constructor inherits from the Ownable object so that when an instance is deployed the address that is calling the deployment will be identified as owner.

The most interesting method is ChangeProducer :

Input: address - identifies the new Producer SC,  
address - identifies the Resource SC,  
uint - identifies the role of the new producer

Execution: get the instance of Resource and Producer SC,  
get wallet account of new Producer SC,  
manipulate producers's resources catalogs,  
change resource ownership

Output: void

```
import "../Resource.sol";
import "@openzeppelin/contracts/proxy/Clones.sol";
contract Producer is Ownable {
    string private _name;
    string private _description;
    address private _account;
    address[] private _resources;
    mapping (address => uint) private _resourceMap;
    event AddResourceEvent (address indexed ownerAddr,
                           address indexed addr);
```

```

event ChangeProducerEvent ( address indexed src,
                            address indexed dest,
                            address indexed res);

constructor (
    string memory name,
    string memory description,
    address account
) Ownable() {
    _name = name;
    _description = description;
    _account = account;
}

modifier onlyAuthorized() {
    require(msg.sender == owner() || msg.sender == _account,
        "ERROR only authorized");

    _;
}

function CreateResource (
    string memory name,
    string memory description,
    string memory unitOfMeasure,
    uint quantity,
    uint role,
    address[] memory prevProd
) public onlyAuthorized roleInRange(role) {
    Resource res = new Resource(name, description, unitOfMeasure,
quantity, prevProd);
    res.addAuthorized(_account, role);
    AddToResources(address(res));
}

function ChangeProducer (
    address newProducer,
    address resAddr,
    uint role
) public onlyAuthorized roleInRange(role) {
    Resource res = Resource(resAddr);
    Producer prod = Producer(newProducer);
    address prodAccont = prod.GetAccount();
    res.addAuthorized(prodAccont, role);
    res.addAuthorized(_account, 0);
    res.transferOwnership(newProducer);
    _resources[_resourceMap[resAddr]] = address(0);
    emit ChangeProducerEvent(address(this), newProducer, resAddr);
}

modifier roleInRange(uint i) {
    require(i >= 0 && i <= uint(type(Role).max),
        "ERROR role number out of range");

    _;
}

```

```
modifier resInRange(uint i) {
    require(i >= 0 && i < _resources.length,
           "ERROR resource number out of range");
}
-;
function AddToResources(address res)
public onlyAuthorized {
    _resources.push(res);
    emit AddResourceEvent(address(this), address(res));
}
function RmFromResources(uint index)
public onlyAuthorized {
    _resources[index] = address(0);
}
```

This is a portion of the code for the Resource SC including several attributes to describe it; to represent the different types of resources (primary or product), and the different types of actors (farmer, agrifirm, certifier, publicBody, retailer, reseller) we used an Enum data type; for the agriEvents array we generated a custom data type, using the `struct` keyword we were able to compose 4 base types into a complex one: `AgriEvent`; we use it in the method `addEvent(...)` making it easy to pass all needed the parameters as one, we then save this object to the contract storage appending it to the events array aforementioned.

```
import "@openzeppelin/contracts/access/Ownable.sol";
enum Role {
    disabled,
    farmer,
    agrifirm,
    certifier,
    publicbody,
    retailer,
    reseller
}
contract Resource is Ownable {
    string      private _name;
    string      private _description;
    uint        private _quantity;
    string      private _unitOfMeasure;
    uint32      private _minTimeInterval;
    AgriEvent[] private _agriEvents;
    address[]   private _origins;
    address[]   private _roles;
    ResourceType private _resourceType;
    mapping ( address => Role ) private _authorized;
    enum ResourceType {
        primary,
        product
    }
    struct AgriEvent {
        uint dateTime;
        address registrant;
        string name;
        bytes parameters;
    }
}
```



```

event agriEvent (address indexed registrant, address indexed
resource);
constructor (
    string memory name,
    string memory description,
    string memory unitOfMeasure,
    uint quantity,
    address[] memory origins
) Ownable() {
    _name = name;
    _description = description;
    _unitOfMeasure = unitOfMeasure;
    _quantity = quantity;
    _origins = origins;
    if (origins.length == 0){
        _resourceType = ResourceType.primary;
    } else {
        _resourceType = ResourceType.product;
    }
}
modifier eventInRange(uint eventNr) {
    require(eventNr >= 0 && eventNr < _agriEvents.length,
        "ERROR: The event number must be within range!");
    _;
}
modifier onlyAuthorized() {
    require(msg.sender == owner() || _authorized[msg.sender] !=
Role.disabled,
        "ERROR: not authorized");
    _;
}
function AddEvent (
    string memory name,
    bytes memory parameters
) onlyAuthorized public {
    _agriEvents.push(AgriEvent(block.timestamp, msg.sender, name,
parameters));
    emit agriEvent( msg.sender, address(this));
}
function SetQuantity(uint quantity)
    onlyAuthorized public {
    require(quantity >= 0, "Cannot be negative");
    _quantity = quantity;
}
function AddOrigin (
    address originAddr
) onlyAuthorized public {
    _origins.push(originAddr);
}

```

```

function addAuthorized (
    address authAddr,
    uint role
) onlyAuthorized roleInRange(role) public {
    _authorized[authAddr] = Role(role);
}
function ReadEvent(
    uint eventNr
) public view eventInRange(eventNr)
returns ( AgriEvent memory) {
    return _agriEvents[eventNr];
}

```

Solidity was born specifically to write Smart Contracts therefore it has some unique functionalities compared to other OOP languages. For instance, `modifier` is a special type of function used to allow or prevent the execution of another function in accordance with a certain condition: `onlyAuthorized` is an example of it; `address` is a data type specific of EVMs, it is essentially a string unique to the network that identifies a Smart Contract or a wallet account, furthermore it can send and receive values with the use of the keyword `address payable`. `event` identifies a type of data storage that is permanent in the blockchain, the difference with the `storage` data type, the main type of permanent storage in EVMs, is that the first it is lightweight and only accessible externally; so whatever it is stored through the use of the `event` keyword it is not reusable by the Smart Contract logic but it is used for logging purposes [20].

## Deployment in IOTA 2.0

The IOTA Smart Contract Platform natively supports the Wasm Virtual Machine to run SCs written in Rust, Golang or Typescript, however it is not yet a tested and proven technology as opposed to the EVM; The EVM has already undergone years of development on Ethereum and we can now benefit from a stable and resourceful working environment. Since IOTA v2 provides an Ethereum Virtual Machine layer we can use the Smart Contracts that have been previously used on the Ethereum platform unchanged, as well as the testing file, apart from very few initial configurations. The IOTA foundation has built Wasp to enable the use of Smart Contracts. Wasp is the node software that enables Smart Contracts validation as a part of a committee. The committee of validators is formed by the connection of multiple wasp nodes. When consensus is reached on a virtual machine state-change, that transaction is anchored to the IOTA tangle, making it immutable. The current experimental implementation of IOTA SCs (IOTA v2.0), is deployed on a fully decentralized development network, powered by GoShimmer nodes [16].

We need to set up a Wasp development node and Goshimmer network and later we need to start a JsonRpc service for that network. The easiest choices now are to either utilize the development network already provided in the wasp repository or to install wasp and wasp-cli commands and point to the public goshimmer development network.

### *Wasp node configuration*

```
"webapi": {
  "bindAddress": "127.0.0.1:9090"
},
"dashboard": {
  "auth": {
    "scheme": "basic",
    "username": "wasp",
    "password": "wasp"
  },
  "bindAddress": "127.0.0.1:7000"
},
"peering":{
  "port": 4000,
  "netid": "127.0.0.1:4000"
},
"nodeconn": {
  "address": "goshimmer.sc.iota.org:5000"
},
"nanomsg":{
  "port": 5550
}
}
```

The web API interface allows access to functionality of the node software via exposed HTTP endpoints.

The IOTA network is a distributed network which uses a gossip protocol to broadcast data among IOTA nodes. [] To participate in the network, each node has to establish a secure connection to the other nodes and mutually exchange messages. Each node can be uniquely identified by a peer identity.

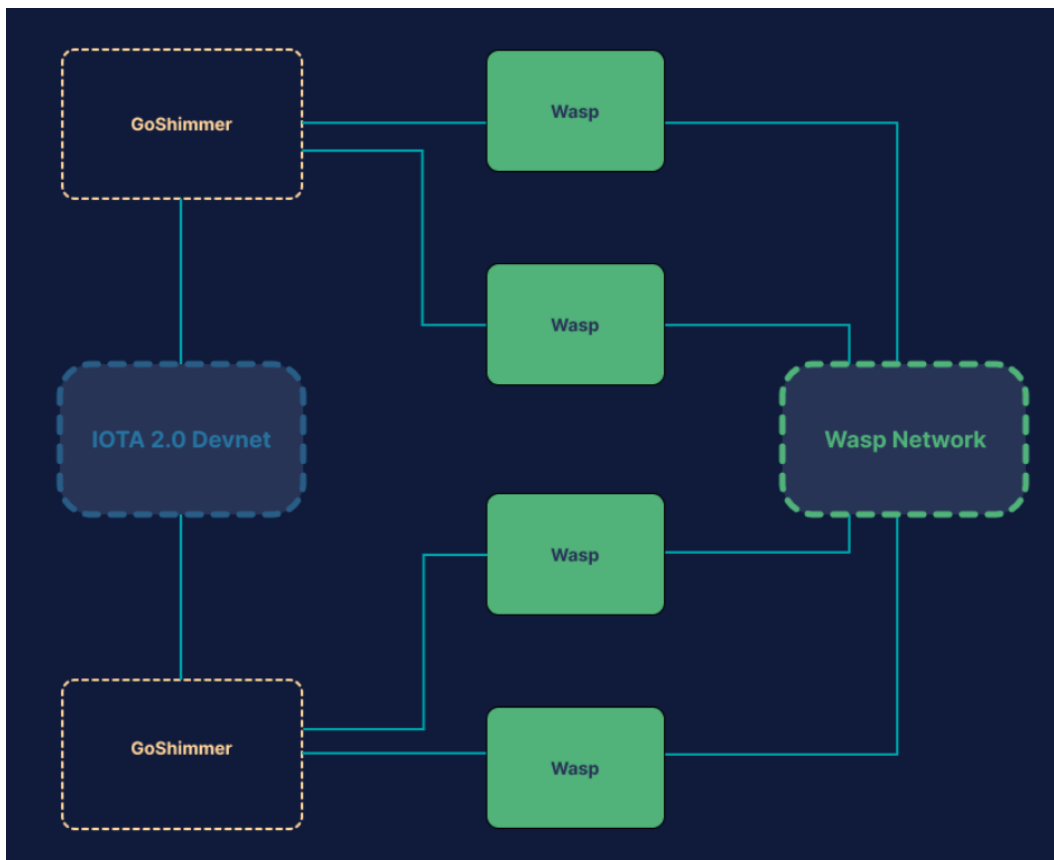
Each Wasp node publishes important events via a Nanomsg message stream. Nanomsg clients can subscribe to the message stream.

Nodeconn defines the TCP port exposed to TXStream for clients to connect and subscribe to real-time notifications on confirmed transactions targeted to specific addresses. It is used by Wasp nodes to be notified about incoming requests and updates on a given chain.

### Wasp-cli configuration

```
"goshimmer": {  
  "api": "https://api.goshimmer.sc.iota.org",  
  "faucetpowtarget": -1  
},  
"wasp": {  
  "0": {  
    "api": "127.0.0.1:9090",  
    "nanomsg": "127.0.0.1:5550",  
    "peering": "127.0.0.1:4000"  
  }  
}
```

Here we define the access points of the Wasp node, and Goshimmer network. We need to be careful to define the same ip and ports as in the wasp server configuration file. Here the Goshimmer API endpoint is the public one provided by the IOTA foundation.



[16]

## Wasp-cli init script

```
wasp-cli init
wasp-cli peering info
PUBKEY=$(wasp-cli peering info | grep PubKey | cut -d':' -f2)
NETID=$(wasp-cli peering info | grep NetID | cut -d':' -f2)
wasp-cli peering trust $PUBKEY $NETID:4000
wasp-cli request-funds
wasp-cli chain deploy --committee=0 --quorum=1 --chain=agrievm
--description="EVM Chain for agrifood traceability"
wasp-cli chain deposit IOTA:10000
wasp-cli chain evm deploy -a agrievm --alloc
$OWNER_ACCOUNT:$VALUE,$PRODUCER1_ACCOUNT:$VALUE,$PRODUCER2_ACCOUNT
:$VALUE,$CERTIFIER_ACCOUNT:$VALUE
wasp-cli chain evm jsonrpc --chainid 1074 --account $OWNERKEY
```

- Here the code communicates the peering details to the network
- Starts a IOTA chain with a consensus quorum of 1, so that transaction can be easily validated in a development environment with only one node
- Deposits 10000 IOTA to the aforementioned chain
- Deploys an EVM chain in the form of a IOTA native SmartContract while allocating four users with a certain balance to it.
- Starts a local Json Rpc service on port 8545 with a unique chain Id of 1074 and allocating the manager account

# Evaluation

## Tests

### Producer Smart Contract Deployment Test

```
before("Deploy contracts", async () => {
  console.log("Create First Producer");
  let factory = new ethers.ContractFactory( ProducerJSON.abi ,
  ProducerJSON.bytecode, ownerAccount);
  let producer1 = await factory.deploy("one", "one desc",
  p1Ac.address);
  await producer1.deployTransaction.wait();
  p1Addr = producer1.address;
  console.log("Create Second Producer");
  let producer2 = await factory.deploy("two", "two desc",
  p2Ac.address);
  await producer2.deployTransaction.wait();
  p2Addr = producer2.address;
});
```

Here the code performs the first deployment of two Producer SCs on the chain, then simply stores their addresses in two variables for later use.

### Create Resource from Producer Smart Contract Test

```
it("create first resource", async () => {
  console.log("Create First resource from first producer");
  let p1 = new ethers.Contract(p1Addr, ProducerJSON.abi, p1Ac);
  let tx = await p1.CreateResource("first resource", "some desc ",
  "uOM", 100, 1, []);
  let receipt = await tx.wait();
  res0Addr = await p1.GetResource(0);
});
```

Here the code calls the Producer SC to perform the deployment of its first Resource SC, then simply stores its addresses in a variable for later use.

### Add Authorized Account to Resource Smart Contract Test

```
it("Add certifier to authorized accounts", async () => {
  let res0 = new ethers.Contract(res0Addr, ResourceJSON.abi,
p1Ac);
  let tx = await res0.addAuthorized(certAc.address, 3);
  let receipt = tx.wait();
});
```

Here the owner of the chain is able to authorize the address of a Wallet account to call certain operations on the target resource, in this case the account is registered as a certifier who will be able to add agriEvents associated to the resource.

### Change Producer Smart Contract Test

```
it("-----transfer resource from one producer to another-----",
async () => {
  let p1 = new ethers.Contract(p1Addr, ProducerJSON.abi, p1Ac);
  let p2 = new ethers.Contract(p2Addr, ProducerJSON.abi, p2Ac);
  let resAddress = await p1.GetResource(0);
  let resourceP1 = new ethers.Contract(resAddress,
ResourceJSON.abi, p1Ac);
  let initialQ = await resourceP1.GetQuantity();
  let tx0 = await p1.ChangeProducer(p2.address,
resourceP1.address, 1);
  tx0.wait();
  let tx1 = await p2.AddToResources(resAddress);
  tx1.wait();
  let resourceP2 = new ethers.Contract(resAddress,
ResourceJSON.abi, p2Ac);
  let tx2 = await resourceP2.SetQuantity(Number(initialQ) + 10);
  tx2.wait();
  let newQ = await resourceP2.GetQuantity();
  assert.equal(Number(initialQ) + 10, Number(newQ));
  assert.equal(p2.address, await resourceP2.owner());
});
```

Here the code instantiates two Producer SCs and transfers the ownership of one Resource SC from the first producer to the second. It then performs some write operation to check if the transfer has been truly successful.



## Read Events from Events Log

```
after("Read Events from events log", async () => {
  let res0 = new ethers.Contract(res0Addr, ResourceJSON.abi,
p1Ac);
  let eventFilter = res0.filters.agriEvent();
  let events = await res0.queryFilter(eventFilter);
  console.log(events);
});
```

```
[
  {
    blockNumber: 55,
    blockHash: '0x2edff1dbef36eabd69834dceafcb996848f6b040df204bcfc8bce5615c607598',
    transactionIndex: 0,
    removed: false,
    address: '0x9A6ada768DC0a3e050b71138c1a2a5096c2Bd0C3',
    data: '0x',
    topics: [
      '0x9e003c88e545ece8ffbfec54d83dd54505596c479a3c3a1de531bf602682c642',
      '0x00000000000000000000000000000000c8a0b5165885d6f7fa0cb4e1c9e11c44067eaca0',
      '0x000000000000000000000000000000009a6ada768dc0a3e050b71138c1a2a5096c2bd0c3'
    ],
    transactionHash: '0xa053400ce9eff1380c501e50f13187cdc3dcd4abb6cb1c67886ff0f716fdc564',
    logIndex: 0,
    removeListener: [Function (anonymous)],
    getBlock: [Function (anonymous)],
    getTransaction: [Function (anonymous)],
    getTransactionReceipt: [Function (anonymous)],
    event: 'agriEvent',
    eventSignature: 'agriEvent(address,address)',
    decode: [Function (anonymous)],
    args: [
      '0xc8a0b5165885D6f7FA0CB4e1c9e11C44067EACA0',
      '0x9A6ada768DC0a3e050b71138c1a2a5096c2Bd0C3',
      registrant: '0xc8a0b5165885D6f7FA0CB4e1c9e11C44067EACA0',
      resource: '0x9A6ada768DC0a3e050b71138c1a2a5096c2Bd0C3'
    ]
  },
  {
    blockNumber: 57,
    blockHash: '0xb5a62c6c2158cd7dce96f750ab2df812f16733350381f74901c2ce4675e8d3a6',
```

This is an example of an **event** log

## Results

### Ethereum Test Results

```
Test Smart contracts for Agri food chain traceability on Ethereum
✓ create first resource (265ms)
✓ create second resource (314ms)
✓ Add and read event from storage (265ms)
✓ Add external certifier to authorized accounts (94ms)
✓ Event from external certifier (238ms)
✓ -----transfer resource from one producer to another----- (425ms)

____ETH LOG____

Create 1st Resource
  Gas Used: 989273
Create 2nd Resource
  Gas Used: 1034379
AgriEvent from Storage:
  block timestamp: 1643320408
  registrar address: 0x79a80228D4D6036b2B93F235a04C4CDb0F8b79a0
  event name: first event
  parameters: {"action":"fertilizing","quantity":"10 liters"}
  Gas Used: 171524
Add authorized account
  Gas Used: 44696
Add Event from an external certifier
  block timestamp: 1643320408
  registrar address: 0xe8d5A1D4B11Bb581285e908143c5F5458f4f4F49
  event name: first event
  parameters: {"action":"certify origins","status":"OK"}
  Gas Used: 156464
Change Producer
  Gas Used: 49465
Add to Resources
  Gas Used: 66637
Set quantity
  Gas Used: 28264
```

### IOTA test Results

```
Test Smart contracts for Agri food chain traceability on IOTA
✓ create first resource (994ms)
✓ create second resource (998ms)
✓ Add and read event from storage (1013ms)
✓ Add external certifier to authorized accounts (987ms)
✓ Event from external certifier (1013ms)
✓ -----transfer resource from one producer to another----- (3009ms)

____IOTA LOG____

Create 1st Resource
  Gas Used: 1000273
Create 2nd Resource
  Gas Used: 1045979
AgriEvent from Storage:
  block timestamp: 1643320408
  registrar address: 0xc8a0b5165885D6f7FA0CB4e1c9e11C44067EACA0
  event name: first event
  parameters: {"action":"fertilizing","quantity":"10 liters"}
  Gas Used: 185624
Add authorized account
  Gas Used: 48596
Add Event from an external certifier
  block timestamp: 1643320410
  registrar address: 0x323d67A43845022791138fdefe5C8Ccc0Db90dD7
  event name: first event
  parameters: {"action":"certify origins","status":"OK"}
  Gas Used: 168464
Change Producer
  Gas Used: 52865
Add to Resources
  Gas Used: 71837
Set quantity
  Gas Used: 30864
```

Test Execution Time Comparison:

Test	IOTA	ETH
Create 1st Resource	1011	290
Create 2nd Resource	1002	323
Add and Read Event from Storage	999	291
Add external Certifier to authorized accounts	996	96
Add Event from an external certifier	1016	254
Transfer resource from one producer to another	3015	428

Transaction Gas Consumption Comparison:

Name	Description	IOTA	ETH
CreateResource	Create 1st Resource	1000273	989273
CreateResource	Create 2nd Resource	1045979	1034379
AddEvent	AgriEvent from Storage	185624	171524
addAuthorized	Add authorized account	48596	44696
AddEvent	Add Event from an external certifier	168464	156464
ChangeProducer	Change Producer	52865	49465
AddToResources	Add to Resources	71837	66637
SetQuantity	Set quantity	30864	28264

As we can see from the results of these tests we can assume 2 main things:

- The IOTA is advancing in the right direction in the implementation of fully decentralized and environment friendly EVMs
- With these settings the Ethereum native EVM is much more performant than the IOTA EVM.

While this assumptions seems reasonable, we need to take in account that IOTA EVMs is undergoing and probably still needs undergo under intense development, changes and integration tests within the IOTA platform before it can be regarded as a stable alternative to the stable Ethereum EVMs, on the other hand with the adoption of more nodes (in this test case there was only 1 local node) while with the IOTA architecture allowing for any node to validate new blocks the performance would grow the Ethereum blockchain would experience lower performances and eventually bottlenecks.

## Conclusions

IOTA presents many advantages for supply chain management and food traceability compared to Ethereum. Spending even a cheap gas fee for each transaction in a traceability system in the never stopping food trade would greatly discourage its adoption from local farmers or little agri firms, as they may not find it feasible, on the contrary bigger agri firms could find the fee negligible. In this use case, with IOTA allowing many free transactions per second, integrating Ethereum EVMs and implementing a truly decentralized system we don't see many advantages with the Ethereum platform.

However, Ethereum is a fairly stable and tested platform, IOTA still has a lot to go through and its version 2.0 is still in the Beta stage. For example when starting the wasp node and goshimmer networks during the development we often had unexpected and virtually inexplicable crashes sometimes simply solved with a simple restart of the programs or at times of the hosting machine. I also encountered problems using the mainstream Web3.js library when interacting with the IOTA network and had to rely on the Ethers.js library. On the bright side Ether.js seems to be a more concise, easy to read and well thought out library, at least in this use case.

IOTA may not be as captivating for users at first since it needs a wide adoption to increase its performance while Ethereum could appear more captivating even if in the long run will probably be slower than the other.

Some improvements and or revisitation of the sample DApp engineered for the purpose of this paper might be the development of a front-end User Interface or a complete revisitation of the application in Rust, a high level performant language which is supported by the ISCP through the native Wasm VM.

## References

- [1] ISO Technical Committee, Traceability in the Feed and Food Chain—General Principles and Basic Requirements for System Design and Implementation, ISO 22005:2007, Geneva, Switzerland, 2016, accessed:2020-11-10.  
[\[https://www.iso.org/standard/36297.html\]](https://www.iso.org/standard/36297.html)
- [2] M. Tripoli and J. Schmidhuber, “Emerging opportunities for the application of blockchain in the agri-food industry,” FAO and ICTSD: Rome and Geneva. License: CC BY-NC-SA, vol. 3, 2018.
- [3] M. P. Caro, M. S. Ali, M. Vecchio, and R. Giaffreda, “Blockchain-based traceability in agri-food supply chain management: A practical implementation,” in 2018 IoT Vertical and Topical Summit on Agriculture-Tuscany (IOT Tuscany). IEEE, 2018, pp. 1–4.
- [4] G. Baralla, A. Pinna, and G. Corrias, “Ensure traceability in European food supply chain by using a blockchain system,” in 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). IEEE, 2019, pp. 40–47.
- [5] S. Wang, D. Li, Y. Zhang, and J. Chen, “Smart contract based product traceability system in the supply chain scenario,” IEEE Access, vol. 7, pp. 115 122–115 133, 2019.
- [6] B. Yu, P. Zhan, M. Lei, F. Zhou, and P. Wang, “Food Quality Monitoring System Based on Smart Contracts and Evaluation Models,” IEEE Access, vol. 8, pp. 12 479–12 490, 2020.
- [7] IOTA Foundation [\[https://www.iota.org/\]](https://www.iota.org/).
- [8] Ethereum [\[https://ethereum.org/en/\]](https://ethereum.org/en/).
- [9] IOTA 2.0 Platform [\[https://v2.iota.org/\]](https://v2.iota.org/).
- [10] N.Szabo, “Smart Contracts: Building Blocks for Digital Markets”(1996)  
[\[https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart\\_contracts\\_2.html\]](https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html)
- [12] S. N. Khan, F. Loukil, C. Ghedira-Guegan, E. Benkhelifa, A. Bani-Hani “Blockchain smart contracts: Applications, challenges, and future trends”  
[\[https://link.springer.com/article/10.1007/s12083-021-01127-0\]](https://link.springer.com/article/10.1007/s12083-021-01127-0).
- [13][\[https://www.gemini.com/cryptopedia/blockchain-trilemma-decentralization-scalability-definition#section-what-is-scalability\]](https://www.gemini.com/cryptopedia/blockchain-trilemma-decentralization-scalability-definition#section-what-is-scalability)
- [14] Bitcoin Energy consumption  
[\[https://www.mdpi.com/1996-1073/14/14/4254/pdf\]](https://www.mdpi.com/1996-1073/14/14/4254/pdf)
- [15] Transaction Confirmation Time Prediction in Ethereum Blockchain Using Machine Learning  
[\[https://arxiv.org/pdf/1911.11592.pdf\]](https://arxiv.org/pdf/1911.11592.pdf)
- [16] IOTA wiki  
[\[https://wiki.iota.org/learn/about-iota/an-introduction-to-iota\]](https://wiki.iota.org/learn/about-iota/an-introduction-to-iota)
- [17] IOTA DevNet Visualizer  
[\[https://explorer.iota.org/devnet/visualizer/\]](https://explorer.iota.org/devnet/visualizer/)

- [18] Lodovica Marchesi, Katuscia Mannaro, Raffaele Porcu  
Automatic Generation of Blockchain Agri-food Traceability Systems  
[\[https://arxiv.org/abs/2103.07315\]](https://arxiv.org/abs/2103.07315)
- [19] OpenZeppelin, “Openzeppelin: Contracts”.  
[\[https://github.com/OpenZeppelin/Openzeppelin-contracts\]](https://github.com/OpenZeppelin/Openzeppelin-contracts)
- [20] Solidity Docs [\[https://docs.soliditylang.org/en/latest/index.html\]](https://docs.soliditylang.org/en/latest/index.html)