SN Computer Science (2022) 3:279
https://doi.org/10.1007/s42979-022-01148-3

ORIGINAL RESEARCH

# A Blockchain-Based System for Agri-Food Supply Chain Traceability Management

**Angelo Marchese**[1] · **Orazio Tomarchio**[1]

Received: 13 September 2021 / Accepted: 11 April 2022


**Abstract**
Ensuring high quality and safety of food products has become a key factor on one hand to protect and improve consumers health and, on the other one, to gain market share. For this reason, much effort in the last year has been devoted to the development of integrated and innovative Agriculture and Food (Agri-Food) supply chains management systems, which should be responsible, in addition to track and store orders and deliveries, to guarantee transparency and traceability of the food production and transformation process. In this paper, differently from traditional supply chains which are based on centralized systems, we propose a fully distributed approach, based on blockchain technology, to define a supply chain management system able to provide quality, integrity and traceability of the entire supply chain process. The proposed framework is based on the Hyperledger Fabric technology, which is a permissioned blockchain system: a prototype has been developed and, by using some use cases, we show the effectiveness of the approach.

**Keywords** Blockchain · Hyperledger fabric · Agri-food supply chain

## Introduction

The recent attention on food safety and product quality requires more reliable and efficient processes for the management of agri-food supply chains ([15, 21]). Government authorities need to respond more promptly to food scandals and accidents to maintain customer confidence in the food industry. To this end, ensuring the traceability of food products allows to provide consumers with a complete view of the different phases of product harvesting, processing and distribution ([5, 10, 20]). Many of the management processes of current supply chains have been automated to reduce operational costs and errors and to improve the monitoring and collection of information related to the various activities within the supply chain. Therefore, by collecting and making accessible the set of traceability information of a product, it is possible for consumers to know about the entire life cycle of that product, its related transactions and chain of owners and its provenance.

However, one of the issues of today's supply chain management systems is that they are often based on centralized systems: supply chain processes and product traceability data are managed by a single authority on which supply chain members rely on to transfer and share their information. These centralized systems are often non-transparent, monopolistic and asymmetric information systems. This can pose a serious threat to the security and reliability of the traceability information and make fraud, corruption and data falsification easier ([23]). Furthermore, another issue with such centralized systems is related with the risk of a single authority to become the weak link and single point of failure. Also operation throughput and scalability are limited.

To deal with such issues, the usage of blockchain technology in this domain has recently been proposed to support the management of supply chain traceability ([2, 12, 27]). Blockchain technology in particular offers cryptographic primitives to store data within a distributed ledger, guaranteeing their immutability and authenticity. The decentralization that blockchain provides reduces the risk of data loss

This article is part of the topical collection "Enterprise Information Systems" guest edited by Michal Smialek, Slimane Hammoudi, Alexander Brodsky and Joaquim Filipe.

✉ Angelo Marchese
angelo.marchese@phd.unict.it

Orazio Tomarchio
orazio.tomarchio@unict.it

[1] Department of Electrical Electronic and Computer Engineering, University of Catania, Catania, Italy

Published online: 07 May 2022

and corruption. In particular, whenever an actor attempts to change data on the blockchain, network participants would be immediately aware of the tampering upon inspection of the chain. This eliminates the need for supply chain members to trust a single entity to manage their traceability information. Furthermore, being a distributed system, the blockchain can mitigate the problems of limited scalability and single point of failure. If, on the one hand, the use of a blockchain implies an additional overhead with respect to a centralized system, on the other one the higher management costs are compensated by a higher traceability and visibility on supply chain operations.

In this paper, by extending our previous work presented in [17], we propose a complete model of a blockchain based agri-food supply chain traceability system providing a prototype implementation to show the applicability of blockchain technology in this domain. The main focus of the system is to take advantage of the blockchain features to allow supply chain members to store and manage product-related traceability information in a transparent, reliable and tamper-proof way. Using this information final consumers can then reconstruct a complete history of the transactions related to a product during its life cycle. Transactions of the real world, involving transformations and exchanges of physical goods, proceed as usual. Our framework registers and stores information about these transactions, but how this information are generated and retrieved is out of the scope of our work. The main goal of the proposed approach is to make supply chain transactions traceable and verifiable by external users. The business logic of the system is executed by a *smart contract* that allows to automate some of the management processes related to supply chain activities. The smart contract offers operations that can be invoked by the supply chain members to store and update traceability information. The proposed system also makes it possible to associate rules with supply chain products, allowing the expression of product-specific quality control mechanisms. This functionality has been included considering that in the context of agri-food supply chains the regulatory aspects are of fundamental importance to ensure food safety and quality, also taking into account that these aspects vary from a case to another one and dynamically evolve over time ([8]).

The system was implemented using the Hyperledger Fabric[1] blockchain, an emerging open-source technology widely used also in other proposed examples of supply chain management systems ([25]). In addition, the components of our system are deployed in a cloud environment within a Kubernetes[2] cluster, showing that, although our system is a prototype, it can be easily migrated to a scalable production environment. Finally, to show the behaviour of our system and the effectiveness of the approach, we present two slightly different use cases where the main features of the prototype are demonstrated.

The rest of the paper is organized as follows. In Sect. "Background and Related Work", we provide a background of technologies exploited for this work and present related works. Section "System Architecture" presents the overall architecture of our framework. Section "Business Logic" describes the system business logic and the operations offered by the smart contract. Section "System Implementation" provides implementation details about system components and the smart contract. Section "Use Cases" discusses about two examples of usage of our system in a prototype environment. Finally, Sect. "Conclusions" concludes the work.

## Background and Related Work

This section provides some background information about the technologies exploited in our work, such as the blockchain technology and in particular the Hyperledger Fabric system. Then, after briefly outlining the advantages that would derive from their use to support agri-food supply chain traceability systems, some related works in the literature are discussed.

### Blockchain Technology and Hyperledger Fabric

Blockchain technology represents a particular class of distributed systems and as such was born with the aim of overcoming some of the problems related to centralized systems ([13, 14]). The application area in which the blockchain was initially introduced is that of transactional systems, in particular electronic payment systems. That is the case, for example, of the Bitcoin blockchain ([19]). However, today blockchain technology is increasingly being adopted in a lot of different application domains.

In general, operations within a blockchain are carried out by nodes connected to each other through a peer-to-peer network. In public blockchains, like Bitcoin, every node can participate in network operations and can decide to exit at any time. Each node participating in the blockchain maintains a local copy of a distributed ledger which contains a set of append-only logs that encode the status information of the blockchain. More specifically, an ordered sequence of blocks is stored inside the ledger. Each block consists of a header and a body that contains an ordered list of transactions which are validated and executed by the peers of the network. To guarantee the immutability and reliability of the data in the ledger, each block of the sequence contains a cryptographic hash of the previous block within a header field. In this way,

[1] https://www.hyperledger.org/use/fabric.

[2] https://kubernetes.io/.

a malicious attempt to change the content of a block would require to correspondingly modify the header of all the following blocks in the sequence, which is a computationally expensive task thanks to the non-invertibility property of hash functions. Peer nodes of a blockchain coordinates with each other throughout a consensus protocol. Public blockchains typically use secure but computational expensive consensus protocols, like for example the proof of work protocol. This is motivated by the fact that any node can join a public blockchain and this makes this type of blockchains an untrusted environment. The use of a computational expensive consensus protocol poses some scalability issues and transaction throughput limitations.

Permissioned blockchains are another category of blockchains. In a permissioned blockchain, only authorized peers can participate in blockchain operations. Permissioned blockchains often set aside proof of work consensus protocol because of its nondeterminism and the computational burden it imposes on peer nodes. Instead, they adopt weaker but more performant consensus mechanisms based on traditional protocols from distributed computing, such as Paxos, Raft and Byzantine fault-tolerant algorithms. This is possible because in a permissioned blockchain membership is limited only to a well-known set of entities and this involves less security risks. Agri-food supply chains fit well in the context of permissioned blockchains ([26]). In an agri-food supply chain scenario a limited set of organizations, whose identities are known, are supposed to actively participate in supply chain operations. Organizations typically don't trust each other, but they need read and write access to a trusted shared data repository. While the presence of an always-online trusted third party authority that manages all supply chain operations allows to avoid the use of a blockchain, this is not always a realistic scenario. In some situations it is not possible to have a single authority trusted by all parties and delegating all the write operations to a centralized entity can cause it to become a single point of failure. Although the use of a blockchain can limit transaction throughput, the decentralized nature of the blockchain technology allows to obtain better scalability and to solve the single point of failure issue.

Hyperledger Fabric is an open-source blockchain platform, which falls within the category of permissioned blockchains ([1]). Hyperledger Fabric is a distributed operating system that runs applications written in general-purpose programming languages, such as Go, Java, JavaScript, and Python. It introduces the execute-order-validate blockchain model for transaction processing unlike other traditional blockchain systems that use the order-execute model. In the order-execute model a protocol for consensus first orders the transactions and propagates them to all peers that execute the transactions sequentially. This model requires that all transactions must be performed sequentially by each peer, and this implies several performance limitations. Furthermore, transactions must be deterministic, which is not always easy to ensure. On the other hand, the execute-order-validate model separates the transaction flow into three steps: the execution of the transaction and check of its correctness, the transaction ordering through a consensus protocol and the transaction validation. In this model, each transaction is executed and checked only by a subset of the peers, which allows for parallel execution and addresses potential non-determinism. This allows to overcome the limitations of the execute-order model mentioned above.

Like some other blockchains, Hyperledger Fabric offers the smart contract primitive. A smart contract is a combination of data and code that encodes a set of transformations on that data. It exposes a set of operations that can be invoked by the users of the blockchain with the aim of changing the state of the distributed ledger. The concept of smart contract, therefore, makes this kind of blockchain a distributed execution environment of general-purpose programmable logic.

Thanks to the aforementioned properties, blockchain technology is a good candidate to address some of the actual problems related to traditional centralized agri-food supply chain traceability systems. In particular, it can guarantee the transparency, verifiability and immutability of traceability data, simplifying the information sharing between the supply chain entities often belonging to distinct administrative organization. In this way the traceability of the supply chain products can be guaranteed, allowing the consumer to reconstruct the entire product's life cycle within the supply chain and to verify its origin and authenticity. Finally, smart contracts can be used to automate the supply chain management and product quality control operations. Although the use of the blockchain technology implies additional overhead than in the case of a centralized system managed by a single authority, this overhead is covered by the interest of supply chain members to produce certified and traceable products to increase consumer trust. In the same way, final consumers would pay more for validated products.

## Related Work

In the literature, there is a variety of works that propose the use of blockchain technology to build agri-food supply chain management systems and in some cases implementations of such systems are also proposed. Some of these works are briefly described below.

In Malik et al. [16] a permissioned blockchain system, called ProductChain, is proposed. The system is administered by a consortium of entities participating in a generic food supply chain, including governmental and regulatory entities. It stores product traceability information made accessible to consumers. The authors propose the use of a three-tier sharded architecture that ensures reliability

and availability of data for consumers and scalability with respect to transaction execution throughput. They also propose the use of a transaction vocabulary and the implementation of access control mechanisms to manage read and write privileges on the blockchain.

Wang et al. [24] propose a product traceability system based on the Ethereum blockchain and the smart contract primitive. The system stores information related to the products life cycle and also provides for the implementation of event-response mechanisms to verify the identities of both parties of all transactions at the time of their submission, so that their validity is guaranteed. All the events are kept in the system permanently so that any disputes can be managed and the responsible for certain actions can be traced.

In Caro et al. [6] the AgriBlockIoT is proposed, a totally distributed and blockchain-based supply chain management system, able to integrate multiple IoT devices that collect and produce digital data along the supply chain. To efficiently evaluate AgriBlockIoT, the authors defined a use case based on the from-farm-to-fork model. This use case was then implemented using two different blockchain systems, namely Ethereum and Hyperledger Sawtooth.

Casino et al. [7] propose a distributed functional model based on blockchain to create distributed and automated traceability mechanisms for a generic agri-food supply chain. To evaluate the feasibility of the proposed model, a use case is presented. The applicability of the model is also illustrated through the development of a fully functional smart contract and a private blockchain.

Tian [11] propose a food supply chain traceability system for real-time food tracing based on HACCP (Hazard Analysis and Critical Control Points), blockchain and the Internet of Things, which provides a platform that ensures openness, transparency, neutrality, reliability and security for traceability information. The proposed system uses BigchainDB, which combines the key benefits of distributed databases and blockchain.

Biswas et al. [4] propose a blockchain-based system to achieve the traceability of the activities that occur within the supply chain related to wine production. The proposed traceability system uses MultiChain to implement a private blockchain.

Shahid et al. [22] present a complete solution for blockchain-based agri-food supply chains. The proposed solution leverages the key features of blockchain and smart contracts, deployed over Ethereum blockchain network. All transactions are written to the blockchain which ultimately uploads the data to Interplanetary File Storage System (IPFS). The storage system returns a hash of the data which is stored on blockchain and ensures an efficient, secure and reliable solution. Authors provide simulations and evaluation of smart contracts along with the security and vulnerability analyses.

Cocco et al. [9] propose a blockchain-based system for the supply chain management of a particular Italian bread. To realize the system authors relied on the blockchain and the Internet of Things technologies to provide a trustless environment. The system is designed so that along the supply chain, the nodes equipped with several sensors directly communicate their data to Raspberry Pi units that elaborate and transmit them to IPFS and to the Ethereum blockchain. Furthermore, authors designed ad hoc Radio Frequency Identification and Near Field communication tags to shortly supply the proposed system with information about the products and batches.

Baralla et al. [3] present a blockchain oriented platform to guarantee the origin and provenance of food items in a Smart Tourism Region context. The proposed solution uses smart contracts in order to guarantee transparency, efficiency and trustworthiness. The system is particularly suitable to manage cold chain since it interfaces with IoT network devices providing detailed information about data monitoring food such as storage temperature, environment humidity, and GPS data.

Marchesi et al. [18] propose a general-purpose approach for the blockchain-based agri-food supply chain management, proposing a system that can be configured for most agri-food productions. The primary purpose is to provide a methodology to facilitate and make more efficient the development of supply chain management applications that make use of blockchain technology. It is based on general smart contracts and apps interacting with the same smart contracts, which are configured, starting from the description of the specific system to be managed, using JSON files.

Like the aforementioned research works, in our work we propose a complete solution of a blockchain-based agri-food traceability system, providing, in particular, a description of the architectural components, the information model and the business logic of this system. A distinctive contribution of our work, is the capability to allow the specification of custom regulations for supply chain products at runtime and to automate the validation of these regulations. Our framework has addressed this aspect considering the heterogeneity of product regulations among supply chains and the fact that these regulations change over time.

## System Architecture

This section provides a high-level description of our blockchain-based system for agri-food supply chain traceability, the goals that guided its design and its general architecture. The proposed system is designed to manage the traceability information of products and activities related to one or more agri-food supply chains. The main objective is to allow to reconstruct the entire flow of activities and

transactions related to a product from its origin to the end consumer. The system has to automate all those operations related to product quality control and regulatory compliance. It has to be able to dynamically adapt to changes in laws and regulations. It should also be scalable, able to handle an ever-increasing amount of information. Finally, the system has to guarantee reliability and availability, especially when dealing with environments characterized by continuous flows of transactions.

The fundamental part of the framework consists of a permissioned blockchain, implemented through the Hyperledger Fabric framework ([1]). In this blockchain, the core of the system's business logic is executed in the form of a *smart contract*. The smart contract offers several operations that allow users of the system to add and modify information in the blockchain in a secure and traceable way. Users of the system are the supply chain members and the regulatory departments. The former add and modify information related to their products, while the latter deal with the management and regulation aspects of supply chains. More specifically, the entities participating in the system operations are user organizations, where each user is identified by a certificate issued by a certification authority associated with the organization to which the user belongs. Since the blockchain is permissioned, only a well-defined set of organizations can participate in the system operations. In Hyperledger Fabric the set of organizations participating in blockchain operations is predetermined. Hyperledger Fabric allows to add a new organization or remove an existing one at run-time by submitting a series of transactions to the blockchain that must be approved by a majority of the participating organizations. In this work, we have not considered the ability to add or remove dynamically organizations to the blockchain and this is something that can be evaluated and included in our system in a future work.

The interaction between users and the blockchain takes place through a client application that runs within an application server and the interaction with the latter takes place through a frontend application that is typically hosted by a web server. Each organization has its own application server and web server.

Each organization has its own role that defines its interactions with the system and the operations it can perform. According to common models of agri-food supply chain described in [11, 24] we consider the following roles:

- *Producer*: organization that requires the registration of one or more primary products (i.e. products whose batches do not derive from any other batch). If a registration request is accepted, this organization can register batches associated with the registered product or products in the system.
- *Manufacturer*: organization that requires the registration of one or more derived products (i.e. products whose batches derive from batches of other primary or derived products). If a registration request is accepted this organization can register batches associated with the registered product or products in the system, specifying a list of batches from which the registered batch derives.
- *Deliverer*: organization that buys batches from organizations and resells them to other organizations.
- *Retailer*: organization that sells products to consumers.
- *Regulatory Department*: organization that manages and monitors the activities within the various supply chains. More specifically, an organization with the role of *Regulatory Department* adds product types to the system, associating them with rules and assigning roles to the various organizations.

In the following, while describing the behaviour of our framework, we refer to a scenario involving five organizations, one for each of the roles listed above, which is depicted in Fig. 1.

## Business Logic

In this section, the basic design of the system and its behaviour is described. First, a domain model with the different resource types and their relationships is illustrated. Then a description of the smart contract operations and their parameters is provided. Finally, the state diagrams that illustrate the state transitions of the system's resource types caused by the execution of the smart contract operations are described.

### Domain Model

Figure 2 shows a domain model of the system's business logic with the different resource types and their relationships.

Each *Organization* of the system is identified by a unique identifier and can be associated with a *RoleSet*, which represents a list of roles. A *RoleSet* associated with an organization defines the set of smart contract operations the organization can invoke.

An organization playing the role of *Regulatory Department* can register one or more product types in the system. A *ProductType* is uniquely identified by a name and can be either primary or derived. In case the product type is derived, it has a list of product types ingredients which it is derived from. This means that any *Batch* associated with this product type must have a list of batches ingredients whose respective product types are in the list of product types ingredients. It is possible to associate one or more rules to a product type, where each *Rule* represents a set of conditions that have to be respected when registering batches associated

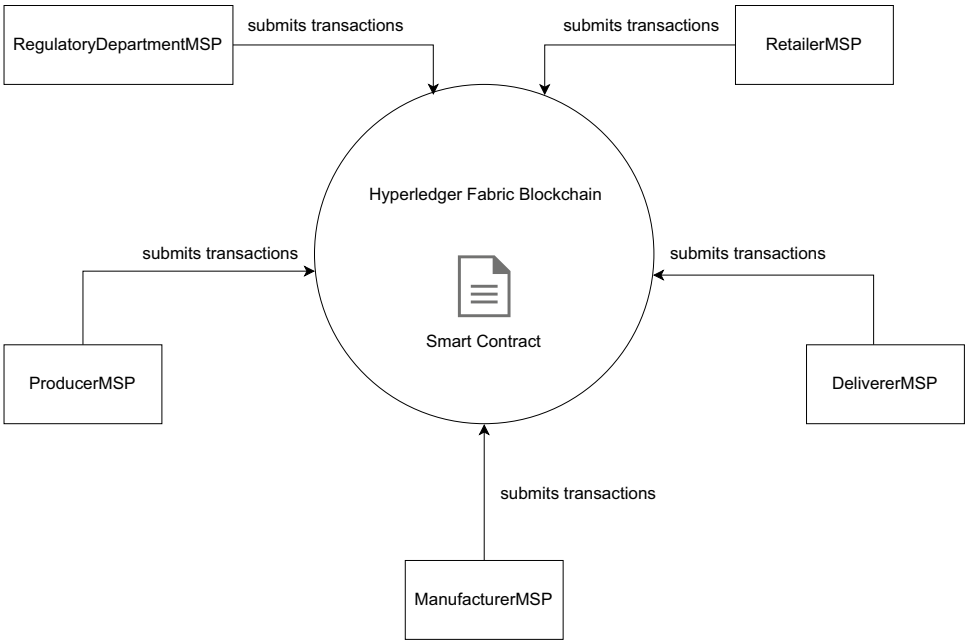


**Fig. 1** Typical agri-food supply chain scenario

with that product type. At the moment of a batch registration these rules are validated using a *RuleEngine*.

A *ProductType* may be associated with one or more *Products*, for each of which an owner organization requires the registration in the system. A request for the registration of a product can be accepted by an organization playing the role of *Regulatory Department* and from that moment the organization that owns the product can register batches of that product in the system. A product is uniquely identified by a name.

A *Product* may be associated with one or more *Batches* that are registered by the organization that owns that product. A batch is uniquely identified by an ID and a set of parameters as specified at registration time. When registering a batch associated with a product of a derived product type, it is necessary to specify a list of batches ingredients from which this batch derives. This list must be consistent with the list of product types ingredients associated with the product type of the registered batch. A batch can be transferred from one organization to another one and an organization that owns a batch can use that batch as an ingredient when registering a new batch. The domain model depicted in Fig. 2 also shows that each resource in the system has a state. It provides information on the current conditions of that resource, determines the operations that can be performed on it and the subsequent states in which it can transit.

## Smart Contract Operations

As already explained in Sect. "Background and Related Work", the Hyperledger Fabric framework is based on the smart contract concept. Several smart contract operations (listed in Fig. 3)have been designed in our systems that can be executed by the different user organizations of the supply chain. In the following we describe the behaviour of each of these smart contract operations, also specifying their inputs and their expected results.

The operation *addRoleSet()* takes the parameters *orgId* and *roles* as inputs. This operation can only be performed by an organization with role of *RegulatoryDepartment* and allows to create a new role set with the specified list of roles associated with the organization identified by *orgId*.

The operation *addProductType()* takes the parameters *productTypeName*, *type* and *productTypeIngredientNames* as inputs. This operation can only be performed by an organization with role of *RegulatoryDepartment* and allows to create a new product type with the specified name, type and list of product types ingredients. The value of the parameter *type* can only be *primary* or *derived*. If the value of this parameter is *derived* then the parameter *productTypeIngredientNames* must contain at least one value, otherwise if the value is *primary* it must be an empty list.

The operation *addRule()* takes the parameters *productTypeName* and *ruleString* as inputs. This operation can only be performed by an organization with role of *RegulatoryDepartment* and allows to create a new rule with the specified string expression associated with the product type identified by *productTypeName*. The operations *enableRule()* and *disableRule()* take the parameter *ruleId* as input. These operations can only be performed by an organization with role of *RegulatoryDepartment* and allow to enable and disable respectively the rule identified by *ruleId*.

The operations *blockProductType()* and *unblockProductType()* take the parameter *productTypeName* as input.

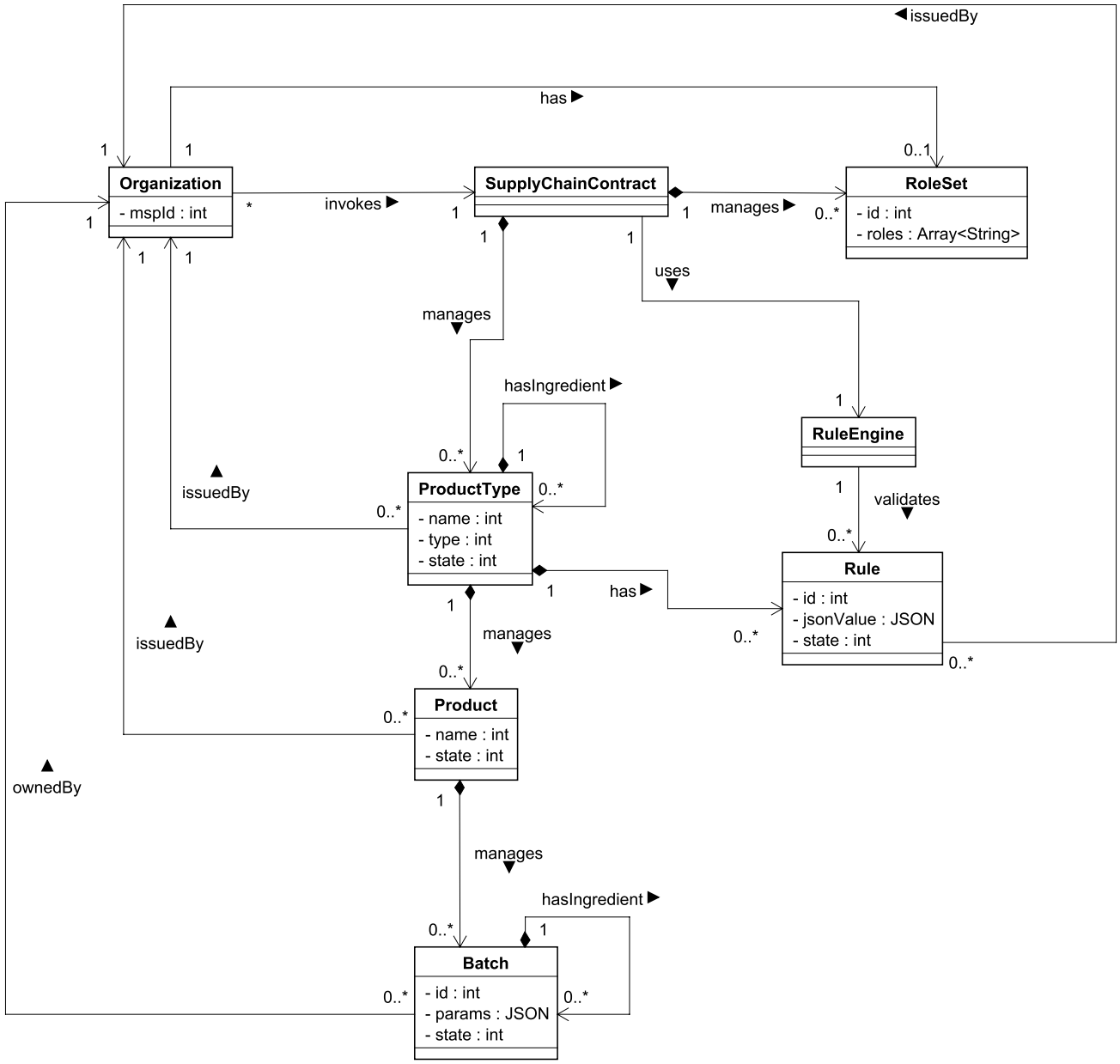


**Fig. 2** Domain model of the business logic

These operations can only be performed by an organization with role of *RegulatoryDepartment* and allow to block and unblock respectively the product type identified by *productTypeName*.

The operation *requestProductRegistration()* takes the parameters *productTypeName* and *productName* as inputs. This operation allows to request the registration of a new product identified by *productName* associated with a product type identified by *productTypeName*. If the product type specified is a primary product type then the operation can be performed only by an organization with role of *Producer*, otherwise if it is a derived product type then the operation can be performed only by an organization with role of *Manufacturer*. The operations *acceptProductRegistration()* and *refuseProductRegistration()* take the parameter *productName* as input. These operations can only be performed by an organization with role of *RegulatoryDepartment* and allow to accept and refuse respectively the registration request for the product identified by *productName*.

The operations *blockProduct()* and *unblockProduct()* take the parameter *productName* as input. These operations can only be performed by an organization with role of *RegulatoryDepartment* or the owner organization of

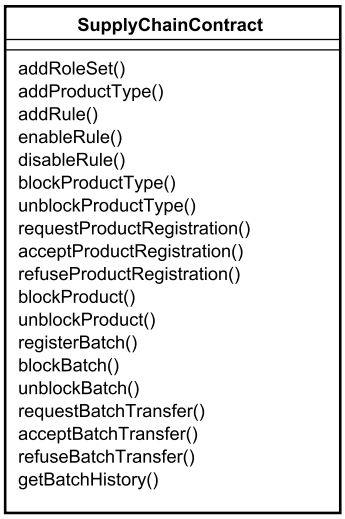


**Fig. 3** Smart contract operations

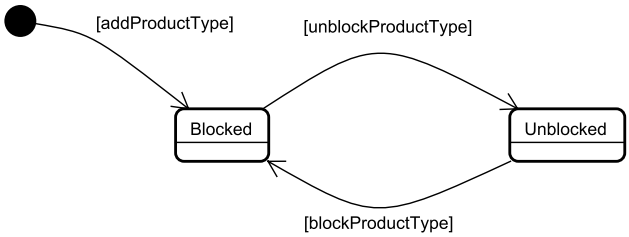


**Fig. 4** State diagram of resource type ProductType

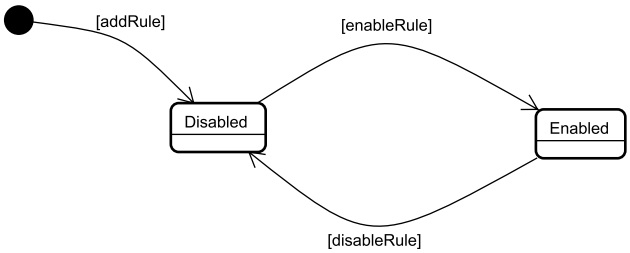


**Fig. 5** State diagram of resource type Rule

the product identified by *productName* and allow to block and unblock that product respectively.

The operation *registerBatch()* takes the parameters *productName*, *batchIngredientIds* and *params* as inputs. This operation can only be performed by the owner organization of the product identified by *productName* and allows to register a new batch associated with that product. The parameter *batchIngredientIds* represents a list of ingredient batch ids. This list must be consistent with the list of product types ingredients of the product type associated with the batch being registered. The parameter *params* represents a set of information related to the batch and some of these information are used during rules validation.

The operations *blockBatch()* and *unblockBatch()* take the parameter *batchId* as input. These operations can only be performed by an organization with role of *RegulatoryDepartment* or the owner organization of the batch identified by *batchId* and allow to block and unblock that batch respectively.

The operation *requestBatchTransfer()* takes the parameter *batchId* as input and allows an organization to request the transfer of the batch identified by *batchId* and owned by another organization. The operations *acceptBatchTransfer()* and *refuseBatchTransfer()* take the parameter *batchId* as input and allows the owner organization of the batch identified by *batchId* to accept and refuse a transfer request for that batch respectively.

Finally, the operation *getBatchHistory()* allows to obtain a complete history of the state transitions related to a batch. In this way, any organization can view the entire batch life cycle and the chain of its owners.

## Resource Types State Transitions

The execution of each smart contract operation may cause a state transition of a specific resource type in the system. In these section, we describe these transitions and the different states in which a resource can be found.

Figures 4, 5, 6 and 7 shows the state transitions of the different resource types available in our system, respectively, for *ProductType, Rule, Product,* and *Batch* resources. A new product type can be registered with the *addProductType()* operation and initially it starts from the *Blocked* state (Fig. 4). In this state no organization can request the registration of a new product for this product type. From the *Blocked* state a product type can be unblocked with the *unblockProductType()* operation, causing it to pass to the *Unblocked* state. From the *Unblocked* state, a product type can be blocked with the *blockProductType()* operation, causing it to pass to the *Blocked* state. When this last transition occurs all the products associated with this product type are also blocked.

A new rule, associated with a product type, can be registered with the *addRule()* operation and initially it starts from the *Disabled* state (Fig. 5). In this state, at the moment of registration of a new batch of the product type which this rule is associated with, the rule is not validated. From the *Disabled* state, a rule can be enabled with the *enableRule()* operation, causing it to pass to the *Enabled* state. In this state, at the moment of registration of a new batch the rule is always validated. From the *Enabled* state a rule can be disabled with the *disableRule()* operation, causing it to pass to the *Disabled* state. The ability to add, enable and disable

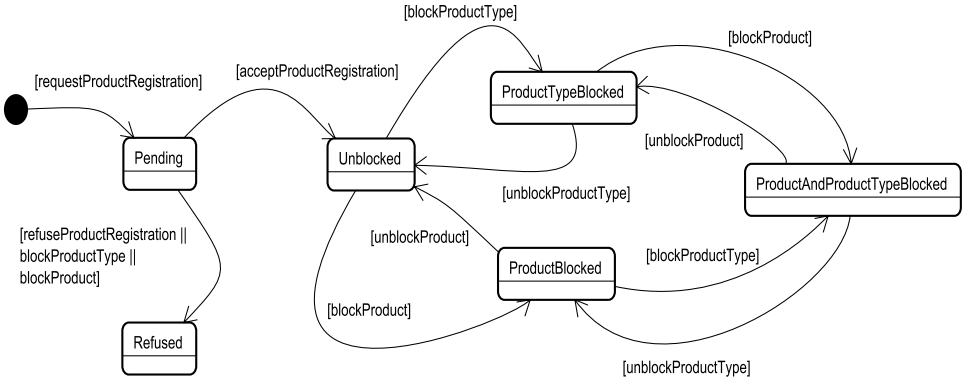


**Fig. 6** State diagram of resource type Product

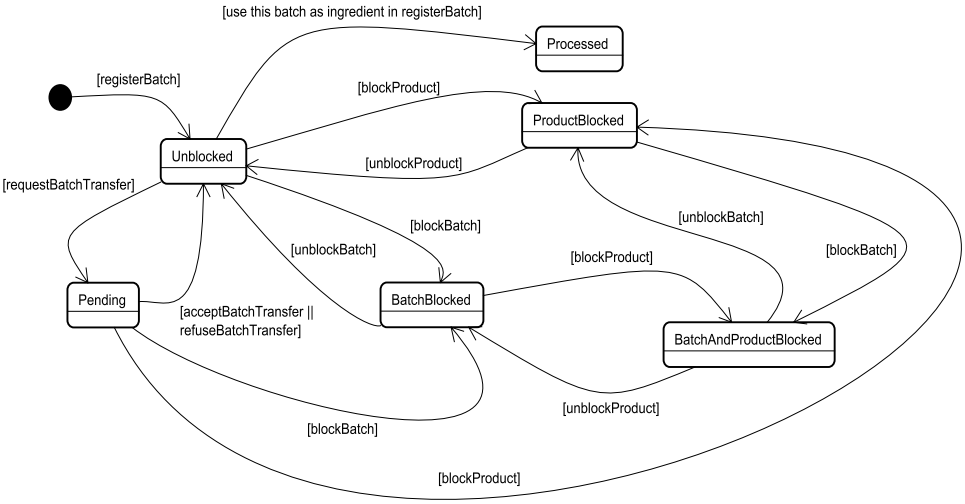


**Fig. 7** State diagram of resource type Batch

custom rules for a product type and to do it at runtime allows to implement product-specific quality control mechanisms that can change over time. This aspect is of fundamental importance due to the requirement of today's agri-food supply chains to establish products specific regulations that can frequently evolve over time.

The registration of a new product can be requested with the *requestProductRegistration()* operation (Fig. 6). When this happens, a new product is created that starts from the state *Pending*. From the *Pending* state, the product passes to the *Unblocked* one when the registration request for the product is accepted with the *acceptProductRegistration()* operation. From the *Pending* state the product passes to the *Refused* one when the registration request for the product is refused with the *refuseProductRegistration()* operation, or when the *blockProduct()* operation is executed or when the product type associated with the product is blocked. From the *Unblocked* state the product passes to the *ProductBlocked* one with the *blockProduct()* operation and to the *ProductTypeBlocked* one when the relative product type is blocked. From the *ProductBlocked* state the product passes to the *ProductAndProductTypeBlocked* one when the related product type is blocked and to the *Unblocked* state with the *unblockProduct()* operation. From the *ProductTypeBlocked* state the product passes to the *ProductAndProductTypeBlocked* one with the *blockProduct()* operation and to the *Unblocked* one when the relative product type is unblocked. From the *ProductAndProductTypeBlocked* state the product passes to the *ProductTypeBlocked* one with the *unblockProduct()* operation and to the *ProductBlocked* one when the relative product type is unblocked. While a product is in the *ProductBlocked*, *ProductTypeBlocked* and *ProductAndProductTypeBlocked* states, all the batches related to this product are also blocked and no new batch for this product can be registered.

The registration of a new batch can be requested with the operation *registerBatch()* (Fig. 7). When this happens, a new batch is created that starts from the *Unblocked* state. From the *Unblocked* state the batch passes to the *BatchBlocked* one with the *blockBatch()* operation, to the *Pending* one with the *requestBatchTransfer()* operation, to the ProductBlocked one when the product associated with this batch is blocked and to the *Processed* one when this batch is used as an ingredient for another batch. From the *Pending* state the batch passes to the *Unblocked* one with the *acceptBatchTransfer()* or *refuseBatchTransfer()* operations, to the *BatchBlocked*

state with the *blockBatch()* operation and to the *ProductBlocked* state when the relative product is blocked. From the *BatchBlocked* state the batch passes to the *BatchAndProductBlocked* one when the relative product is blocked and to the *Unblocked* one with the *unblockBatch()* operation. From the *ProductBlocked* state the batch passes to the *BatchAndProductBlocked* one with the *blockBatch()* operation and to the *Unblocked* state when the relative product is unblocked. From the *BatchAndProductBlocked* state the batch passes to the *ProductBlocked* one with the *unblockBatch()* operation and to the *BatchBlocked* when the relative product is unblocked. While a batch is in the *BatchBlocked*, *ProductBlocked* and *BatchAndProductBlocked* states, it cannot be transferred to other organizations and cannot be used as an ingredient for another batch.

The ability to block and unblock the different product types, products and batches in the system allows, together with the rule validation mechanism, to enhance the quality of the respective supply chains and to reduce the probability of food safety accidents. For example, if a batch does not pass quality control tests, it can be timely blocked so as to prevent it from reaching the final consumer or being used as an ingredient for another batch. Subsequent attempts to buy and sell this batch or to use it in a processing stage are prevented by the smart contract because the batch is in a blocked state. In the same way, if many batches related to the same product present some anomalies or their production process is found to be irregular, the product can be blocked, causing all its batches to be blocked. This allows to do some verification tests while avoiding the batches to go forward in the supply chain. If all verification tests pass, the product can be unblocked together with its batches. Finally, if harmful substances are found on a specific product type, this product type can be blocked, causing all its related products and batches to be blocked too.

## System Implementation

In this section, some implementation details about the system prototype are provided. First the deployment architecture and its main components are described. Then a description of the implementation of the smart contract and its class diagram are provided. The designed framework has been implemented and a prototype has been deployed within a Kubernetes cluster to emulate the distributed nature of the whole system, and to increase its portability and interoperability with existing organization IT systems. Figure 8 shows the software architecture in terms of the main components composing our framework: in particular, it shows the components for each organization of the scenario presented in Sect. 3, plus a set of components making up the Hyperledger Fabric blockchain.

Each of the organizations in the system runs a peer node that participates in blockchain operations and maintains information about its local copy of the distributed ledger in a dedicated database node (CouchDB in our prototype). Using a database node as local storage for a peer node is not necessary, but it allows for greater availability and for more complex queries on ledger data. While in our prototype, for simplicity purposes, we have chosen to run only one peer node for each organization, in a production environment each organization should run multiple peer nodes, in order to ensure high availability and to handle a higher transaction load.

When an organization wants to perform a smart contract operation, it submits a transaction, through a client application, to a majority of the peers in the blockchain. These peers validate and approve the transaction and, if successful, they send their approvals to the client application. In the case of a write operation on the ledger state the client application then sends the transaction along with the approvals to an Orderer node. The task of this node is to establish a total order of all transactions and to build blocks containing ordered transactions. These blocks are then distributed to the peer nodes and appended in the blockchain. Each peer commits and executes all the valid transactions in a block on its local copy of the ledger. As before, While in our prototype, for simplicity purposes, we have chosen to run the ordering service as a single node, in a production environment it should be executed by a set of nodes coordinating with each other via a consensus protocol (e.g. Raft), to ensure high availability of the service.

Each organization runs its own certificate authority that issues certificates for that organization's users and peer nodes. In addition, each organization runs an application server which executes the client application logic to submit transactions to the blockchain, a database where the application server keeps user data (MongoDB in our prototype) and a web server that hosts a frontend application that allows users to interact with the application server. The system also runs a certificate authority that issues TLS certificates. These certificates are used by users and system nodes to secure communications.

To make our prototype scalable and easily portable on a production environment, each component of the system runs on a Docker container inside a Kubernetes Pod that is managed by a Kubernetes Deployment. Each Pod is exposed to the remaining components of the cluster through a specific Kubernetes Service. Each stateful component stores its data within Kubernetes Volumes in order to ensure data availability and fault tolerance.

The core of the system's business logic is represented by a smart contract. This smart contract was implemented using the Node.js Fabric SDK. Figure 9 shows a simplified class diagram of the smart contract.

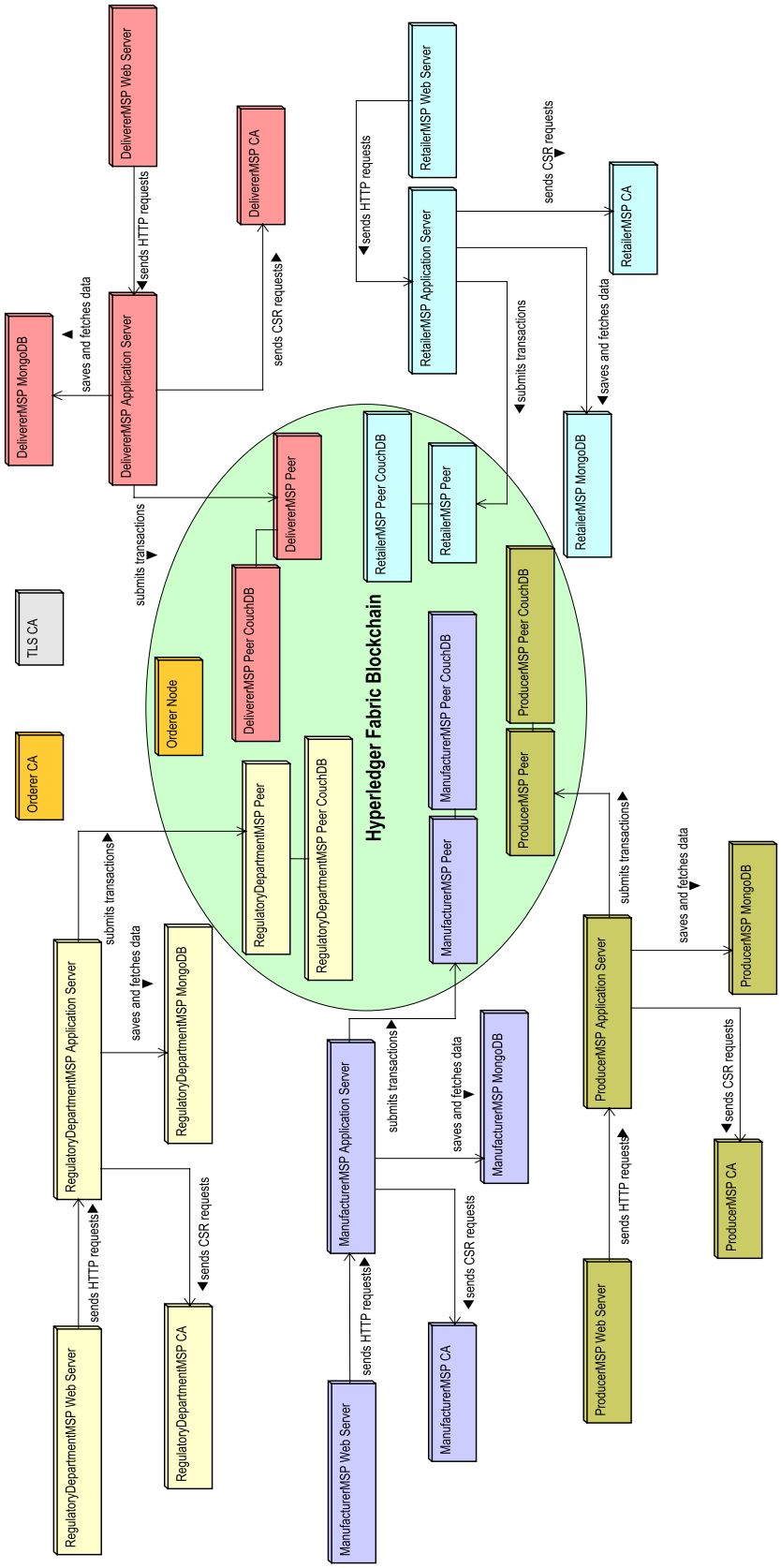


**Fig. 8** System components architecture

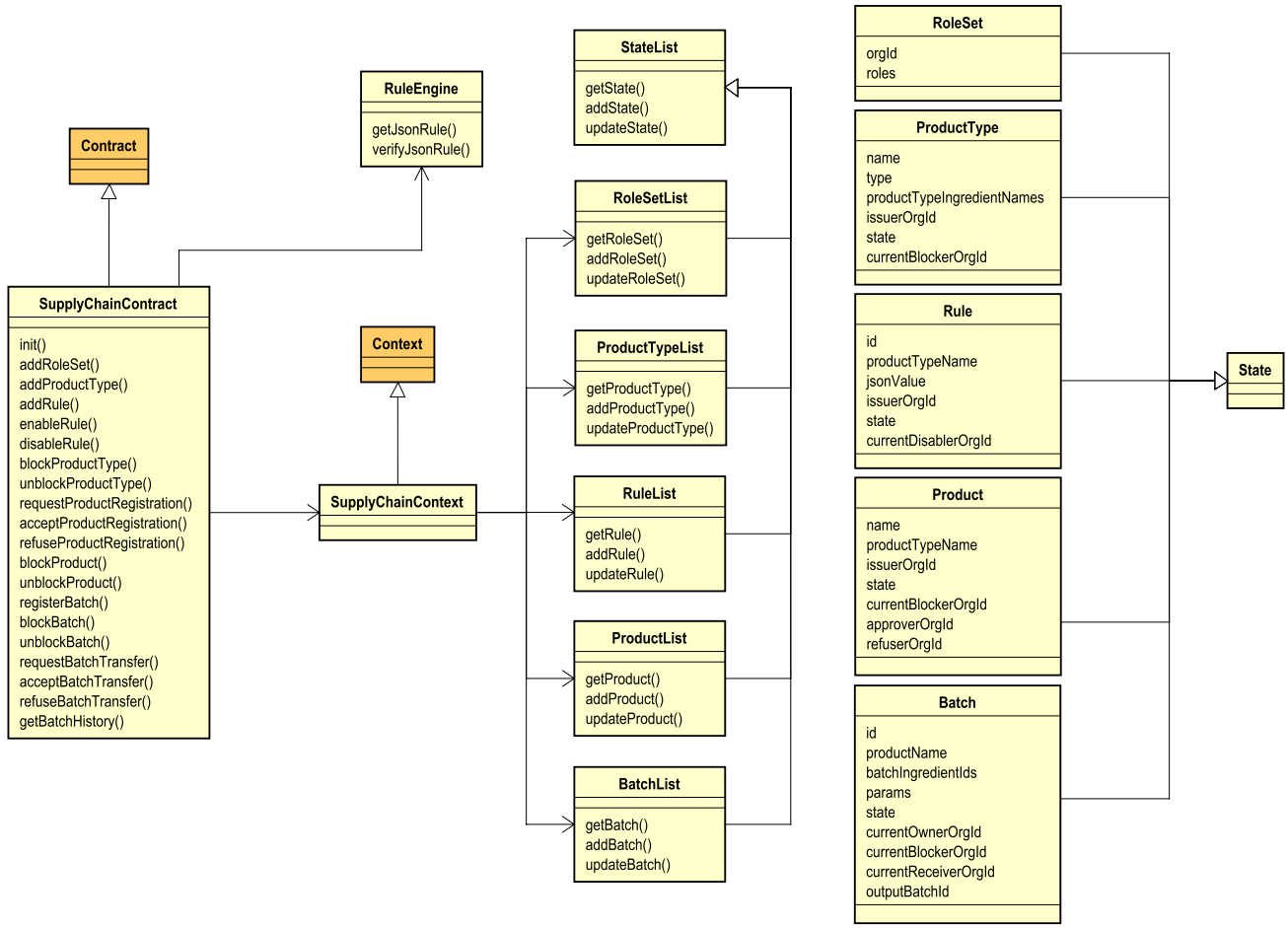


**Fig. 9** Smart contract class diagram

The *SupplyChainContract* class extends the *Contract* class, which is part of the SDK, and represents a controller class for the smart contract itself. Indeed, this class implements methods that, except for the *init()* method, represent the smart contract operations that have been illustrated previously in Sect. 4.2. These operations allow users to create and modify resources in the blockchain ledger. The *init()* method is the first method of this class that is invoked as soon as the smart contract is deployed and allows to inizialize it. Considering the scenario illustrated in Sect. 3, in the *init()* method a new role set is created that associates the *RegulatoryDepartment* role to the RegulatoryDepartmentMSP organization. The *SupplyChainContract* class has a reference to an object of the *SupplyChainContext* class, which extends the SDK *Context* class. This object allows to read and modify the ledger state and to retrieve information about a transaction, such as the identity of the user who submitted that transaction. More specifically, it has a reference to the *RoleSetList*, *ProductTypeList*, *RuleList*, *ProductList* and *BatchList* classes. These classes extend the *StateList* class and represent repositories that allow to create, modify and retrieve objects of the *RoleSet*, *ProductType*, *Rule*, *Product* and *Batch* classes respectively. These latter classes extend the *State* class and represent an abstraction layer to interact with the corresponding resources in the ledger. An object of the class *RoleSet* represents an instance of the role set resource type and has the following fields:

- *orgId*: the identifier of the organization the role set is associated with.
- *roles*: the list of roles associated with the organization identified by *orgId*.

An object of the class *ProductType* represents an instance of the product type resource type and has the following fields:

- *name*: the product type name.
- *type*: the type of the product type (*primary* or *derived*).
- *productTypeIngredientNames*: the list of product types ingredients of the product type.
- *issuerOrgId*: the organization that registered the product type.

- *state*: the current state of the product type.
- *currentBlockerOrgId*: the identifier of the last organization that blocked the product type.

An object of the class *Rule* represents an instance of the rule resource type and has the following fields:

- *id*: the rule identifier.
- *productTypeName*: the name of the product type the rule is associated with.
- *jsonValue*: the rule expression encoded as a json object.
- *issuerOrgId*: the organization that registered the rule.
- *state*: the current state of the rule.
- *currentDisablerOrgId*: the identifier of the last organization that disabled the rule.

An object of the class *Product* represents an instance of the product resource type and has the following fields:

- *name*: the product name.
- *productTypeName*: the name of the product type the product is associated with.
- *issuerOrgId*: the organization that registered the product.
- *state*: the current state of the product.
- *currentBlockerOrgId*: the identifier of the last organization that blocked the product.
- *approverOrgId*: the identifier of the organization that approved the registration request for the product.
- *refuserOrgId*: the identifier of the organization that refused the registration request for the product.

An object of the class *Batch* represents an instance of the batch resource type and has the following fields:

- *id*: the identifier of the batch.
- *productName*: the name of the product the batch is associated with.
- *issuerOrgId*: the organization that registered the batch.
- *state*: the current state of the batch.
- *currentOwnerOrgId*: the identifier of the current owner organization of the batch.
- *currentBlockerOrgId*: the identifier of the last organization that blocked the batch.
- *currentReceiverOrgId*: the identifier of the last organization that requested a transfer for the batch.
- *outputBatchId*: the identifier of the output batch for which this batch has been used as an ingredient.

Finally the *SupplyChainContract* class has a reference to the *RuleEngine* class which implements the *getJsonRuleFromString()* and *verifyJsonRule()* methods. The first is called during the execution of the *addRule()* method of the *SupplyChainContract* class and starting from the string representation of a rule, validates the rule string format and returns the corresponding JSON object of that rule which then is stored in the ledger. The latter is called during the execution of the *registerBatch()* method of the *SupplyChainContract* class and validates a rule on the parameters of a batch at the time of its registration.

## Use Cases

This section illustrates an example of the usage of our system in the context of the scenario presented in Sect. 3 and depicted in Fig. 1: in that scenario five different organizations (one for each of the defined role) are considered.

Two use cases are presented: the first one shows a simple success scenario where all operations succeed while the second one shows an alternative scenario where some operations fail. The main goal of these use cases is to demonstrate the system's ability to automate supply chain operations, to maintain traceability information and to provide a complete life cycle history of each batch.

The resources created during the execution of both use cases are quite the same. The difference between the two use cases is that some of the resources follow different state transitions. Figure 10 shows a diagram of the different resources considered in the two use cases and shared between them, also including their respective attribute values and relationships. From the figure, it can be seen that the *orange-juice* product type derives from the *orange* and *sugar* primary product types. For each product type, a product is created (*orangeX*, *sugarX* and *orange-juiceX* products) and for each product a batch is registered, where the *orange-juiceX:1* batch derives from the *orangeX:1* and *sugarX:1* batches.

Figures 11, 12, 13, 14 and 15 show the state transitions of the different resource types on the system triggered by the execution of the first use case's steps, which are numbered from 0 to 29 (both in the text description and in the Figures). For the sake of simplicity, whenever a step involves a similar transition on different resource types, only one diagram has been represented.

In this first use case, when the smart contract is initialized, the *Regulatory Department* role is associated with the RegulatoryDepartmentMSP organization (*step 0* in Fig. 11). With this role the RegulatoryDepartmentMSP organization can perform administrative operations.

The RegulatoryDepartmentMSP organization registers the *orange* primary product type (*step 1* in Fig. 12). This product type is initially in the *Blocked* state: in this state no organization can request the registration for a product related to this product type. Then the RegulatoryDepartmentMSP organization unblocks the *orange* product type causing it to pass to the *Unblocked* state (*step 2*). In the same way the RegulatoryDepartmentMSP organization registers the

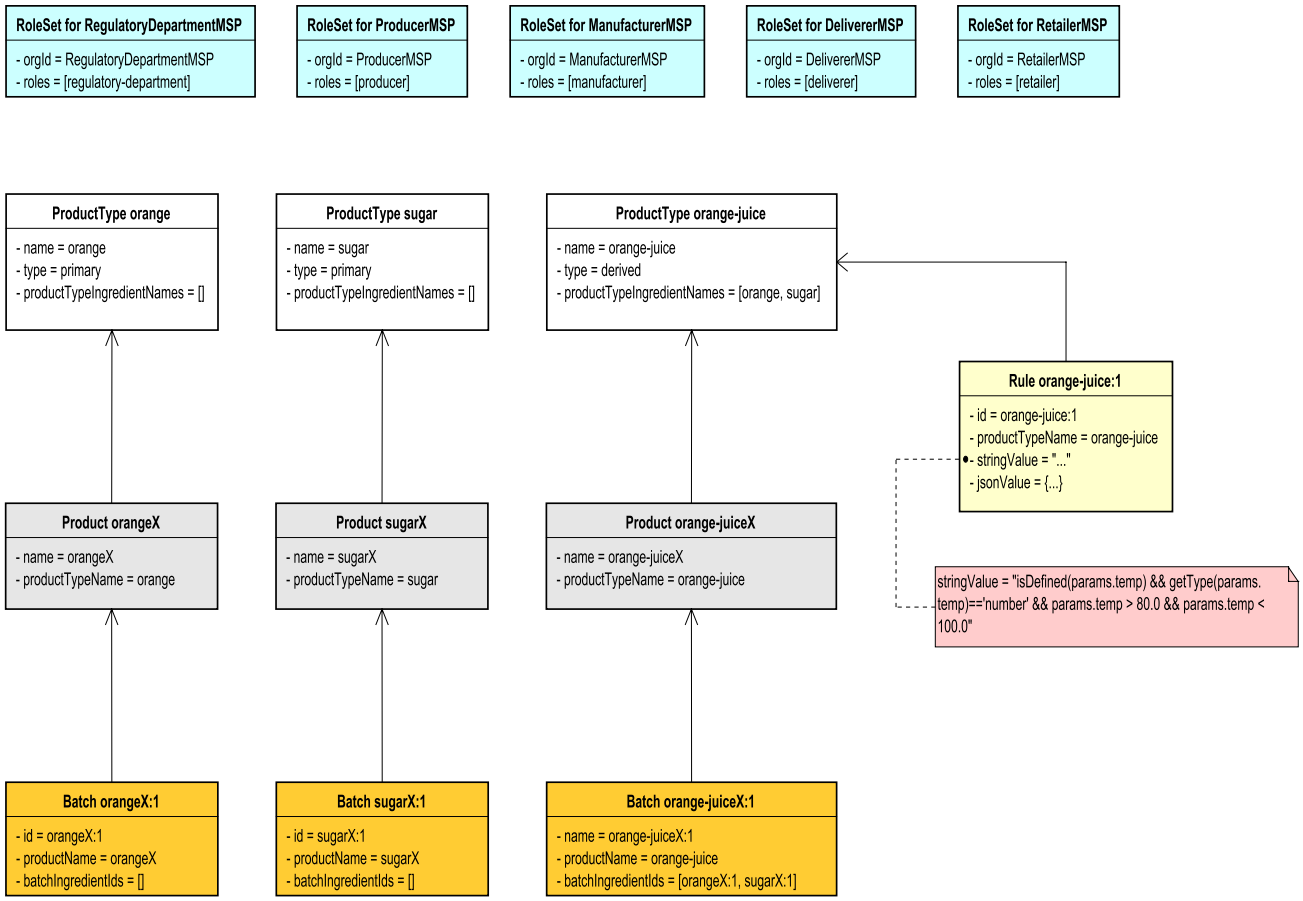


**Fig. 10** Use cases resource diagram

*sugar* primary product type (*step 3*) and unblocks it (*step 4*). After registering the *orange* and *sugar* product types, the RegulatoryDepartmentMSP organization can register the *orange-juice* product type, which is a derived one, specifying the two primary product types as ingredients (*step 5*). Once again the *orange-juice* product type starts from the *Blocked* state and after the RegulatoryDepartmentMSP unblocks it, this product type passes to the *Unblocked* state (*step 6* in Fig. 12).

The RegulatoryDepartmentMSP organization is enabled to register rules to impose constraints on the productions process: in this use case we assume that it registers a new rule associated with the *orange-juice* product type (*step 7* in Fig. 13). This rule is identified by the *orange-juice:1* id and requires that batches related to this product type contain, among the parameters, a thermal processing temperature parameter with a value that must fall within the range between *80.0* and *100.0*°*hboxC*. The rule is initially in the *Disabled* state, meaning that it is not yet activated for batch validation. The RegulatoryDepartmentMSP organization enables then the rule causing it to pass to the *Enabled* state (*step 8*).

The RegulatoryDepartmentMSP organization then associates the ProducerMSP, ManufacturerMSP, DelivererMSP and RetailerMSP organizations with the roles of *Producer*, *Manufacturer*, *Deliverer* and *Retailer* respectively (*steps 9, 10, 11, 12* in Fig. 11). In this way the ProducerMSP and ManufacturerMSP organizations can request the registration for a primary and derived product respectively. Moreover the DelivererMSP organization can buy and resell batches from and to other organizations and the RetailerMSP organization can only buy batches.

After having gained the role of *Producer*, the ProducerMSP organization requests the registration of the *orangeX* product associated with the *orange* product type (*step 13*, in Fig. 14). This product is initially in the *Pending* state and after the RegulatoryDepartmentMSP organization accepts the registration request it passes to the *Unblocked* state (*step 14*). In the same way the ProducerMSP organization requests the registration of the *sugarX* product associated with the *sugar* product type (*step 15*) and the RegulatoryDepartmentMSP organization accepts the relative registration request (*step 16*). The ManufacturerMSP organization, after having gained the role of *Manufacturer*, requests the registration of

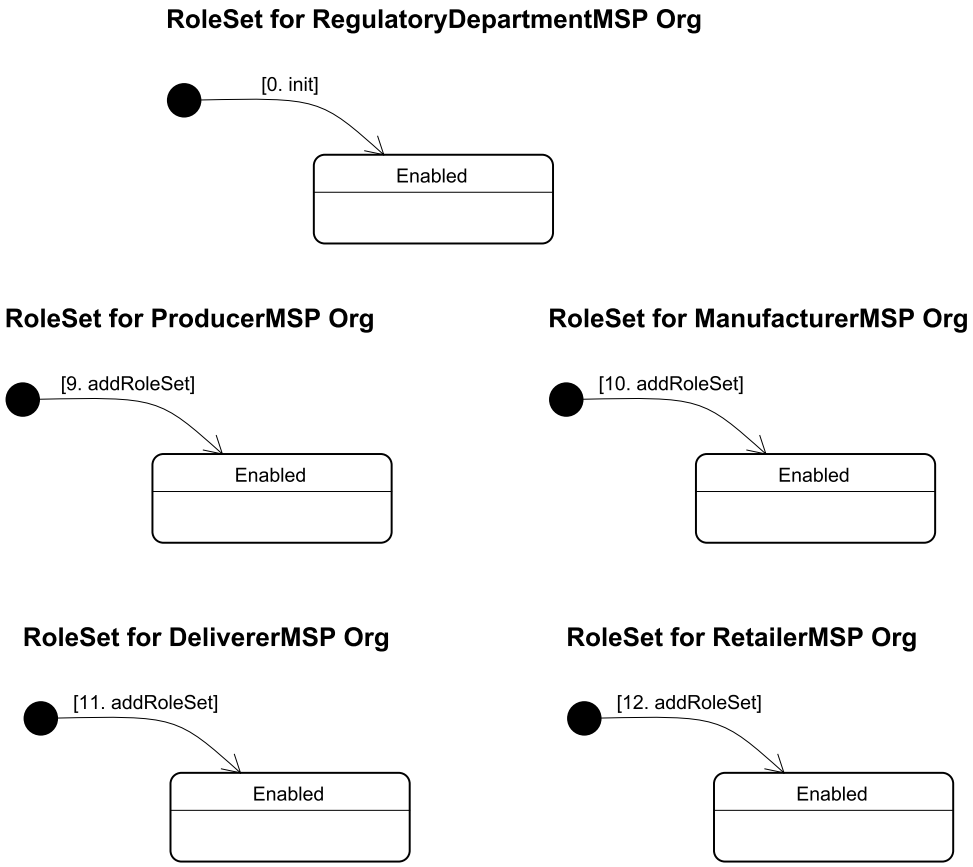


**Fig. 11** UC1 state diagram of resource type RoleSet

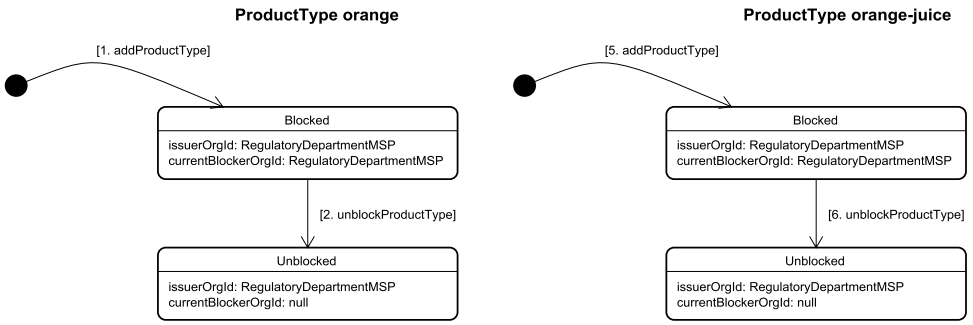


**Fig. 12** UC1 state diagram of resource type ProductType

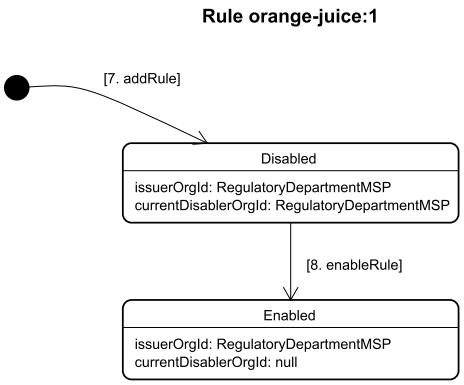


**Fig. 13** UC1 state diagram of resource type Rule

the *orange-juiceX* product associated with the *orange-juice* product type (*step 17* in Fig. 14). Once again, this derived product is initially in the *Pending* state and after the RegulatoryDepartmentMSP organization accepts the registration request it pass to the *Unblocked* state (*step 18*).

The ProducerMSP organization then registers two batches associated with the *orangeX* and *sugarX* products respectively (*steps 19, 20* in Fig. 15). These batches are identified by the *orangeX:1* and *sugarX:1* respectively and they are initially in the *Unblocked* state. The ManufacturerMSP organization submits a transfer request for the *orangeX:1* batch causing it to pass to the *Pending* state (*step 21* in Fig. 15). After the ProducerMSP organization accepts the transfer

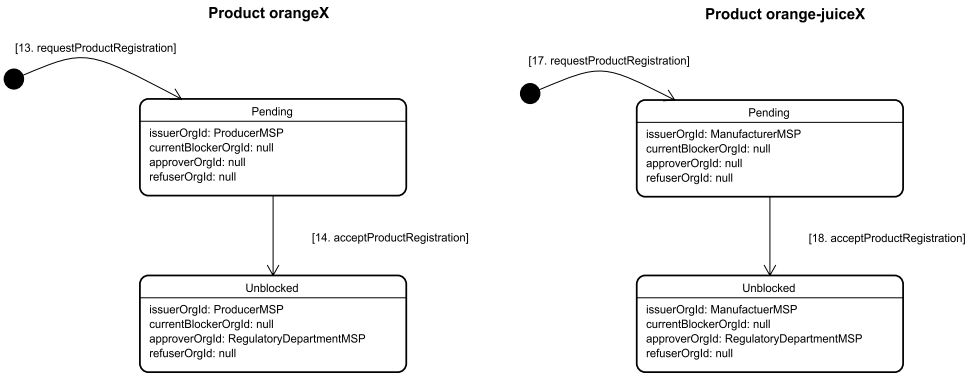


**Fig. 14** UC1 state diagram of resource type Product

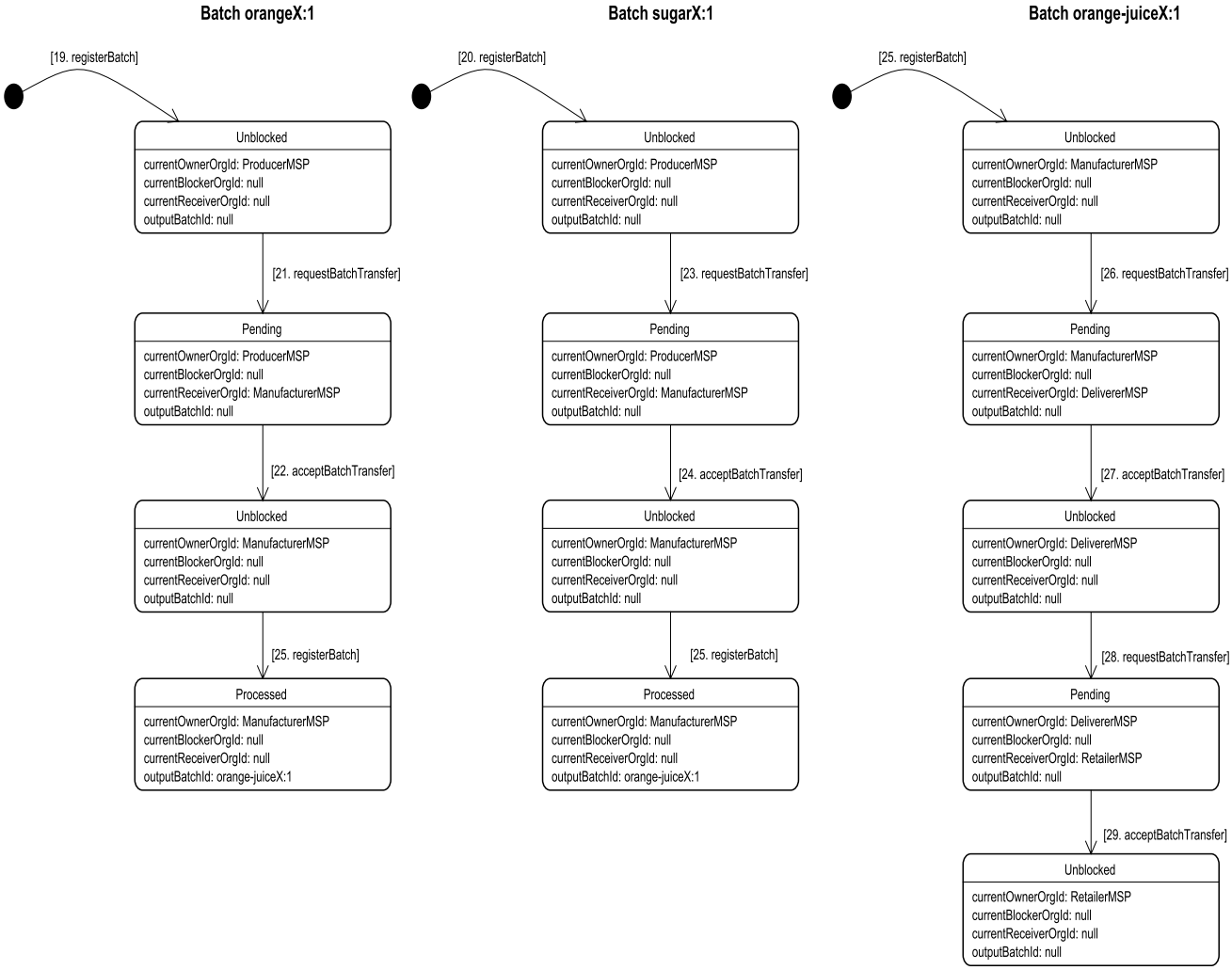


**Fig. 15** UC1 state diagram of resource type Batch

request, the ManufacturerMSP organization becomes the new owner of that batch and it returns to the *Unblocked* state (*step 22*). In the same way the ManufacturerMSP organization submits a transfer request for the *sugarX:1* batch (*step 23*) and becomes the new owner of that batch after the ProducerMSP organization accepts the request (*step 24*). The ManufacturerMSP organization can then register a batch associated with the *orange-juiceX* product, using the new acquired batches which pass to the *Processed* state (*step 25* in Fig. 15). The new registered batch is identified by the

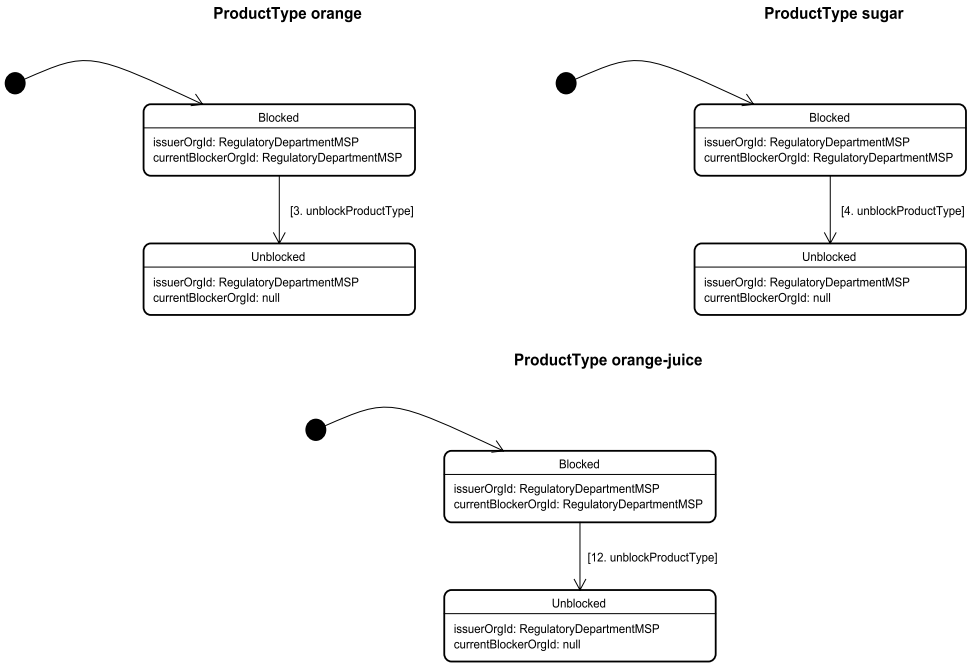


**Fig. 16** UC2 state diagram of resource type ProductType

*orange-juiceX:1* id. In the registration operation the ManufacturerMSP organization specifies a value of *90.0°hboxC* for the thermal processing temperature parameter, a value that is compliant with the range specified in the *orange-juice:1* rule.

The DelivererMSP organization then submits a transfer request for the *orange-juiceX:1* batch and the ManufacturerMSP organization accepts the request causing the DelivererMSP organization to become the new owner (*steps 26, 27*). Finally, in the same way, the RetailerMSP organization submits a transfer request for the same batch: the DelivererMSP organization accepts the request causing the RetailerMSP organization to become the new owner (*steps 28, 29* in Fig. 15).

Let us now describe the second use case, where an alternative flow is considered. As in the first use case, we show in Figs. 16, 17 and 18 the state transitions of the product type, product and batch resource types triggered by the execution of the involved steps (numbered from 1 to 35). State diagrams for role set, product type and rule resource types are not shown because the respective state transitions are similar with the ones of the first use case.

The starting point in the description of this second use case is that the *orange*, *sugar* and *orange-juice* product types have been registered by the RegulatoryDepartmentMSP organization but they are still in the *Blocked* state. Furthermore the same rule of the first use case has been associated with the *orange-juice* product type and for each of the organizations of the system a role set has been registered.

Initially the ProducerMSP organization requests the registration of the *orangeX* and *sugarX* products associated with the *orange* and *sugar* product types respectively (*steps 1, 2*). However, this requests fail because the *orange* and *sugar* product types are still in the *Blocked* state and this prevent to register products (and the corresponding batches) related to these product types. After the RegulatoryDepartmentMSP organization unblocks the two product types (*steps 3, 4* in Fig. 16), the ProducerMSP organization retries to execute the requests and, since this time the two products are already registered on the system, they are accepted and pass to the *Pending* state (*steps 5, 6* in Fig. 17). Then the ProducerMSP organization also requests the registration of the *orangeY* product associated with the *orange* product type (*step 7*). The RegulatoryDepartmentMSP organization accepts the registration requests for the *orangeX* and *sugarX* products causing them to pass to the *Unblocked* state (*steps 8, 9* in Fig. 17), but, let us assume that it refuses the registration request for the *orangeY* product causing it to pass to the *Refused* state (*step 10*). Similarly, the ManufacturerMSP organization requests the registration of the *orange-juiceX* product associated with the *orange-juice* product type, but the request fails because the *orange-juice* product is in the *Blocked* state (*step 11*). After the RegulatoryDepartmentMSP organization unblocks the *orange-juice* product type (*steps 12*), the ManufacturerMSP organization retries to execute the request and this time it is accepted and the product is registered on the system, starting from the *Pending* state (*step 13*). Then the RegulatoryDepartmentMSP organization accepts the registration request for the product causing it to pass to the *Unblocked* state (*step 14*).

The ProducerMSP organization then registers two batches associated with the *orangeX* and *sugarX* products

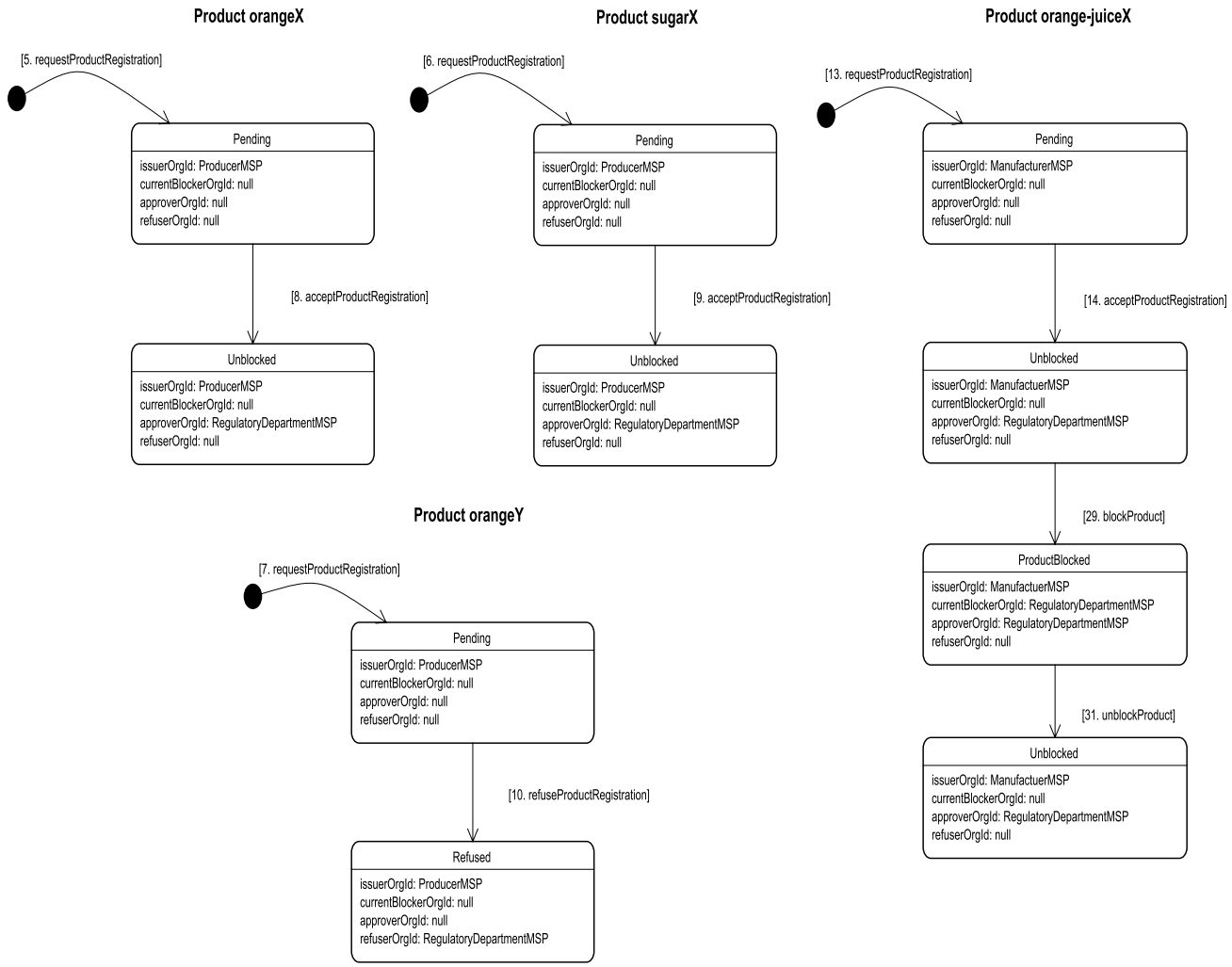


**Fig. 17** UC2 state diagram of resource type Product

respectively (*steps 15, 16*). These batches are identified by the *orangeX:1* and *sugarX:1* ids respectively and, initially, are in the *Unblocked* state. Then the RegulatoryDepartmentMSP organization decides to block temporarily the two batches to execute some quality control tests on them, causing the batches to pass to the *BatchBlocked* state (*steps 17, 18* in Fig. 18). In the meantime the ManufacturerMSP organization submits two transfer requests for the two batches, but they are automatically rejected because the batches are in the *BatchBlocked* state (*steps 19, 20*). After the quality control tests complete successfully the RegulatoryDepartmentMSP organization unblocks the two batches (*steps 21, 22*). Then the ManufacturerMSP organization retries to submit the transfer requests for the two batches: this time they are accepted, causing them to pass to the *Pending* state (*steps 23, 24*). After the ProducerMSP organization accepts the transfer requests, the ManufacturerMSP organization becomes the new owner of the requested batches and these return to the *Unblocked* state (*steps 25, 26*).

Let us assume now that the ManufacturerMSP organization tries to register a batch associated with the *orange-juiceX* product, using the new acquired batches as ingredients and specifying the value *60.0°hboxC* for the batch parameter *temp* (*step 27*). When the registration request is submitted, the rule *orange-juice:1* gets activated and the request is rejected because the rule condition is not met. The ManufacturerMSP organization then retries the request specifying the value *85.0 °hboxC* for the batch parameter *temp* and this time the request is accepted causing the two batches ingredients to pass to the *Processed* state: a new batch with *orange-juiceX:1* id is created that starts from the *Unblocked* state (*step 28*). After noticing the first batch registration request was not accepted due to the fact that the rule condition was not met, the RegulatoryDepartmentMSP organization decides to temporarily block the *orange-juiceX* product to carry out some checks (*step 29*). This causes the *orange-juiceX* product and the *orange-juiceX:1* batch to pass to the state *ProductBlocked*.

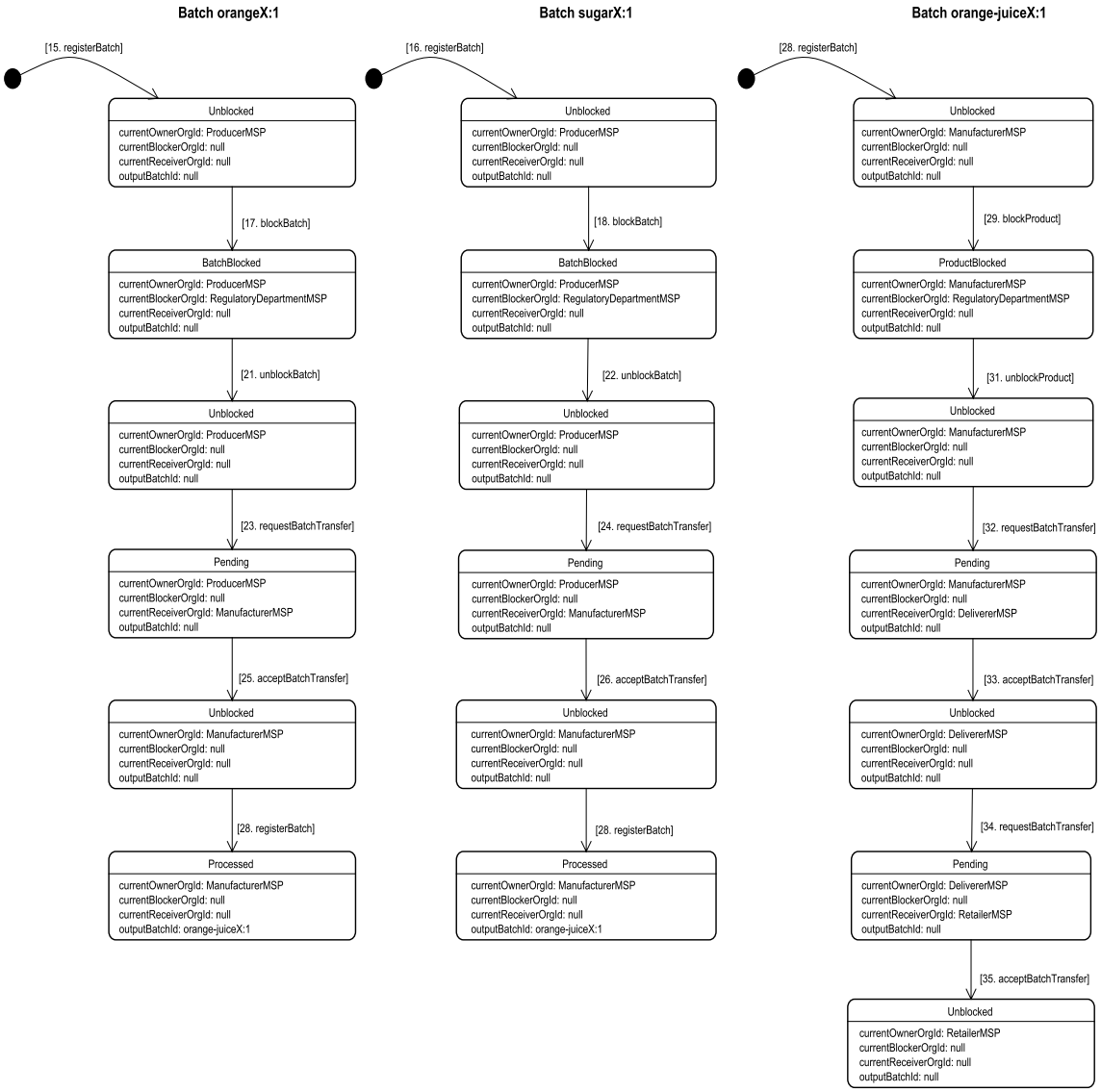


**Fig. 18** UC2 state diagram of resource type Batch

In the meantime, the DelivererMSP organization submits a transfer request for the *orange-juiceX:1* batch but the request is rejected because it is now in the *ProductBlocked* state (*step 30*). After the checks complete successfully, the RegulatoryDepartmentMSP organization unblocks the *orange-juiceX* product causing it and the *orange-juiceX:1* batch to pass to the *Unblocked* state (*step 31*). Then the DelivererMSP organization retries to submit the transfer request for the batch causing it to pass to the *Pending* state (*step 32*). After the ManufacturerMSP organization accepts the transfer request, the DelivererMSP organization becomes the new owner of the requested batch and this returns to the *Unblocked* state (*step 33*). Finally, in the same way as the first use case, the RetailerMSP organization submits a transfer request for the same batch and the DelivererMSP organization accepts the request causing the RetailerMSP organization to become the new owner (*steps 34, 35*).

The realization of the use cases, despite their simplicity, demonstrates that the proposed system supports all the basic lifecycle management steps of an agri-food product, from its origin to the end consumer. The simple rule-based system embedded within the framework, also show the flexibility of the framework: it is possible to add any kind of rule at runtime to cope with specific quality control strategies needed by any of the organization involved in the supply chain.

## Conclusions

A complete model of a blockchain-based agri-food supply chain traceability system has been proposed in this work, also showing a prototype implementation. The system is based on the Hyperledger Fabric permissioned blockchain, a category of blockchain where participation is limited to a well-defined set of members. This type of blockchain fits well in the context of agri-food supply chains because typically only a limited set of organizations can participate in supply chain operations. By using blockchain technology, supply chain traceability data can be stored in a more transparent and reliable way with respect to using a centralized entity. Furthermore, as a fully distributed system, blockchain mitigates the problems of limited scalability and single point of failure. The proposed system allows to automate supply chain management operations with the use of the smart contract primitive and maintain traceability information in a transparent, secure and immutable way. Moreover, the system gives the possibility to add and modify rules at runtime and this allows to implement product-specific quality control mechanisms. Finally, the system provides a complete view of the different phases of harvesting, processing and distribution to which batches of product are subject allowing to reconstruct the entire life cycle of each batch, also obtaining provenance information.

Future works will involve the design of a more complex rule engine system to implement more sophisticated quality control mechanisms. At the moment the rule engine is implemented as a library used by the smart contract. Our intention is to use more complex rule engines deployed as external services that can be contacted by the smart contract. Furthermore, some experiments will be conducted to evaluate the performances of the system in terms of transaction throughput and scalability.

**Funding Information** This work has been partially financially supported by the funding programme PO FESR Sicilia 2014/2020, research project QUALIAGRO 4.0 (project no. 08CT6201000224).

## Declarations

**Conflict of interest** The authors declare that they have no competing interests



## References

1. Androulaki E, Barger A, Bortnikov V, et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference, EuroSys '18, 2018; https://doi.org/10.1145/3190508.3190538
2. Antonucci F, Figorilli S, Costa C, et al. A review on blockchain applications in the agri-food sector. J Sci Food Agricult. 2019;99(14):6129–38. https://doi.org/10.1002/jsfa.9912.
3. Baralla G, Pinna A, Tonelli R, et al. Ensuring transparency and traceability of food local products: A blockchain application to a smart tourism region. Concurrency and Computation: Practice and Experience, 2021;33(1). https://doi.org/10.1002/cpe.5857
4. Biswas K, Muthukkumarasamy V, Lum W. Blockchain based wine supply chain traceability system. In: Future Technologies Conference (FTC 2017), 2017; 56–62
5. Bosona T, Gebresenbet G. Food traceability as an integral part of logistics management in food and agricultural supply chain. Food Control. 2013;33(1):32–48. https://doi.org/10.1016/j.foodcont.2013.02.004.
6. Caro MP, Ali MS, Vecchio M, et al. Blockchain-based traceability in agri-food supply chain management: a practical implementation. In: 2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany), 2018; 1–4 https://doi.org/10.1109/IOT-TUSCANY.2018.8373021
7. Casino F, Kanakaris V, Dasaklis T, et al. Modeling food supply chain traceability based on blockchain technology. IFAC-PapersOnLine. 2019;52:2728–33. https://doi.org/10.1016/j.ifacol.2019.11.620.
8. Chen K, xin WANG X, ying SONG H,. Food safety regulatory systems in europe and china: A study of how co-regulation can improve regulatory effectiveness. J Integrative Agricult. 2015;14(11):2203–17. https://doi.org/10.1016/S2095-3119(15)61113-3.
9. Cocco L, Mannaro K, Tonelli R, et al. A blockchain-based traceability system in agri-food sme: case study of a traditional bakery. IEEE Access, 2021;9:62,899–62,915. https://doi.org/10.1109/ACCESS.2021.3074874
10. Dabbene F, Gay P, Tortia C. Traceability issues in food supply chain management: a review. Biosyst Eng. 2014;120:65–80. https://doi.org/10.1016/j.biosystemseng.2013.09.006.
11. Feng Tian. A supply chain traceability system for food safety based on haccp, blockchain internet of things. In: 2017 international conference on service systems and service management, 2017;1–6. https://doi.org/10.1109/ICSSSM.2017.7996119
12. Galvez JF, Mejuto J, Simal-Gandara J. Future challenges on the use of blockchain for food traceability analysis. TrAC Trends Anal Chem. 2018;107:222–32. https://doi.org/10.1016/j.trac.2018.08.011.
13. Gamage HTM, Weerasinghe HD, Dias NGJ. A survey on blockchain technology concepts, applications, and issues. SN Comput Sci. 2020;1(2):114. https://doi.org/10.1007/s42979-020-00123-0.
14. Kolb J, AbdelBaky M, Katz RH, et al. Core concepts, challenges, and future directions in blockchain: A centralized tutorial. ACM Comput Surv, 2020;53(1). https://doi.org/10.1145/3366370

15. Li D, Wang X, Chan HK, et al. Sustainable food supply chain management. Int J Prod Econ. 2014;152:1–8. https://doi.org/10.1016/j.ijpe.2014.04.003.
16. Malik S, Kanhere SS, Jurdak R. ProductChain: Scalable blockchain framework to support provenance in supply chains. In: NCA 2018 - 2018 IEEE 17th international symposium on network computing and applications, 2018; https://doi.org/10.1109/NCA.2018.8548322
17. Marchese A, Tomarchio O. An agri-food supply chain traceability management system based on hyperledger fabric blockchain. In: Proceedings of the 23rd international conference on enterprise information systems - Volume 2: ICEIS,, INSTICC. SciTePress, 2021; 648–658 https://doi.org/10.5220/0010447606480658
18. Marchesi L, Mannaro K, Porcu R. Automatic generation of blockchain agri-food traceability systems. In: 2021 IEEE/ACM 4th international workshop on emerging trends in software engineering for Blockchain (WETSEB), 2021;41–48. https://doi.org/10.1109/WETSEB52558.2021.00013
19. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008; https://bitcoin.org/bitcoin.pdf
20. Olsen P, Borit M. The components of a food traceability system. Trends Food Sci Technol. 2018;77:143–9. https://doi.org/10.1016/j.tifs.2018.05.004.
21. Ray Z, Xun X, Lihui W. Food supply chain management: systems, implementations, and future research. Ind Manag Data Syst. 2017;117(9):2085–114. https://doi.org/10.1108/IMDS-09-2016-0391.
22. Shahid A, Almogren A, Javaid N, et al. Blockchain-based agri-food supply chain: A complete solution. IEEE Access, 2020;8:69,230–69,243. https://doi.org/10.1109/ACCESS.2020.2986257
23. Tian F (2016) An agri-food supply chain traceability system for china based on rfid & blockchain technology. In: 2016 13th international conference on service systems and service management (ICSSSM), https://doi.org/10.1109/ICSSSM.2016.7538424
24. Wang S, Li D, Zhang Y, et al. Smart contract-based product traceability system in the supply chain scenario. IEEE Access, 2019;7:115,122–115,133. https://doi.org/10.1109/ACCESS.2019.2935873
25. Wang Y, Han JH, Beynon-Davies P. Understanding blockchain technology for future supply chains: a systematic literature review and research agenda. Supply Chain Management: An International Journal. 2018;24. https://doi.org/10.1108/SCM-03-2018-0148.
26. Wüst K, Gervais A. Do you need a blockchain? In: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT), 2018; 45–54. https://doi.org/10.1109/CVCBT.2018.00011
27. Zhao G, Liu S, Lopez C, et al. Blockchain technology in agri-food value chain management: a synthesis of applications, challenges and future research directions. Comput Ind. 2019;109:83–99. https://doi.org/10.1016/j.compind.2019.04.002.