Olive Oil as Case Study for the *-Chain Platform

Stefano Bistarelli[1], Francesco Faloci[2,3,*], Paolo Mori[2] and Carlo Taticchi[1,*]

[1] *Università degli Studi di Perugia*
[2] *Istituto di Informatica e Telematica - Consiglio Nazionale delle Ricerche*
[3] *Università di Camerino*

Abstract

Certification of product origin and supervision of supply chains are fundamental activities in today's market scenario. Hence, it is of the highest interest to develop platforms that allow domain experts to quickly and easily build supply chain management systems allowing them to trace their production/-transformation processes. This paper presents how the *-chain framework can solve this problem. In particular, we used the model and the graphical language defined by our framework to represent an olive oil supply chain, and the suite of tools we develop within such a framework to generate the related blockchain based traceability system, i.e., to automatically generate a set of solidity smart contracts implementing the system and two web interfaces to interact with them (one for supply chain administrators, the other for the actors of the production/transformation process), starting from the graphical representation.

Keywords

Supply Chain, Blockchain, Distributed Ledger Technology, Domain Specific Graphical Language, Smart Contracts, Automatic Smart Contract Generation.

1. Introduction

Supply chains are network of organizations, involved in the different processes and activities, that produce value in the form of products and services for the final buyer [6]. Depending on the specific scenario (e.g., product processing, service provisioning, warehousing) different types of supply chains can be considered. For instance, a production supply chain represents the flow of goods from the raw material to the final product. Supply chain management is therefore a crucial process to optimise the production cycle and lower the related costs. Blockchain technologies have been proven effective in developing solutions for the implementation of management systems for supply chains. Several works are currently available in this regard (like [1, 4, 5]), but these solutions are too specific for the field they were designed for. Consequently, they are not general enough to be used in alternative scenarios without major changes and adjustments, for which a relevant knowledge of the blockchain technology and expertise in smart contracts development is required. In this regard, we introduced the *-chain platform [3, 2], a platform to

DLT 2022: 4th Distributed Ledger Technology Workshop, June 20, 2022, Rome, Italy
*Corresponding author.
stefano.bistarelli@unipg.it (S. Bistarelli); francesco.faloci@unicam.it (F. Faloci); paolo.mori@iit.cnr.it (P. Mori); carlo.taticchi@unipg.it (C. Taticchi)
0000-0001-7411-9678 (S. Bistarelli); 0000-0002-6413-6834 (F. Faloci); 0000-0002-6618-0388 (P. Mori); 0000-0003-1260-4672 (C. Taticchi)

simplify the implementation of blockchain based supply chain management systems (SCMSs). Our platform provides a powerful Domain-Specific Graphical Language to represent supply chains, an easy-to-use graphical interface for authoring such representations, and a suite of tools for translating graphical models into the smart contracts building up the SCMS.

Our platform serves two main purposes: on the one hand, it provides a general solution for implementing supply chain management systems; on the other hand, it allows the supply chain administrators to implement applications distributed via blockchain technologies. In this paper, we demonstrate the capabilities of our platform, using the olive oil supply chain as a study case. In particular, we use *-chain to model all the steps required to produce marketable Olive Oil, starting from the harvesting phase up to bottling.

The rest of this paper is organized as follows: in Section 2 we introduce fundamental notions of blockchain and smart contracts; Section 3 describes *-chain functionalities, together with insights on its implementation; Section 4 summarises the Olive Oil supply chain, highlighting the main stages that the product goes through; in Section 5 we show how the considered supply chain can be modeled through our tool; Section 6, finally, summarises and concludes the paper, also pointing out new directions for future work.

2. Background

Distributed Ledger Technology (DLT) refers to systems and protocols that allow simultaneous access, validation, and updating with immutable data across a network. In simple words, the DLT is all about the idea of a ”decentralized” network against the conventional monolithic centralized mechanism. Blockchain Technology (BT) is a special case of DLT, focusing on industries and financial sectors. The BT offers great potential to foster various sectors with its unique combination of characteristics as decentralization, immutability, and transparency. So far, the most prominent attention the technology received was through news from industry and media about the development of cryptocurrencies (such as Bitcoin[1], and Monero[2]), which all are having remarkable capitalization. BT, however, is not limited to cryptocurrencies; there are already existing blockchain based applications in industry and the public sector. Also, BT can have applications on non-financial sector, such as traceability problems and workflow organization. A smart contract is a self-executing contract (script) with the terms of the agreement between two actors, generally a buyer and a seller, directly written into lines of code. The code and the agreements contained in the script exist across a distributed decentralized blockchain system. One of the most popular coding languages for describing smart contracts is Solidity[3], widely used for Ethereum[4] systems.

[1]Bitcoin Project: https://bitcoin.org
[2]Monero project: https://www.getmonero.org
[3]Solidity white paper: https://docs.soliditylang.org/en/v0.8.6/
[4]Ethereum project: https://ethereum.org/en/

3. *-chain framework

The *-chain framework [2] consists of a set of tools designed to aid the development of blockchain-based SCMSs. In particular, the end-user is provided with a web interface built upon a Domain-Specific Graphical Language (DSGL) that allows for tracing supply chains on the blockchain [3]. The framework translates the representation specified by the user into a set of smart contracts that are then used to implement an SCMS within the blockchain. The main purpose of *-chain is to enable experts in the context of a specific supply chain process (such as olive oil production) to contribute in the definition of SCMSs that can be distributed via blockchain, even with little or no knowledge of DLT. Therefore, our framework decouples the distinct tasks of supply chain process design and smart contract programming, which can be assigned to different professionals in the two respective fields.

The web interface provided by the *-chain framework integrates a graphical editor that can be used by supply chain domain experts to design SCMSs through a dedicated DSGL, equipped in turn with primitives that allow representing the most common types of supply chains. The editor also produces a textual, JSON-formatted, representation of the specified supply chain model, which is used to generate the smart contracts in the final SCMS. Those smart contracts corresponds to asset types belonging to a given supply chain and are endowed with functions that trace the execution of operations supported by the various assets.

4. Olive oil supply chain

The olive oil production process requires a series of steps that we can summarise in the following main phases:

- agricultural maintenance
- preparation of the olives
- crushing, kneading
- extraction
- separation
- conservation

During each of those steps, various parameters are monitored to guarantee the quality of the final product. Olive oil is, indeed, subjected to quality controls that determine its commercial classification[5]. For example, the temperature reached by the olive paste and the degree of oxidation of the oil can alter the final result, both in terms of taste/odour and chemical properties. Exceeding the imposed thresholds can cause, in addition to unpleasant smells and tastes, also harmful effects on the human body. In this section, we illustrate the whole olive oil supply chain, also providing, for demonstration purposes, insights into the agricultural maintenance phase.

The **agricultural maintenance** (for which we show a flow chart in Figure 1) constitutes the first phase of the olive oil supply chain and has a major impact on the organoleptic characteristics

[5]Olive oil classification follows the EEC regulation 2568/91 and subsequent amendments (656/95 and 2472/97), and the commercial standard of the International Olive Committee (Norma COI, 1998).

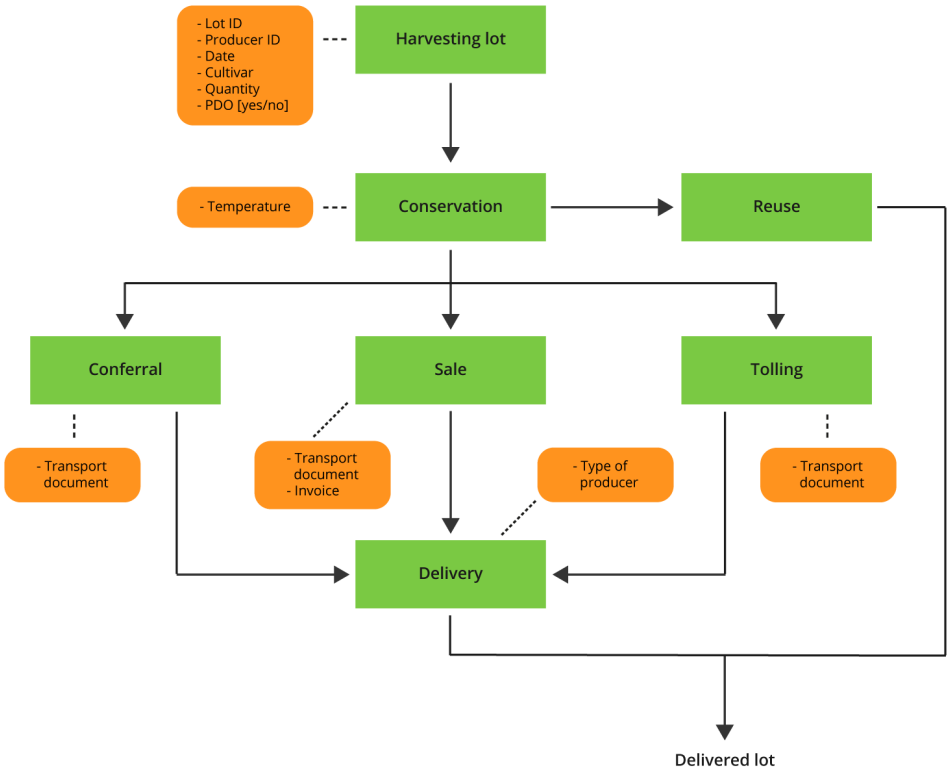


Figure 1: Flowchart fragment of the olive oil supply chain showing the stages of agricultural maintenance (green rectangles) along with the properties to be recorded (in orange with rounded edges).

of the final product. In this phase, olives are harvested, transported and then either sold or used in the following phases. Before the **processing** phase, olives undergo a series of operations, such as, for example, weighing, husking (elimination of foreign material), selection (division according to healthiness and size) and washing. Between weighing and husking there can also be short storage of the olives (maximum 24 hours) following the indications given for conservation in the agricultural phase. The weighing is carried out in the oil mill at the time of delivery. As for washing, this is done either by immersion or with special washing machines that maintain a forced movement of the water. In the **crushing** phase, the olive cell wall is broken in order to release juices. The crusher (with discs, hammers or knives) is mostly used in continuous cycle systems because it better responds to automation needs. The loading is carried out mechanically and the unloading takes place from the bottom, again mechanically, with the pouring of the oil paste into the kneading machines. The processing can take place in a very short time, and with a minimum footprint. **Kneading** is the phase in which oil drops aggregate and grow in volume. During this process, the olive paste coming from the crushing is slowly stirred to ease the separation of the oily component from the aqueous one. The temperature is an important parameter to monitor, as the pasta should reach a maximum of around 25 $^{\circ}C$. Another discriminating factor in the final result is the kind of used machinery, which varies

from traditional millstones to modern malaxers. The **extraction** process consists in separating oil, water and solid parts in the kneaded paste. The factors that most affect the final result during this phase are again the temperature and the time needed to conclude the operation. Moreover, three main types of extraction technology can be used, each deriving from a different physical principle: pressure, percolation and centrifugation. The oil obtained after the extraction phase often contains a minimum percentage of water, which undergoes a further **separation** process, which can be carried out either through decantation or centrifugation. The final product will have a different yield depending on the methodology used. At this stage, the oil can already be consumed and it is stored in containers where residues settle on the bottom to make the product clearer. If the oil is intended for marketing, it is subjected to **filtration** and **bottling**. Filtration removes solid and colloidal impurities from the oil and can be performed by using different techniques. For bottling, hygiene and storage regulations must be taken into account.

To better demonstrate the functioning of the methodology we propose, we describe all the mandatory and optional steps involved in the **agricultural maintenance** phase. As we can see from Figure 1, this phase begins with the harvesting of olives from trees and the creation of the corresponding lot. The information to monitor regards which cultivars are grown, as well as the harvesting date and the quantity of olives in the lot. Furthermore, it is specified whether the lot contains a PDO (Protected Designation of Origin) production, together with the owner's data. After the harvest, the olives can be stored in cool, well-ventilated rooms for a period of time that does not exceed 2 or 3 days. The temperature of the room must be controlled since it can compromise the quality of the product. The olives, then, can be either directly reused by the same owner or transferred to third parties. In the latter case, there are three possible options: sale, tolling and conferral. For tolling and conferral a transport document is mandatory, while in the case of a sale, such a document can be replaced by a regular invoice. Finally, when the olives are delivered, we are interested in the type of producer to which the asset is transferred.

5. Supply chain model translation

In order to translate the control process of the Olive Oil supply chain, we must first adapt the various phases of the flow chart described in Section 4. The first step is to translate the components of each phase into a logic block paired with the DSGL's blocks of the *-chain framework. We have to identify the relevant objects of the workflow with respect to the focus of the analysis: the aim of this analysis is to supervise the Olive Oil, tracing the various production processes in order to accredit the origin of the goods. Each element we want to track is identified as an asset, paired with a blue rectangle shape. Each feature of this asset represents a relevant detail for the supply chain representation meaning. These features are paired with small grey circles attached to the blue shape of the asset itself: each circle is a property of the asset. Lastly, each activity applied to an asset is represented as operation: these operations are paired with a pointed arrow that links an asset to another object on the schema; the shape of the arrow defines the type of the operation.

Figure 1 shows a subsection of the workflow of olive oil supply chain. Figure 2 represents the translated schema: this schema is drawn with the *-chain framework, using one of the main tools of the platform: the design interface.

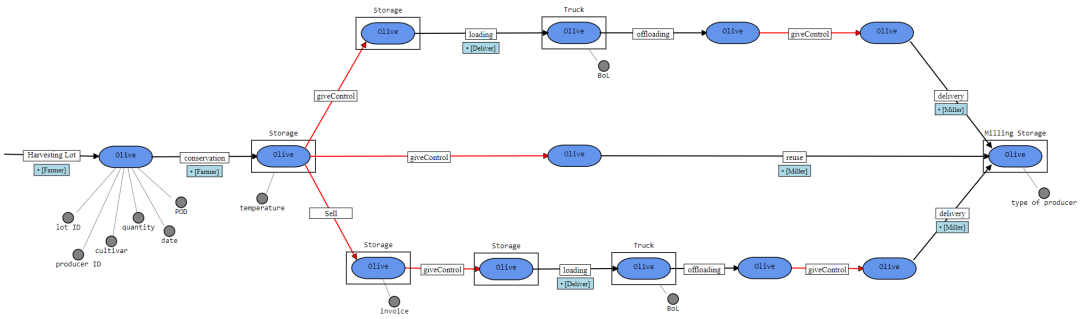


Figure 2: Representation of Olive oil supply chain with the *-chain framework.

5.1. Supply chain model representation

The first asset –the blue icon in the leftmost side of the schema– is the "*Olive*", which is a countable and consumable asset; this asset does not have any incoming arrows from another asset: this could mean that: *i)* this is the point of origin of any asset *Olive*, or *ii)* the production of this asset is not tracked using this supply chain schema. In both cases, the asset *Olive* is generated at this point of the schema, using an *asset_create()* operation, labelled as "Harvesting Lot". Under this create operation the label "*Farmer*" is specified, which means that the creation of an *Olive* asset can only be performed by a user paired with the "*Farmer*" role. The asset *Olive* is also defined with a set of properties, which are: "lot ID", "producer ID", "date", "cultivar", "PDO" and "quantity". We can consider the *Olive* asset as the observation element of the supply chain. The second element of the schema is still a *Olive* asset, wrapped into a "*Storage*" container. This container should represents a room, a warehouse, or a silos, thus a generic location where the *Olive* asset is stored. The "*Storage*" container is a non-consumable container. The container *Storage* is defined with only a property: "temperature". The two sides of these *Olive* assets are connected with a black arrow representing the operation "*conservation*": *conservation* represents the conservation process of the **agricultural maintenance** phase. The "*conservation*" is an "asset_pack()" operation. This type of operation represents the process to transfer an asset into a specific item defined as container. Also under the "*conservation*" operation the label for the permission role is specified: even in this case, the operation can be performed only by a user paired with the "*Farmer*" role.

The next step is divided into three different paths drawn in the diagram. Each direction represents a different procedure: "Conferral", "Tolling", "Sale" and "Reuse". The Conferral and Tolling procedures are identical under the procedural aspect, they differ only on their names. For the sake of simplicity these two procedures will be represented with the same path. Each of the different paths represents a plausible process choice. As it is described in the specifications of the **agricultural maintenance** phase, it is possible for an *Olive* asset to be sold to another user, transferred for processing to another user while maintaining the original owner, or simply reused for a subsequent phase.

The path that starts with a red pointed arrow oriented to the top right side is representing the Conferral and Tolling procedures. This path starts with a "*giveControl*" operation, establishing

the change of controller from a user with the "Farmer" role to a user with the "Deliver" role. The *giveControl* operation does not modify any other property of the asset *Olive*. through this operation, only the controller of the *Olive* asset is changed, meanwhile the owner is still a user with "Farmer" role. Following this path the *Olive* asset is loaded into a vehicle for the delivering: this procedure is represented as following: a "*Truck*" container represents the vehicle for the delivering procedure. The *Truck* is a non-consumable container with the bill of lading ("BoL") descriptor as the only property. The "*loading*" operation is declared as "asset_flow()" -transferring an asset from a package to another-, loading the *Olive* into the *Truck*: this operation has a role constraint on user with "Deliver" role. Hence, at the destination, the asset *Olive* is unloaded from the *truck*; its controller is also changed from "Deliver" to "Miller". These changes of the asset are represented by two arrows: the black arrow with the label "*offloading*" represents the asset being unloaded from the vehicle and placed in the "Miller"; the red arrow with the label "*giveControl*" represents the change of controller to a user with the "Miller" role. The user with the "Miller" role is the one that could load the asset *Olive* into the "*Milling Storage*". This storage is a non-consumable container, and is the starting point for the **processing** phase. Last operation on this path is the "*delivery*" one. This operation is an "asset_pack()" that store the *Olive* asset into the *Milling Store*.

The path that starts with a red pointed arrow oriented to the bottom right side is representing the Sale procedure. This path starts with a "*sell*" operation, establishing the change of Owner from a user with the "Farmer" role to a user with the "Miller" role. The *sell* operation modifies the owner and the controller of the asset *Olive*. In the same way as the previously described path, the *Olive* asset is loaded into a vehicle for the delivering. After the *sell* operation, this scenario follows exactly the same procedure, ending with the deliver transaction. Therefore -for simplicity- we do not describe again the exact operations performed in this short step, since they are also performed in the same order. Also, the last operation on this path is -again- the "*delivery*", following the same constraints.

The third path to be represented is that of reuse: in this step of the **agricultural maintenance** phase an *Olive* asset is stored by the same producer (the Owner). So there is no sale or transfer of control: the owner must also be a user with the "Miller" role. To perform operations on the *Milling Store* it is necessary for the user to have the "Miller" role.

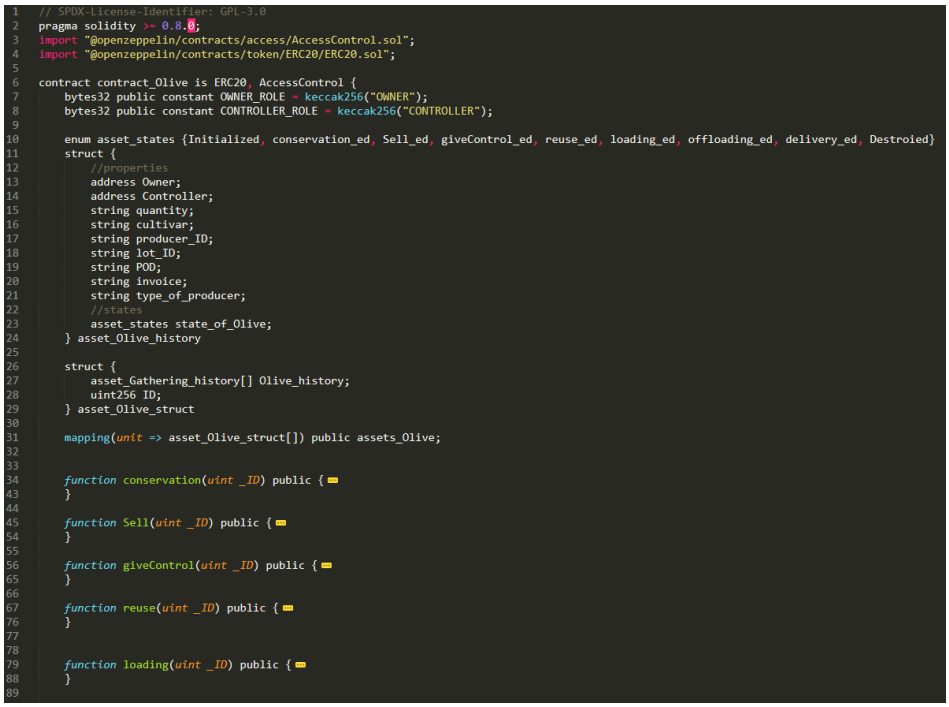
5.2. Translation of the graphical representation

Once the design is complete, the *-chain framework translates the graphical model into smart contract skeletons: the translation tool collects the assets, operations and roles defined in the design interface. Each asset is represented by a different smart contract. Each smart contract contains:

- A data structure representing the history of all the assets of the same type.
- All explicit operations defined on the asset.
- The implicit operations of creation and destruction.
- The implicit operation of "view()" that provides the history of a given asset.

There are four main contracts in the smart contract skeleton: *contract_Olive*, *contract_Storage*, *contract_Truck* and *contract_Milling_Storage*. The main actor is *contract_Olive* which refers

to the asset Olive in the graphic representation. The other contracts are generated from the container elements, used for to store or to transfer the asset. The contract contract_Olive has all the declared property on the asset *Olive*, listed in lexicographical order. On this contract are also auto-generated and listed the functions: “conservation()”, “reuse()”, “loading()”, “offloading()”, “delivery()”, “sell()”, and “giveControl()”. In order to easily understand the structure of the auto-generated smart contract, a snippet code from the programming list of the solidity language is shown in Figure 3.



```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >= 0.8.0;
import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract contract_Olive is ERC20, AccessControl {
    bytes32 public constant OWNER_ROLE = keccak256("OWNER");
    bytes32 public constant CONTROLLER_ROLE = keccak256("CONTROLLER");

    enum asset_states {Initialized, conservation_ed, Sell_ed, giveControl_ed, reuse_ed, loading_ed, offloading_ed, delivery_ed, Destroied}
    struct {
        //properties
        address Owner;
        address Controller;
        string quantity;
        string cultivar;
        string producer_ID;
        string lot_ID;
        string POD;
        string invoice;
        string type_of_producer;
        //states
        asset_states state_of_Olive;
    } asset_Olive_history

    struct {
        asset_Gathering_history[] Olive_history;
        uint256 ID;
    } asset_Olive_struct

    mapping(uint => asset_Olive_struct[]) public assets_Olive;


    function conservation(uint _ID) public {
    }

    function Sell(uint _ID) public {
    }

    function giveControl(uint _ID) public {
    }

    function reuse(uint _ID) public {
    }


    function loading(uint _ID) public {
    }
```

Figure 3: Snippet of Solidity code generated by *-chain framework, translating the graphical representation of Figure 2.

6. Conclusion and Future work

We have demonstrated how the *-chain framework can easily translate very complex supply chains: the DSGL developed for the platform manages to represent complex forms of process, providing easy-to-identify elements. The framework is also able to translate the supply chain schema into a solidity code listing, which is almost optimal to be deployed by a domain expert.

In future work we plan to better analyze the potential of the DSGL, comparing it with several other models and tools, aiming to underline differences or similarities. Also, we plan to develop a validation phase for the generated smart contract, to ensure a more clear and working output code. Furthermore, it would be interesting to analyse the robustness of the code validating the model and the generated code through qualitative analysis. Our task is to refine the framework

and make the graphical interface much easier to handle, especially for inexperienced managers who lack specific knowledge of the model. To further improve the usability of the framework, we plan to introduce the possibility of defining macro-functions, i.e., the composition of existing operations. The goal is to reduce procedural costs and earn an easier and clearer design. As a successive step, we plan to translate it into other languages for DLT, such as Chaincode[6]. Finally, we plan to investigate whether the adoption of different data structures to represent the asset history reduces the storage and execution costs of the proposed solution.

7. Acknowledgments

This work has been partially supported by:

- Gruppo nazionale per il Calcolo Scientifico "GNCSINdAM" - CUP E55F22000270001
- project "RACRA"- funded by Ricerca di Base 2028-2019, Univeristy of Perugia
- Project BLOCKCHAIN4FOODCHAIN: funded by Ricerca di Base 2020, Univeristy of Perugia
- Project DopUP - REGIONE UMBRIA PSR 2014-2020

References

[1] Rita Azzi, Rima Kilany Chamoun, and Maria Sokhn. The power of a blockchain-based supply chain. *Computers & Industrial Engineering*, 135:582–592, 2019.

[2] Stefano Bistarelli, Francesco Faloci, and Paolo Mori. *.chain: automatic coding of smart contracts and user interfaces for supply chains. In *Third International Conference on Blockchain Computing and Applications, BCCA 2021, Tartu, Estonia, November 15-17, 2021*, pages 164–171. IEEE, 2021.

[3] Stefano Bistarelli, Francesco Faloci, and Paolo Mori. Towards a graphical dsl for tracing supply chains on blockchain. In *(To appear in) Euro-Par 2021: Parallel Processing Workshops - Euro-Par 2021 International Workshops, Online Event, September 01-03, 2021*, Lecture Notes in Computer Science. Springer, 2021.

[4] Khaled Salah, Nishara Nizamuddin, Raja Jayaraman, and Mohammad Omar. Blockchain-based soybean traceability in agricultural supply chain. *IEEE Access*, 7:73295–73305, 2019.

[5] Julian Solarte-Rivera, Andrés Vidal-Zemanate, Carlos Cobos, José Alejandro Chamorro-Lopez, and Tomas Velasco. Document management system based on a private blockchain for the support of the judicial embargoes process in colombia. In Raimundas Matulevicius and Remco M. Dijkman, editors, *Advanced Information Systems Engineering Workshops - CAiSE 2018 International Workshops, Tallinn, Estonia, June 11-15, 2018, Proceedings*, volume 316 of *Lecture Notes in Business Information Processing*, pages 126–137. Springer, 2018.

[6] Hartmut Stadtler and Christoph Kilger. *Supply Chain Management and Advanced Planning: Concepts, Models, Software, and Case Studies*. Springer Publishing Company, Incorporated, 4th edition, 2008.

[6] https://hyperledger-fabric.readthedocs.io/en/release-1.3/chaincode.html