EasyChair Preprint

№ 6411

Towards a Graphical DSL for Tracing Supply Chains on Blockchain

Stefano Bistarelli, Francesco Faloci and Paolo Mori

August 26, 2021

Towards a graphical DSL for tracing supply chains on blockchain

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

[1] Dipartimento di Matematica e Informatica - University of Perugia
[2] Dipartimento di Informatica - University of Camerino
[3] Istituto di Informatica e Telematica - National Research Council

Abstract. Nowadays, supply chain tracing notarization is among the most used non-financial blockchain applications. However, creating a blockchain based system for the management of a supply chain remains a complex task. In this paper, we propose a graphical domain specific language (DLS) and a tool allowing the supply chain domain expert to easily represent the supply chain he needs to trace. The graphical representation of the supply chain is then translated in automatic way in a set of solidity smart contracts implementing it. A small intervention of a programmer is required to customize and finalize such smart contracts. The obtained semi-automatic process of smart contract generation will boost the blockchain usage for supply chain traceability.

Keywords: Supply chain management · Blockchain · Smart Contract · graphical DSL

1 Introduction

According to the definition of Stadtler and Kilger [12], a supply chain (SC) is a "network of organizations that are involved, through upstream and downstream linkages, in the different processes and activities that produce value in the form of products and services in the hands of the ultimate consumer". From an analytic point of view, we can define a SC as a flow of goods or services generated by the processes that transform raw objects into intermediate objects, and such objects into final products. Hence, depending on the specific scenarios where they are applied, different types of SC can be defined, e.g., production, distribution, maintenance and sales supply chains. Several studies and applications propose to implement supply chain management systems exploiting the blockchain technology [11,1,9], but according to the same researches, they provide solutions that are not general enough.

This paper proposes a general model aimed at easily representing any specific SC. This model will be then exploited for automatizing SC management systems

design and development over a blockchain. The design phase will be facilitated by a graphical interface enabling the SC manager to represent the objects (assets) involved in the supply chain process as basic components, the operations that can be done on these objects as relations among objects, and their constraints. The development phase will be facilitated because a set of smart contracts skeletons representing the objects, the operations, and the constraints of the supply chain are automatically derived from its graphical representation. Programmers will then finalize and customize these skeletons according to the specific supply chain features.

The paper is organized as follows. Section 2 describes SC features and typologies. In Section 3 we propose our model and framework for blockchain based SC design and development, while Section 4 describes a real use case exploiting our model. Section 5 describes the tool we developed which implements the model we defined. Finally, Section 7 draws the conclusions and describes possible future works.

2 Background

A SC is a system of organizations, people, activities, information and resources involved in the process of transferring or supplying a product or service from the supplier to the customers [12]. In this sense, a SC is a representation of a real system where some "agents" participate to fulfill the service. An agent is any entity involved in the SC including abstract or real subjects like: producers, vendors, warehouses, transportation companies, distributions centers, or retailers [8]. To analyze the definition of SC we study characteristics and properties of SC already classified in literature.

Production supply chains are designed to organize the creation of a product. This type traces the phases in which the asset under analysis undergoes transformations: from the origin point to the end of the life cycle of production. This chain describes in detail the changes, the time required for transformations, the information required for production. These models generally include production of both goods and services [6]. Production supply chain is often represented by a flowchart, where there is always a well-known (defined) initial state, and possibly one or multiple final states.

Distribution supply chains type aims to organize and manage the traceability of resources. A supply chain of this type highlights channels for each macro termination area, and specifies all the agents or the intermediaries involved an asset from the producer to the customer. Distribution channels can include wholesalers, retailers, deliverers, and even the Internet [5][13].

Sales supply chains describe the relationships between distribution nodes of an asset; it does not deal with changes in the asset or possible substantial

changes, but only with the path chain that a product undergoes in its sales or delivery cycle: generally we speak of a finished product from producer to consumers [14]. This topology generally tracks back all the trades of many products in order to analyze their life cycle. This involves analysis such as market overview, production planning and financial strategies [15][2].

Distributed Ledger Technology (DLT) refers to systems and protocols that allow simultaneous access, validation, and updating with immutable data across a network. DLT, more commonly known as Blockchain Technology (BT), given its potential across industries and financial sectors. In simple words, the DLT is all about the idea of a "decentralized" network against the conventional monolithic centralized mechanism. 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 [4], and Monero [5]), 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 [6], widely used for Ethereum [7] systems.

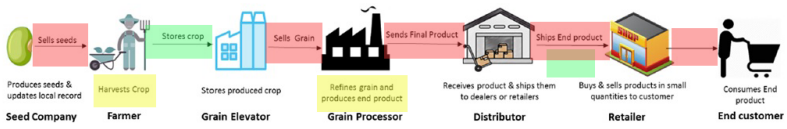

Fig. 1. Scheme of the Supply chain used on soybeans traceability study [9].

Figure 1 shows an example of a real use case of supply chain representing the soybeans life cycle, from the seed production phase, to the end customer sell. This use case has been used in [9] to develop a blockchain based application able to represent supply chains for agricultural products. On the supply chain schema

[4] Bitcoin Project: `https://bitcoin.org`
[5] Monero project: `https://www.getmonero.org`
[6] Solidity white paper: `https://docs.soliditylang.org/en/v0.8.6/`
[7] Ethereum project: `https://ethereum.org/en/`

of Figure 1 we can highlight different phases that describe three different type of supply chain: transitions from a point to another characterize moving operation (highlight in green); the passing from an owner to another represents sales phases (highlight in red); the various phases where the object under examination changes its properties are transformation phases (highlight in yellow).

3 Supply Chain model and graphical representation

Our approach analyzes SC structures in order to highlight their typical elements and to identify recurrent patterns in the interactions among them. As a matter of fact, analysing the existing literature, we found out that there are a number of interaction patterns among the elements building up a SC that are general, i.e., they are not strictly related to the specific business case represented by the supply chain. For instance, a typical pattern is the one which represents the packaging of a number of items in one single traceable package. The identified patterns are exploited to define the basic components of our model. The idea behind our model is to be able to define the workflow of a SC by simply composing the components representing the identified patterns.

We identified the following families of elements involved in a SC: Assets, Containers (packaging), Operations, and Roles. To ease the usage of the proposed model, we define a graphical DSL representing a supply chain model. In this way, users will be able to define the workflow representing their specific SCs by properly combining the graphical components representing the constructs of our model.

Assets They are the objects that the supply chain treats: they typically represent the goods involved in the operations on the supply chain. As a matter of fact, some goods are loaded in the supply chain at the beginning of the process (e.g., raw materials), some operations are applied to such goods to obtain semi-finished products, further operations are applied until the final product is obtained. The assets that, in order to be tracked, must be contained into containers are called *uncountable* (e.g., the milk needs to be stored in a bottle). If an asset involves any kind of destruction as a consequence of its use or transformation, it is called "*consumable*".

Containers Whenever an asset is inserted or accumulated into another object, the latter is referred as "container". Examples of containers are: silos, haulers, ships, and packagings. Containers are used in two cases: *i)* when an asset, for its own nature, must be necessarily contained in a support (for instance, the water must be contained in a bottle), this is the case of uncountable asset; *ii)* when an asset is contained into a package in order to be transported, stored or cataloged (for instance a case of water bottles) Containers are countable and traceable objects. Each asset or container on a supply chain can be "*consumable*" or "*non-consumable*". An egg, a liter of milk, a bag of seeds, bucket

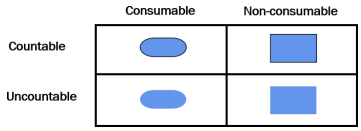

Fig. 2. Assets and containers graphical representation.

of manure, are examples of consumable objects. A tree, a field, a vineyard, are instead examples of non-consumable objects.

Figure 2 shows a diagram of how assets and containers are graphically represented according to the proposed model, based on their properties.

Operations are the components of our model which allow to represent updates, modifications or transformations of an asset. Figure 3 shows some examples of the main operations defined in our model: each operation has specific properties, parameters, and outputs, all described in the following.

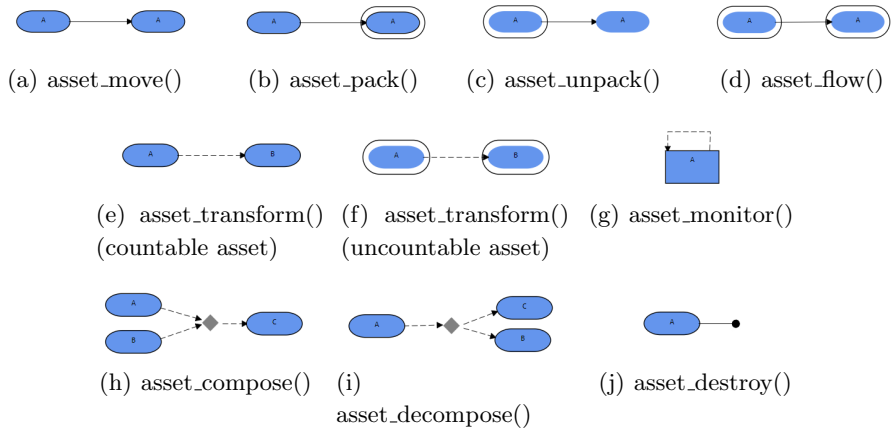

Fig. 3. Examples of possible operations defined on the model.

- **asset_move** (Figure 3(a)) this operation concerns the update of the position - or the geolocalization - of the asset. Figure 3(a) shows the operation applied over a countable asset, but it is also applicable to containers (with uncountable asset inside).
- **asset_pack** (Figure 3(b)) This operation represents the packaging or collection of the asset inside an object suitable for transport or tracking. There is no change of original asset information. The asset is inserted into a further object which in turn can be a source of operations and traceability. This

operation can obviously be repeated several times, with different container objects; it could be also applied to each pair formed by any object and any type of container.

- **asset_unpack** (Figures 3(c)) In this case an asset is extracted from a container. When a non consumable container releases all the assets it contains, it is destroyed. The object contained is removed from the package.
- **asset_flow** (Figures 3(d)) This operation represent the transfer of an asset from one package to another package.
- **asset_monitor** (Figure 3(g)) This operation is used when an object requires a control operation. Therefore this operation does not change the traits of an asset, nor its geolocation: the operation keeps track of information of the asset, relevant for its traceability.
- **asset_transform** (Figures 3(e) 3(f)) The transformation operations are meant to change features of an asset, and they are typically dependent on the specific asset and supply chain. Hence, transformations imply a substantial change of properties and traits of an asset. If the operation is applied to consumable assets, it has the main effect to destroy the original asset and to create the new one (or ones). Sometimes, a non consumable assets may generate assets: in this use, the "transform()" operation has the task of generating the new asset.
- **asset_compose and asset_decompose** (Figure 3(h) 3(i)) Assembly operations exploits existing assets to create a new asset without destroying them. In the opposite way. the operation designed to disassemble objects previously assembled with a composition operation, is defined decomposition. Notice the similarly of the "asset_compose()" with the "asset_transform()": in both cases a new asset is created. However, the transform operation destroys the previous asset, while in the compose operation the original assets are still there.
- **asset_destroy** (Figure 3(j)) when an object has to be destroyed and is no longer part of the supply chain, it is destroyed.

4 Representing a real case

This section shows how the proposed model can be exploited to represent the supply chain defined in [9] and represented in Figure 1.

Here we suppose that the Soy Bean Producer, SBP, wants to track the soybean production process, from the acquisition of the seeds to their commercialization. For this reason, SBP exploits our framework to represent the main steps of the production process, thus automatically obtaining the skeletons of the smart contracts that represent each asset of the supply chain. Figures 4 and 5 show the graphical representation of the soybean production process using our framework.

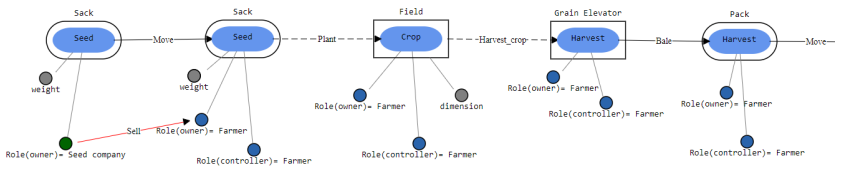

Fig. 4. Representation of the soybeans supply chain use case [9], seen in Figure 1, with the proposed framework (part I).

The first asset of the supply chain represented in Figure 4 is *Seed* (the leftmost box in the figure), which is an uncountable asset and hence it is enclosed in a container, called *Sack*. This asset does not have any incoming arrow. This means that the production of this asset is not tracked using our framework, and this asset is simply created by a subject. Our framework allows to set constraints on the role of the subject who can create/buy an asset. In the reference example, the subjects allowed to create *Seed* assets must have the role *Seed Company*. As a matter of fact, in the figure we can see the constraint Role(owner)=*Seed Company* paired with the *Seed* assets.

The operation that is done on the *Seed* asset is *Sell*, which is paired with the operation *Move*. Hence, in the graphical representation of the supply chain we have a second instance of the *Seed* asset on the right of the first instance, and these two instances are directly connected through a *Move* arrow, which represent the physical transfer of the asset and, at the same time, the owner properties of the two instances are connected with a red arrow representing the *Sell* operation. The framework, by default, imposes the constraint that only the owner of an asset can perform the *Sell* operation. This constraint is not explicitly reported in the supply chain graphical representation. In the reference example, a further constraint is defined on the role of the entity which can buy *Seed*. This constraint is represented in the Figure 4 as Role(owner) = *Farmer*, and it is paired with the second instance of the *Seed* asset.

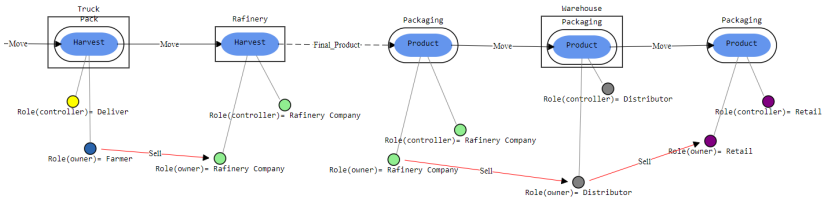

Fig. 5. Representation of the soybeans supply chain use case [9], seen in Figure 1, with the proposed framework (part II).

The second operation in the soybeans supply chain represented in Figure 4 is the *Plant* one. This operation is a transformation (as shown by the dashed line) because the *Seed* asset is transformed in *Crop* asset when it is planted in the field. The *Crop* asset is uncountable as well, and hence it is included in a *Field* container, which represents the place where the seeds have been planted. The *Field* container is, obviously, *Non-consumable* and hence it is represented by a rectangle in the figure. More than one *Field* container can be defined in the supply chain, and the ID of the one to be used is specified in the invocation of the *Plant* operation. Our framework, by default, imposes the constraint that only the controller of an asset can perform an operation on such asset (with the exception of the *Sell* one which requires the invoking entity to be the owner, as previously explained). The reason is that the controller is the entity who have the physical availability of the asset. This constraint is automatically embedded in the smart contract representing the asset. Moreover, since the *Plant* operation can be executed only by entities having the role of *Farmer*, this additional constraint is explicitly paired with the *Plant* operation, and in Figure 4 is represented by the constraint Role(controller)=*Farmer* paired with the *Crop* asset. However, differently from the *Sell* operation, in this case the constraint is imposed on the Controller of the asset, i.e., on the entity who has the physical availability of the asset. Another constraint that is imposed on this operation is that a given ratio between the weight of the seeds and the dimension of the field must be respected. Hence, a constraint taking into account the weight property of the *Seed* asset and the dimension property of the *Field* container is defined by the *SBP* on the *Plant* operation. More than one seed sack could be planted in the same field, generating multiple *Crop* assets included in the same *Field* container. Hence, the constraint will take into account the total weight of all the seed sacks already planted in the field to decide whether the *Plant* operation can be executed. Our framework, when producing the smart contract representing the assets, defines the methods representing the operations and the constraints, and the related invocations. The programmers will then customize such methods by writing the code implementing the required constraint checks.

The third operation of the supply chain is *Harvest_crop*, and its features are very similar to the *Plant* operation. The result of the *Harvest_crop* operation applied to each *Crop* asset is a new asset, called *Harvest*. The *Harvest* assets are stored in a *Grain Elevator Non-consumable* container. More than one *Grain Elevator* containers can be defined in the supply chain, and the ID of the one to be used is specified in the invocation of the *Harvest_crop* operation.

The next operations are very similar the ones we have already described, hence we will not provide a detailed description.

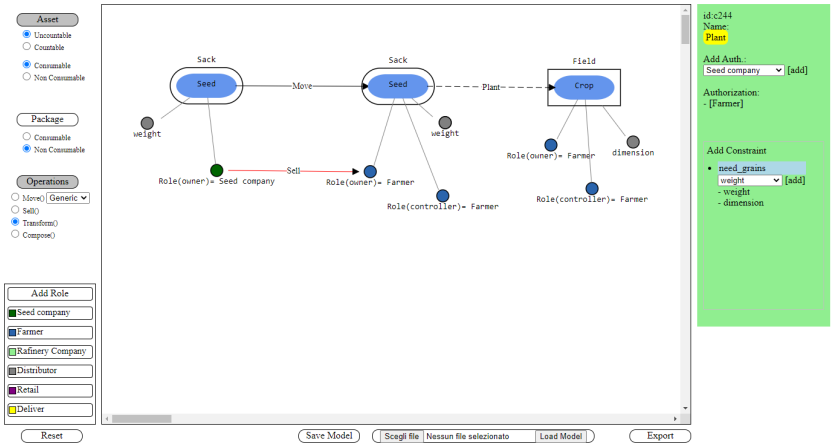

Fig. 6. A screenshot example of the proposed tool.

5 Describing the tool

The developed tool point to describe the SC, and translate it into smart contracts. The user has only to draw the equivalent model of the supply chain : solid blocks that represent assets, linked particles for the properties of assets; arrows to represents the operations; roles and constraints that enrich each function specified with arrows. The tool translates the components into code suitable for creating smart contracts for a related framework. The smart contract structure is procedurally generated in solidity-like language, starting from blocks, arrows, constraints, and roles.

First set of available functions to the user is the representation of the assets: the three different classifications of an asset -described in the proposed model- are available through the “Asset” button. On this button selection, is possible to choose among the characteristic: uncountable or countable, consumable or non-consumable. Another basic feature is the “Package” draw option. In this case, there are only two options: consumable and non-consumable. Because a package is considered always a countable asset. Once a package object has been placed, any asset object could be dragged into it (or any other package object with its relative content).

Another basic set of functions is operations. In this subset it is possible to draw each operation of the model according to each family: move, transform, compose. Also, it is possible to select the “sell” operation, which is enabled only between two “owner properties” of the same asset in two different instances. To characterize the various operations of a SC it is possible to assign roles with and

impose constraints on access to certain functions at a given time. Through the "roles" panel it is possible to build the set of roles necessary for the specific SC.

Role constraints are only one specific constraint that can be defined. Interface prompt to set constraint and any defined property of the assets. Any object drawn in the main panel can be edited. By clicking on one of them, the green editing area will appear which the names and properties of each object can be changed: as already mentioned, selecting an operations it is possible in this area to add constraints an authorizations.

Otherwise, in cases of the assets, packages, or properties, the editable options are limited to appearance, nomenclature, and handling properties.

To make the tool more user-friendly and to make the model easily editable over time, two functions allow saving and loading the drawn schema. The function "Save model" saves the drawing and the properties of the SC thus constructed, translated in JSON format. The function "Load model" loads a JSON file on the tool, then automatically redraws the schema.

The function "Export" translates the model into smart contract prototypes, which is based on the recently released *solidity* standard. The translator parses the saves file JSON as a starting point. Based on assets, operations, roles, and constraints, it generates a Solidity code.

6 Relevant Related Work

One of the most similar study[7] presents a solution by a tool capable to design solidity code based on predetermined logic blocks: since models are usually easier to understand than software source code. This solution is based on a virtual environment that allows to build a smart contract, giving easily understandable bricks. Unlike this approach, our solution does not include a few pre-set bricks or a few common combinations of solidity code. Our graphical DSL aims to be as general as possible such that the framework can represent multiple combinations and types of supply chains.

In[10], a model based on the Business Process Model and Notation (BPMN) representation is shown. The developed graphical DLS translates blockchain smart contracts using the graphical representation of the DEMO modelling language[4]. This representation makes it easier for the user to represent a workflow or the transaction operations of the same asset. Due to the nature of the BPMN representation, through this is not possible to design various types of supply chains.

This study[3] presents an automatic smart contract template generation framework that uses ontologies and semantic rules to encode specific transaction problems. The template uses the structure of abstract syntax trees to organise the constraints in the generated template in a solidity script. The minimum atom

of the constraint is the declaration of the owner and his/her ownership of an object. Similarly to our model, the study describes how the roles and minimum relations (atoms) can build more complex operations.

7 Conclusion and Future work

Given the various natures of SCs, it is difficult to design a SCMS general enough to be able to represent all possible types of SCs. We present a universal model for SCs ; this model represents every aspect of the most used SC such as production processes or in business management. The model constitutes a graphical DSL for the representation of SCs. Through this model, we analyze and reconstruct a well-known use case: soybean traceability schema. its SC is translated through the presented model, also adding more detail to the original schema. Furthermore, we developed an easy-to-use graphic framework: the proposed tool allows a manager to design the various components of a CS and to specify their relationships and constraints.

As future work we plan to better analyze the proposed model, comparing it with several other general schemes, aiming to underline the differences in use or similarities. Our task is to refine the tool and make the graphical interface easier to handle, especially for inexperienced managers who lack specific knowledge of the model.

Also, we plan to analyze a specific use case such as "DOPUP: Dop Olive oil for a new Presence of Umbria on the Planet", about Umbria's olive supply chain.

As a successive step we plan to translate into other code languages for DLT, such as Chaincode [8].

References

1. Azzi, R., Chamoun, R.K., Sokhn, M.: The power of a blockchain-based supply chain. Computers & Industrial Engineering **135**, 582–592 (2019). https://doi.org/10.1016/j.cie.2019.06.042
2. Brennan, L., Rakhmatullin, R.: Global Value Chains and Smart Specialisation Strategy: Thematic Work on the Understanding of Global Value Chains and their Analysis within the Context of Smart Specialisation. JRC Working Papers JRC98014, Joint Research Centre (Seville site) (Dec 2015). https://doi.org/https://doi.org/10.2791/44840, `https://ideas.repec.org/p/ipt/iptwpa/jrc98014.html`
3. Choudhury, O., Rudolph, N., Sylla, I., Fairoza, N., Das, A.: "auto-generation of smart contracts from domain-specific ontologies and semantic rules" (09 2018). https://doi.org/10.1109/Cybermatics_2018.2018.00183

[8] White paper of Chaincode: `https://hyperledger-fabric.readthedocs.io/en/release-1.3/chaincode.html`

4. Dietz, J.L.: Demo: Towards a discipline of organisation engineering. European Journal of Operational Research **128**(2), 351–363 (2001). https://doi.org/https://doi.org/10.1016/S0377-2217(00)00077-1, https://www.sciencedirect.com/science/article/pii/S0377221700000771, complex Societal Problems
5. Govindan, K., Soleimani, H., Kannan, D.: Reverse logistics and closed loop supply chain: A comprehensive review to explore the future. Eur. J. Oper. Res. **240**(3), 603–626 (2015). https://doi.org/10.1016/j.ejor.2014.07.012
6. Malik, S., Kanhere, S.S., Jurdak, R.: Productchain: Scalable blockchain framework to support provenance in supply chains. In: 17th IEEE International Symposium on Network Computing and Applications, NCA 2018, Cambridge, MA, USA, November 1-3, 2018. pp. 1–10. IEEE (2018). https://doi.org/10.1109/NCA.2018.8548322
7. Mao, D., Wang, F., Wang, Y., Hao, Z.: Visual and user-defined smart contract designing system based on automatic coding. IEEE Access **7**, 73131–73143 (2019). https://doi.org/10.1109/ACCESS.2019.2920776
8. Ph.D., Rutner, and Shepherd, C.: Is Marketing Still Part of Supply Chain Management, and Should Marketing Academics and Practitioners Care? (09 2017)
9. Salah, K., Nizamuddin, N., Jayaraman, R., Omar, M.: Blockchain-based soybean traceability in agricultural supply chain. IEEE Access **7**, 73295–73305 (2019). https://doi.org/10.1109/ACCESS.2019.2918000
10. Skotnica, M., Klicpera, J., Pergl, R.: "towards model-driven smart contract systems – code generation and improving expressivity of smart contract modeling" (2020)
11. Solarte-Rivera, J., Vidal-Zemanate, A., Cobos, C., Chamorro-Lopez, J.A., Velasco, T.: Document management system based on a private blockchain for the support of the judicial embargoes process in colombia. In: Matulevicius, R., Dijkman, R.M. (eds.) Advanced Information Systems Engineering Workshops - CAiSE 2018 International Workshops, Tallinn, Estonia, June 11-15, 2018, Proceedings. Lecture Notes in Business Information Processing, vol. 316, pp. 126–137. Springer (2018). https://doi.org/10.1007/978-3-319-92898-2_10
12. Stadtler, H., Kilger, C.: Supply Chain Management and Advanced Planning: Concepts, Models, Software, and Case Studies. Springer Publishing Company, Incorporated, 4th edn. (2008). https://doi.org/https://doi.org/10.1007/978-3-540-74512-9
13. Tetteh, A.: Supply chain distribution networks: single-, dual- and omni-channel (05 2014)
14. Trautmann, N., Fündeling, C.: Supply chain management and advanced planning in the process industries. In: Waldmann, K., Stocker, U.M. (eds.) Operations Research, Proceedings 2006, Selected Papers of the Annual International Conference of the German Operations ResearchSociety (GOR), Jointly Organized with the Austrian Society of Operations Research (ÖGOR) and the Swiss Society of Operations Research (SVOR), Karlsruhe, Germany, September 6-8, 2006. pp. 503–508 (2006). https://doi.org/10.1007/978-3-540-69995-8_80
15. Wang, T., Lan, Q., Chu, Y.: Supply chain financing model: Based on china's agricultural products supply chain. Applied Mechanics and Materials **380-384**, 4417 – 4421 (2013). https://doi.org/https://doi.org/10.4028/www.scientific.net/AMM.380-384.4417