Enterprise Modelling and Information Systems Architectures
Vol. 15, No. 15 (2020). DOI:10.18417/emisa.15.15

Special Issue on Blockchain Technologies by Hans-Georg Fill, Peter Fettke and Stefanie Rinderle-Ma

Decentralized Business Process Control using Blockchain

An experience report from two applications: Food Supply Chain and Car Registration

Sérgio Guerreiro[*,a,b], Diogo Silva[a,b], Tiago Rosado[a,b], André Vasconcelos[a,b], Miguel Correia[a,b], Pedro Sousa[a,b]

[a] INESC-ID, Rua Alves Redol 9, 1000-029 Lisbon, Portugal
[b] Instituto Superior Técnico, Universidade de Lisboa, Av. Rovisco Pais 1, 1049-001 Lisbon, Portugal

Abstract. *Many applications supported by blockchain technology are emerging in the industry. Blockchain can be used to enforce the correctness of business process execution and, consequently, increasing the trust among the stakeholders involved in complex business processes. Since few experiences in the area have been reported, more are needed to better understand the benefits and pitfalls of blockchain-based solutions for decentralized business process control. This paper is an experience report of two projects that enforce decentralized business process control using blockchain – a food supply chain and a car registration – using a Design Science Research Methodology approach. Each application follows a different business process design – DEMO and BPMN – leading to different implementations based on Hyperledger Fabric. The major lessons learned are related to the appropriateness of DEMO and BPMN as business process modelling languages in the context of blockchain applications.*

Keywords. Blockchain • Business Processes • Control • BPMN • DEMO • Hyperledger Fabric

Communicated by Hans-Georg Fill. Received 2019-06-25. Accepted after 1 revision on 2020-10-05.

1 Introduction

A *business process* (BP) is a collection of events, activities, and decisions that brings value to the customers of an organization (Dumas et al. 2017). In industrial environments, this notion has to be decoupled in: the *BP model*, referring to the desired behavior of a BP; and the *BP instance*, referring to the actual behavior in operation in a specific organization. Due to the existence of workarounds (Alter 2015), a BP model *per se* does not guarantee that actors perform their responsibilities in BP instances without misalignment. Therefore, BP instances have to be monitored and validated against the corresponding BP models; when a misalignment is identified, change actions have to be performed (Rozinat and Aalst 2008; Van Der Aalst 2011).

We define *business process control* (BP control) as the ability to steer, with bounded effort, the operation of BP instances towards the desired BP model whenever changes or perturbations occur (Guerreiro and Tribolet 2013). There have already been experiments with mechanisms to enforce BP control using an explicit design in the BP models, e. g., by persisting the relevant facts in data stores (Guerreiro et al. 2012).

We define *blockchain* as a decentralized ledger that records transactions and allows tracking assets securely and reliably (Nakamoto 2008; Peck 2017; Underwood 2016). A blockchain is *decentralized*

[*] Corresponding author.
E-mail. sergio.guerreiro@tecnico.ulisboa.pt
This work was supported by national funds through Fundação para a Ciência e Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID) and by the European Commission program H2020 under the grant agreement 822404 (project QualiChain).

in the sense than it is composed of a set of nodes managed by different entities, so it is not controlled by any entity individually. A blockchain is an append-only log, whose records are grouped as timestamped blocks. Each block contains a list of transactions, and contains a cryptographic hash that references the block that comes before it (Christidis and Devetsikiotis 2016). Each block is immutable, *i.e*, it cannot be modified. The sequence of blocks is replicated in all nodes. These properties are useful in a complex supply chain where actors do not necessarily trust each other or to register data with high-integrity requirements like car registrations.

Blockchain seems to be a promising approach towards enforcing BP instance *decentralized control*, i. e., control of what each actor can do, accounting and actuating autonomously. There are two motivations for this: *(i)* the increasing importance of controlling BP instances in areas such as food supply, government, finance, intellectual property, and real state and *(ii)* the increasing mistrust towards centralized institutions. Therefore, the research questions addressed by this paper are: *Is blockchain technology able to support decentralized business process control? How appropriate are the DEMO and BPMN languages to model business processes supported by blockchains?*

We aim to answer these questions using the Design Science Research Methodology (DSRM) (Peffers et al. 2007). Winter proposes a two dimensions' framework: research outputs and research activities (Winter 2008, p. 255). Research outputs are composed by constructs, models, methods or instantiations. Research activities are composed by processes such as build and evaluate.

In this paper two artefacts, the output of the research, are presented: a *food supply chain* and a *car registration service*. Each artefact represents a different approach towards blockchain applications: the first resorts to an ontological design using Design & Engineering Methodology for Organizations (DEMO) and is mainly focused on the conceptual integration between BP and blockchain assets. The second uses Business Process Model and Notation (BPMN) models that are implemented directly without a conceptual concern for reuse. The two industrial applications are different but have a common need to control the asset's life cycle. Regarding DSRM, artefacts are useful if they show evidence of an increase in the body of knowledge. In our paper, the gain is measured by the new knowledge regarding how to use blockchain in the context of industrial BPs. Truth is guaranteed by the *(i)* qualitative/quantitative outcome analysis, *(ii)* separation of the artifacts' domains from the research team and *(iii)* inclusion of artifacts from professional landscape to increase practicability.

The paper is organized as follows. Considering the DSRM, after the motivation and problem identification, presented in this section, Sect. 2 introduces additional blockchain and BP concepts. Sect. 3 and 4 present the two artifacts developed in the context of the solution definition (Sect. 3.1 and 4.1), and the design, development, demonstration (Sect. 3.2 and 4.2), and evaluation (Sect. 3.3 and 4.3) activities of the DSRM. Sect. 5 presents an additional evaluation by comparing the results obtained with the two artifacts in order to extract lessons learned. Sect. 6 compares our work with others. Finally, Sect. 7 concludes the paper (communication activity of the DSRM).

2 Background

The background for this research is provided next. Firstly, the Blockchain and Hyperledger Fabric concepts are presented. Then, BPMN and DEMO are also introduced.

2.1 Blockchain

The term blockchain was originally introduced in the context of the Bitcoin cryptocurrency (Nakamoto 2008). The blockchain is the data structure used to register transactions of Bitcoins. However, the concept evolved and today the term blockchain is used in manifolds senses. First, a blockchain, or more generically a distributed ledger, is a replicated data structure where some sort of operations and other data are stored (Wood 2014). Moreover, the term blockchain is also used to designate a distributed system composed of a set of nodes (computers) that store copies of the above-mentioned data structure. Blockchains are permissionless if they allow any node to join the system, or permissioned if only authorized nodes can join (e. g., nodes of a certain consortium of organizations) (Peck 2017). Many permissionless blockchains implement a specific service, typically a cryptocurrency, like Bitcoin, Litecoin, Monero, and more than 5,700 others at the time of this writing.[1] Others, both permissionless and permissioned, support the implementation of smart contracts and distributed applications, e. g., Ethereum (permissionless), Hyperledger Fabric, Quorum (CONSENSYS 2020), Hyperledger Burrow (Hyperledger 2020), and Corda (Corda 2020) (all permissioned). The selection of a permissionless or permissioned blockchain depends on the application. The openness provided by permissionless blockchains is desirable for many publicly-available applications, but certainly undesirable for others. Blockchains are append-only, meaning that they provide a ledger in which blocks are inserted at the end of the list and never removed. For the blockchain system to agree – to find consensus – on the next block to append, there is a need to decide which block it should be. Bitcoin uses a proof-of-work (PoW) mechanism to decide this matter (Nakamoto 2008). The idea is that the first node (miner) solves the cryptopuzzle, obtains a valid PoW, and then disseminates its block with the PoW. In response, the other nodes accept to append the block to the chain. Due to the existing competition between miners, two blocks with valid PoWs may be disseminated concurrently. When that happens, nodes append both to their copy of the blockchain creating a branch, but eventually prune the shortest branch.

Hyperledger Fabric is a blockchain implementation sponsored by the Linux Foundation and IBM (Androulaki et al. 2018). Hyperledger Fabric, as a permissioned blockchain, restricts the participation in the system to nodes that are identifiable and trusted. Restricting the nodes of the system, enables performance improvements and power consumption reduction in comparison to permissionless blockchains. As the nodes are known to every element in the system, Byzantine fault-tolerant (BFT) consensus algorithms (Correia 2019) can replace the typical power hungry PoW. Permissioned blockchains can usually be implemented within a group of entities who may not completely trust each other. Fabric allows the of plugging different consensus algorithms. The current version provides only crash fault-tolerant consensus based on Apache Kafka. However, BFT-SMaRT, a BFT consensus/replication library has been integrated with Fabric (Sousa et al. 2018). In Fabric the processing of a transaction has three phases, in an *execute-order-validate* pattern. In the execution phase, each transaction is executed and its correctness verified by a restricted set of nodes called endorsement peers. During this phase it is possible for transactions to be executed in parallel. The second phase is performed by an ordering service, responsible for establishing the order of the transactions received and already signed by the endorsement peers, using the consensus mechanism defined and fabricating blocks accordingly. The ordering service nodes are also responsible for updating the state of the blockchain to all peers using atomic broadcast. In the validate phase, transactions are validated by the remaining peers of the network, checking against the trust assumptions considered for each specific application and endorsement policies are verified. If there is no issue in this last step, the block is then appended to the blockchain on each node local copy.

The prototypes of our two applications are based on Hyperledger Fabric for three reasons. First, we needed a permissioned blockchain, as in both cases only authorized actors can be allowed to write to the blockchain. Second, Fabric is the most adopted permissioned blockchain, with a market share of around 50% (Hyperledger 2019). Third, it allows developing applications using Hyperledger Composer (Linux Foundation 2018). Composer provides a domain specific language that greatly simplifies the development of code for Fabric.

[1] https://coinmarketcap.com/

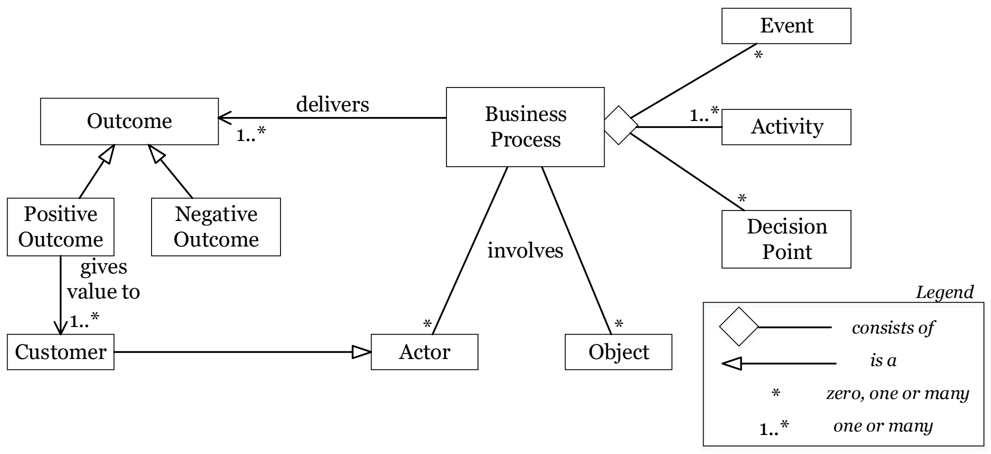


Figure 1: BPMN metamodel as proposed by Dumas et al. 2017.

2.2 Business Process

Business Process Model and Notation

A BP is commonly perceived as the sequencing of activities and resources that organizations use to produce value to their clients. The graphical representation of activities, in particular their sequence, is an effective way of modeling processes. The BPMN specification (Dumas et al. 2017; OMG 2011) is one of the most common graphic process modeling notations used for this purpose. Initially proposed by the Business Process Management Initiative, it is now under the scope of Object Management Group,[2] and the ISO 19510 standard.[3] BPMN is focused on modeling the articulation activities, resources, flows, gateways, events, messages and data objects that occur in a business process.

A high-level BPMN meta model proposed by (Dumas et al. 2017) is depicted in Fig. 1. It supports a detailed specialization of activities, events, and gateways leading to over 100 graphic symbols. Despite such a high amount of symbols, BPMN can lead to disparate model interpretations concerning the two core concepts of any BP: resource and activity. In this context, Fickinger and Recker 2013 concluded that BPMN has a level of 51.3% of overlapping language concepts and lacks state concept to ensure more sound semantics.

[2] https://www.omg.org/
[3] https://www.iso.org/standard/62652.html

Design & Engineering Methodology for Organizations

Dietz and Mulder 2020 introduce the notion of Enterprise Ontology (EO) as a set of capabilities to deal with the essential aspects of process-based organizations. DEMO describes the organization using the essential models to represent organizational design and operation. Décosse et al. 2014 show a broad usage of DEMO by the industry, reinforcing the idea that EO can capture the essence of the organization while offering abstraction from implementation details.

Among other aspects, DEMO prescribes a pattern for the communication and production of acts and facts that occur between actors in the scope of a business transaction. This pattern is of key importance in this paper.

Fig. 3 depicts the DEMO constructs for representing business transactions. Starting in the top left part of the figure, an elementary actor is represented by a white box, while a composite actor (a network of transactions and other actors inside) is represented by a grey box. A business transaction type is represented by a circle with a diamond inside, and inherits the previously introduced DEMO standard pattern of a transaction definition. Next, in the second row of the figure, the boundary of an organization is represented by a grey line where all the business transaction's types and actor roles are designed inside it. The actor

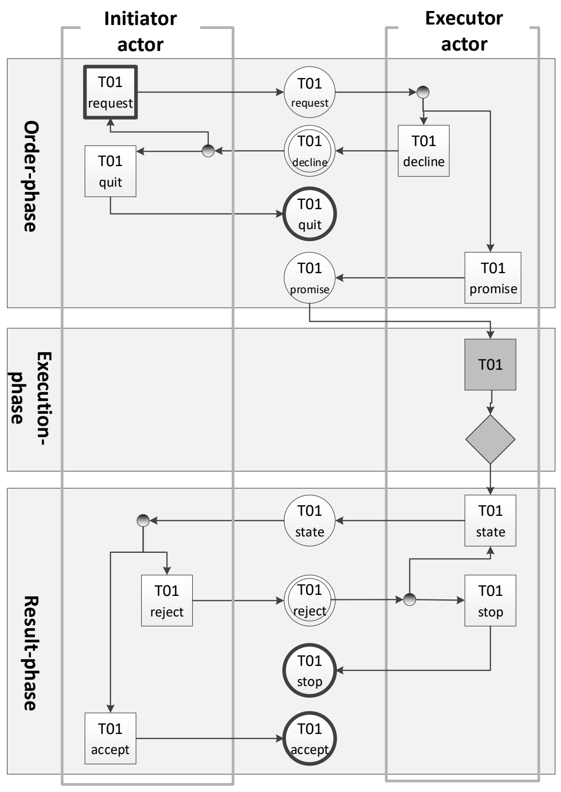


Figure 2: DEMO standard pattern of a transaction between two actors with separation between communication and production acts (Dietz and Mulder 2020).

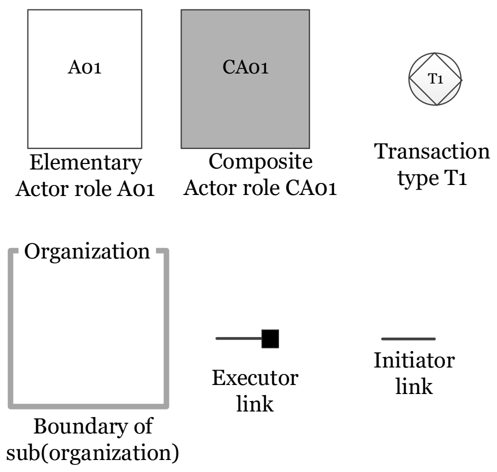

A01
Elementary Actor role A01
CA01
Composite Actor role CA01
T1
Transaction type T1
Organization
Boundary of sub(organization)
Executor link
Initiator link

Figure 3: Constructs for representing DEMO business transactions (Dietz and Mulder 2020).

role executor is represented with an executor link connected to the transaction type. Similarly, the actor role initiator is represented with the initiator link.

Similarly with other approaches (Bork and Sinz 2013; Ferstl and Sinz 1998; Sinz 2018), a DEMO business transaction model has two distinct worlds: *(i)* transition space and *(ii)* state space. DEMO's transition space is grounded in a theory named Ψ-theory (PSI), where the standard pattern of a transaction includes two distinct actor roles: the *Initiator* and the *Executor*. The transaction pattern is performed by a sequence of coordination and production acts that leads to the production of the new production fact. In detail, it encompasses: *(i)* order phase that involves the acts of request, promise, decline and quit; *(ii)* execution phase that includes the production act of the new fact itself; and *(iii)* result phase that includes the acts of declare, reject, stop and accept. Firstly, when a Customer desires a new product, he requests it. After the request for the production, a promise to produce is delivered by the Producer. Then, after the production, the Producer declares that the production is available. Finally, the Customer accepts the new fact produced. DEMO's basic transaction pattern aims at specifying the transition space of a system that is given by a set of allowable sequences of transitions (see Fig. 2).

We use an example to better explain the conceptualization beneath DEMO Ψ-theory. This example uses the DEMO principles but is expressed using the BPMN notation to take benefit of a well-known notation (BPMN) and the comprehensive business process semantics definition (DEMO). Considering a business transaction of *producing a product*; at least two business actors need to be considered: the transaction initiator (TI) and the transaction executor (TE). Considering that both actors are located in two different organizations, then two BPMN pools are considered here. The first initiates the transaction and the second executes it. Moreover, the second only starts production in reaction to an explicit request from the first one. When TI decides to request a product, the task of *Request product* is executed resulting

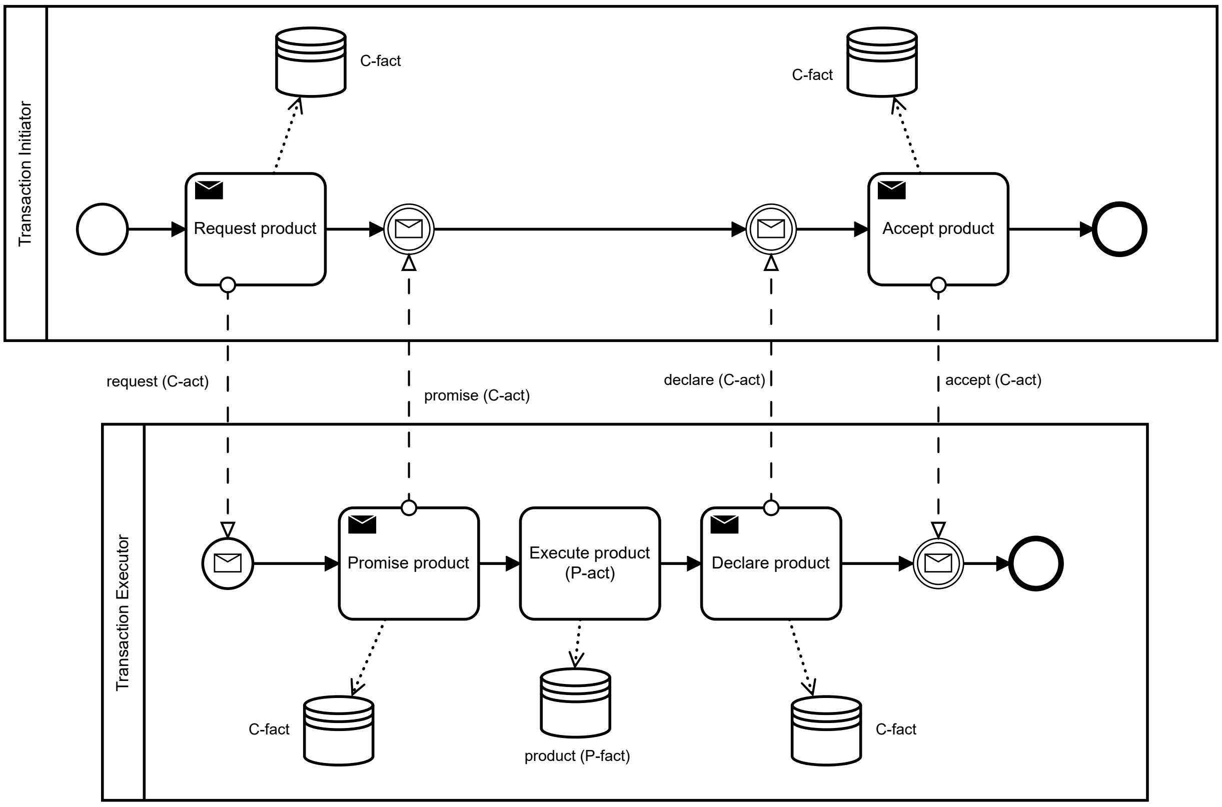


Figure 4: The DEMO standard pattern of a transaction between two actors with separation between communication and production acts represented in BPMN.

in *(i)* the emission of a communication act (C-act) with the purpose of *Requesting a product* to the TE, and *(ii)* the creation of a new communication fact (C-fact) in the world. This double outcome is repeated in all the business transaction steps, except on the *Execute product* step where no C-act is expected but a P-act is. Fig. 4 expands all the business transaction steps included in a DEMO standard pattern, clarifying how the sequence of states originate a new P-fact in the world. All the business transactions respect this pattern, even when some acts or facts are not observable. In that situation, the acts or facts are considered implicit in the execution of the business transaction.

A more detailed pattern is presented in Fig. 5. This time, the disagreements between actors are considered, namely declining to produce a product and the rejection of a produced product. If a business actor disagrees with another, then a decision point is achieved. In case of declination, it is up to the TI to issue a new request; on the contrary, in case of rejection, the TE needs to evaluate the rejection arguments before deciding between stopping or re-declaring the product. Both situations can lead to a deadlock situation; it is the context of social norms that prevents a situation of eternal blockage. Besides disagreements, the comprehensive DEMO pattern of a transaction also includes revocations. Due to the scope of our paper, this aspect is not further discussed.

In real environments, the business transactions are defined in a network of actors and transactions, e. g., a *Payment* transaction succeeds a *Production* transaction. Fig. 6 exemplifies a possible dependency between transactions: the request after promise pattern. After Tx1 is promised, then Tx2 request and Tx3 request are triggered. Conversely, Tx1 is only declared after the declaration of Tx2 and Tx3. In this example, it is noted that an actor can be assigned with multiple roles, e. g., Actor Role B is Tx1 TE, Tx2 TI and Tx3 TI. As a conclusion, DEMO Ψ-theory specifies semantic meaning to the business transaction. The business transaction dynamic is detailed encompassing the actors, the communications, the productions and all its dependencies. In contrast, BPMN does not prescribe any semantic for the business process model; it only provides a set of constructs that could be combined accordingly with a specification.

3 Experience 1: Food Supply Chain

3.1 Universe of Discourse

We consider the case of a food supply chain in the UK (Wilson 1996). There are 3 major companies in the case: *(i)* J. Sainsbury, a UK retailer; *(ii)* Mack Multiples, an operation division of M & W Mack that is a distributor of J. Sainsbury; and *(iii)*, a major family-owned plantation business in Ecuador. For readability, from this point forward the concept of DEMO business transaction will be denominated BP. This food supply chain involves the BP of production, testing, transporting, delivering and payment between the participants involved. The journey from Ecuador to J. Sainsbury in the UK takes 13 days. The plantation of bananas starts in Ecuador, in Noboa, considering the J. Sainsbury customer preferences. When the plantation is over, bananas are enclosed in plastic wraps. Their stem is harvested, they are cut and are transported to the packhouse. Then, they are inspected by quality managers, floated through a fungicide bath to prevent infections and packaged to be transported on a ship to Zeebrugge, Belgium. At Zeebrugge, the bananas are bought and inspected by Mack personnel. The merchandise is then delivered to Mack at Kent, where bananas are stored in ripening rooms for five to six days in order to achieve the stage of ripeness that J. Sainsbury has specified. After that, the fruits are transported to J. Sainsbury depots in a Mack temperature-controlled truck, where its staff test and pay for the bananas if defined specifications are met. The fruits are finally delivered to J. Sainsbury's stores where the customers can purchase them.

3.2 Implementation

Food supply chain modelling in DEMO starts with the construction of a Transaction Result Table (TRT) that defines the existent BPs and

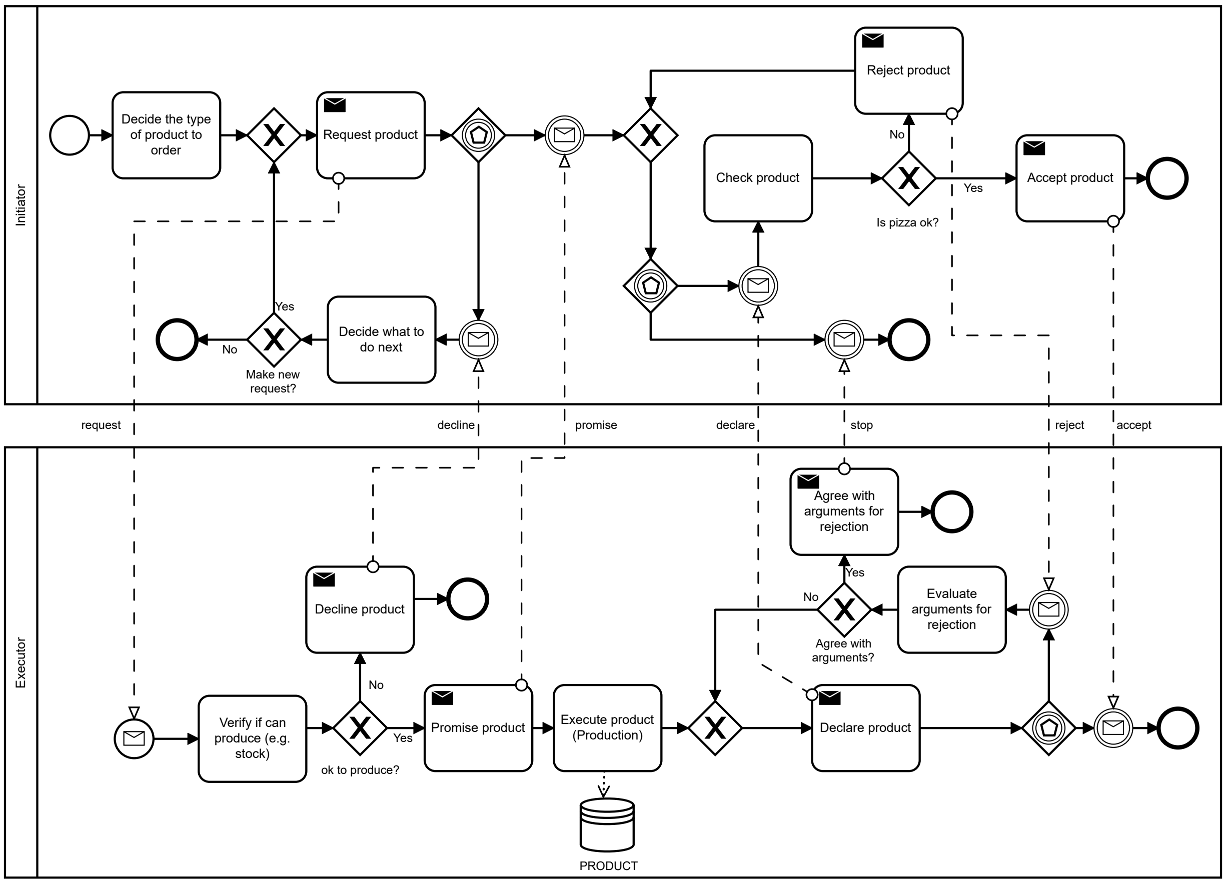


Figure 5: The DEMO standard pattern of a transaction between two actors with separation between communication and production acts, and presenting the disagreements, represented in BPMN.

Table 1: Food supply chain: DEMO Transaction Result Table

Transaction kind	*Product kind*
T01 – Best Producers Report completing	P01 – Best Producers Report BPR is completed
T02 – Taste Test completing	P02 – Taste Test TB is completed
T03 – Consumer profiling completing	P03 – Consumer Profile CP is completed
T04 – Order creating	P04 – Order O is created
T05 – Order producing	P05 – O is produced
T06 – Order transforming	P06 – O is transformed
T07 – Order transporting	P07 – O is transported
T08 – Order unloading	P08 – O is unloaded
T09 – Order testing	P09 – O is tested
T10 – Order treating	P10 – O is treated
T11 – Order packing	P11 – O is packaged
T12 – Order paying	P12 – O is paid
T13 – Order invoicing	P13 – O is invoiced
T14 – Order storing and treating	P14 – O is stored and treated

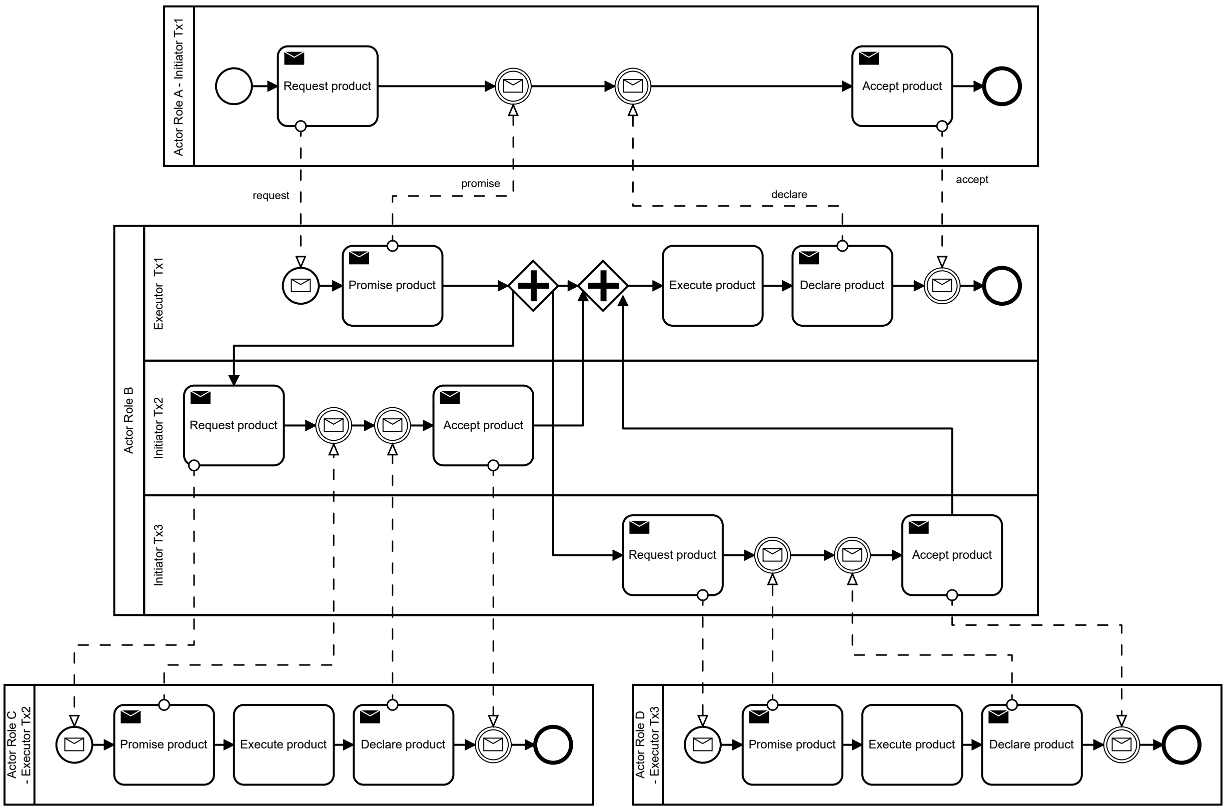


Figure 6: Dependency between transactions: request after promise pattern.

their expected outcome production (see Tab. 1). Afterwards, an Actor Transaction Diagram (ATD) is constructed, to include the relations between actor roles (initiator and executor), the BPs expressed in TRT and the boundary of the existent organizations defining which BPs are inside each organization. The overall food supply chain ATD is in Fig. 7. The boundaries correspond to the three organizations involved plus an intermediary place, Zeebrugge, where the bananas are transferred to. In Tab. 1, the transaction kinds from T04 to T14 refer to the different phases of the orders of bananas made by diverse actors from the three organizations. The others refer to the creation of other artefacts, for example the execution of T01 delivers a *best producers* report as outcome.

Using ontological transactions, the orders of bananas are modelled (order issued by J. Sainsbury, order issued by Mack, *etc.*) as the same order in the ATD and TRT. Despite the orders being started by different actors, all orders contain a phase change. For example, when A01 orders bananas from A02 and A02 orders bananas from A05, the product kind resultant of these transaction kinds are exactly the same (an order was created). In the same way, when bananas are transported (T07), the product kind result is always the same (order transported), independently from its source or destination. There are other phases of the order. For example, the order is produced in Noboa, transformed, tested, transported and unloaded to the packhouse, treated and packaged. Afterwards, the order is transported to Zeebrugge, unloaded, tested and paid to Noboa personnel and finally invoiced.

Fig. 7 represents the ATD that was implemented in Hyperledger Composer (HC) (Linux Foundation 2018). In our implementation, the first step is the creation of the blockchain participants using an HC transaction script. Each ID participant follows the figure encompassing name and organization to which it belongs (in some cases a balance and a business network card). Next, roles are assigned to each participant, according to the modelled DEMO Actor Roles. After all identities have been issued, participants can connect to the business network by using their respective business network card. Then, each BP is created as an HC asset with the following properties: i) *instanceId*, an id that is allocated by a timestamp; ii) *instanceStatus*, state of the asset that changes when submitting transactions steps, (i. e., DEMO Ψ-theory: created, requested, promised, declined, executed, declared, accepted and rejected); iii) *details*, a description of what is being requested; iv) *kind*, the kind of DEMO BP; v) *numberinstance*, number of the instance of that transaction kind; vi) *initiator*, identifier of the initiator of the BP; vii) *executor*, identifier of the executor of the BP.

Participants submit HC transactions that modify the status of the *Transaction Instance* asset. The possible values of the *instanceStatus* referenced above correspond to states that a transaction went through according to the DEMO standard transaction pattern (Dietz and Mulder 2020). To change *instanceStatus*, the prototype verifies if some conditions are being fulfilled by a previously defined state machine, e. g., it is not possible to submit a Promise transaction, if *instanceStatus* is not equal to REQUESTED. If a participant tries to do so, when submitting a transaction, the system throws an error containing the message: “It is not possible to promise the BP instance”. This works for the other transactions as well. It is not possible to submit an *Accept* if *instanceStatus* is not equal to DECLARED. Also, the initiators only have the permission to submit initiators operations. This means that if a participant is an initiator of a transaction, he can only submit transactions of the type Request, Reject and Accept. In the same way, an executor can only submit transactions of the types: Promise, Decline, Execute and Declare.

The submission of the assets gets recorded in the Historian Registry as Historian Records entries, allowing participants to track the events in the blockchain business network. Each Historian Record contains the date of submission of the transaction time stamped, the type of transaction and the participant that has submitted it.

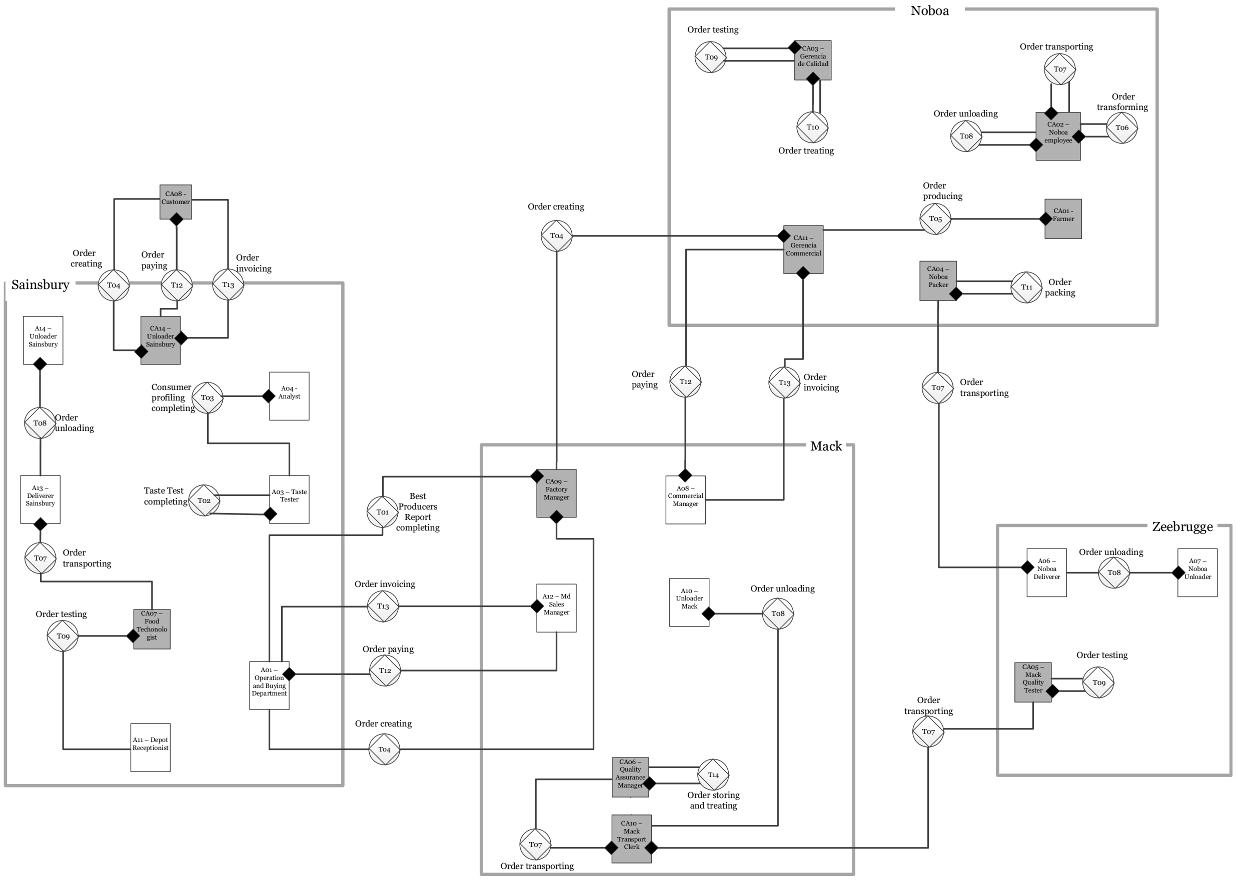


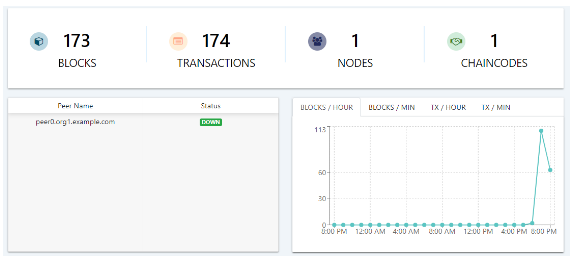
Figure 7: Food supply chain: DEMO Actor Transaction Diagram.

3.2.1 Enforcing dependencies between transactions

We implemented dependencies between transactions in the prototype. In DEMO there are two kinds of dependencies among BPs: *(i)* Request after Promise – request of a new BP is triggered after a promise of a previous; and *(ii)* Request after Accept – request of the new BP is only triggered when the previous one has been accepted. To implement these dependencies, three maps have been defined: *initiators*, *executors* and *dependencies*. The *initiators* map contains as key the kind of the transaction plus the number of the instance of that transaction kind (T01.1 for example) and contains as value the id of the initiator. The keys of the *Executors* map are equal to the initiators map and the values correspond to the executors' id. Keys from this map are similar to the keys from the previous maps. However, besides kind and number of the instance, it has also the type of dependency that will trigger another transaction. The *dependencies* map contains the dependencies among assets. The dependency type is represented by an 'A' if it is a request after accept and it is represented by a 'P' if it is a request after promise. Thus, the format of the key for this map is for example T01.1.A (kind T01, instance number 1 for that kind and request after accept of dependency). Dependency map values contain the transaction represented by its kind and number, that will be triggered by its key. If a pair of the map is ("T02.1.A","T03.1"), it means that after the acceptance of the transaction kind T02, instance number 1, a transaction of the kind T03 instance number 1 is triggered and requested. Depending on the type of dependency, transaction processor functions that implement the *Accept* and *Promise* transactions, read these maps and if the current transaction is present in the keys of dependencies map, create a new transaction instance of the type presented in the value of that key with its state as REQUESTED and fills the initiator and executor fields conforming initiators and executors map.

3.3 Results

As explained in the previous section, there is a type of asset named Transaction Instance that represents a BP instance of DEMO and suffers changes of state across the process. In the prototype implemented for this approach, each HC transaction corresponds to a BP step of the DEMO standard pattern (Dietz and Mulder 2020). To complete a DEMO BP kind, it is necessary to execute a set of HC transactions, i. e., a set of BP steps to complete the transaction pattern.

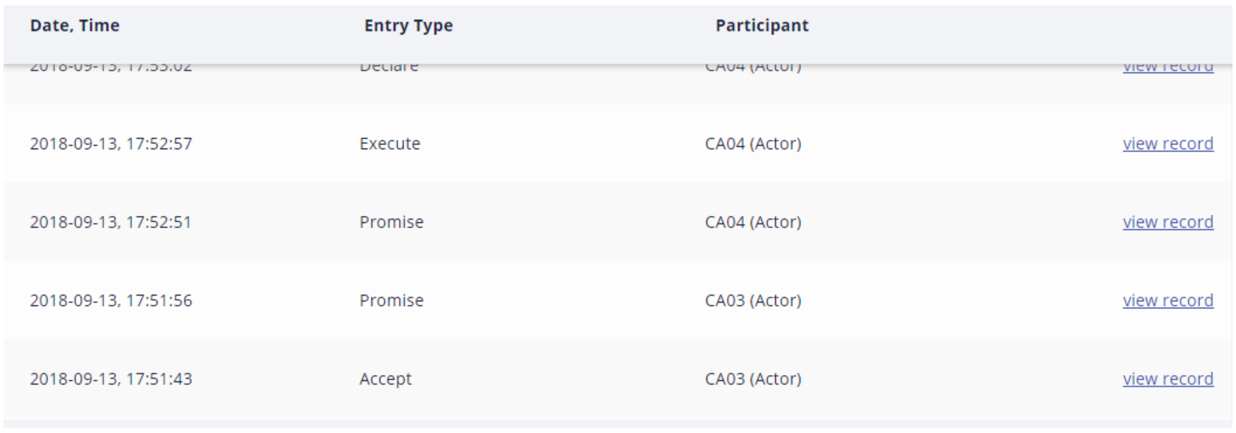


Peer Name	Status
peer0.org1.example.com	DOWN

Figure 8: Number of transactions necessary to execute experience 1.

For this reason, to complete a business collaborative scenario (as Sect. 3.1 shows) this approach implies the submission of a high number of transactions. Fig. 8 shows the Hyperledger Explorer dashboard in the deployment of this prototype in the simplest Fabric environment. It demonstrates that in this approach, it is necessary to submit 173 transactions to complete the scenario of Sect. 3.1 including the creation of the participants and issuing the identities for those participants.

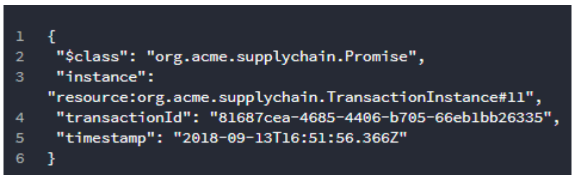
This prototype can detect unexpected situations in an objective manner. For example, suppose that a banana infection appears because the treaters from Noboa do not apply a pesticide on the fruits. A user detects this situation by observing that there are two Promise transactions next to each other (see Fig. 9). This could result from the fact of a Promise triggering a Request of a BP instance. However, the user notices that the transaction instance which the first *Promise* transaction acts is not concluded. Thus, the user consults in Fig. 10 the historic record of the first *Promise* transaction to know what BP Instance was not concluded. The



Date, Time	Entry Type	Participant	
2018-09-13, 17:53:02	Declare	CA04 (Actor)	view record
2018-09-13, 17:52:57	Execute	CA04 (Actor)	view record
2018-09-13, 17:52:51	Promise	CA04 (Actor)	view record
2018-09-13, 17:51:56	Promise	CA03 (Actor)	view record
2018-09-13, 17:51:43	Accept	CA03 (Actor)	view record

Figure 9: A user realizes the existence of two DEMO promises *next to each other*

user notices that the BP Instance corresponds to the process of treating the bananas, which was not concluded suggesting that the pesticides to the bananas were not applied and then caused the infection in the bananas.



```
{
 "$class": "org.acme.supplychain.Promise",
 "instance":
"resource:org.acme.supplychain.TransactionInstance#11",
 "transactionId": "81687cea-4685-4406-b705-66eb1bb26335",
 "timestamp": "2018-09-13T16:51:56.366Z"
}
```

Figure 10: The historian record of first DEMO promise *in Fig. 9*

4 Experience 2: Car Registration

4.1 Universe of Discourse

Car registration systems in Europe are managed by each European Union (EU) member state. The use of blockchain-based car registries may provide a single car registration system across EU member states and enhance BPs related to car registration. Considering the current car registration systems, multiple entities, ranging from tax authorities to different EU member states or even leasing companies, have frequent access to the car registration systems. A blockchain-based car registration system may simplify the access to car registries for the different parties involved, specifically the use case for information exchange across different member states. Furthermore, this application can benefit from the use of blockchain to provide an immutable trace of registry changes. The use of blockchain may also provide higher resilience to system faults, given the decentralized nature of this technology. Considering the Portuguese car registration system, registry employees are required to interact with the system for every modification and information request. However, a blockchain-based system may provide enhanced BP leading registry employees to a supervision role requiring to modify the system whenever a registry is wrongly updated.

4.2 Implementation

In the car registration system, entities interact with the system after their identity and permissions are verified. All updates to the car registry take place and are logged in the blockchain. A request is handled with minimal intervention from a registry employee, as long as all the information required is correctly provided. The full application's programming created with Hyperledger Composer is available in Annex A. Essentially we had to define the data model, the participants, the transactions (or operations supported), the access control list (permissions), and the business logic (the chaincode).

Let us consider the process of changing the ownership of a vehicle. The process can be divided into two steps (see Fig. 11). The ownership change process is started by the current vehicle's owner. The current owner is required to fill a form with all the necessary information and documentation. Once this task is complete, a pending ownership change is submitted to the blockchain. The new owner, identified in the initial task of the process, is then required to accept or reject the ownership change. Once the ownership change is confirmed by the new owner, the blockchain is updated with the new vehicle registry information and the prospect owner is registered as the vehicle owner.

During the execution of the different car registry operations, registry employees are required to supervise the operations and verify the information submitted to the blockchain. Thus in case of non-compliance, the request can be reverted by issuing a new information update replacing the vehicle's information to the values registered before the request was made. Most of the car registry operations, as a change of ownership, require the issuance of a new certificate of ownership (a printed document, in Portugal today). Certificates of ownership contain information regarding vehicle characteristics such as color, engine, and weight of the vehicle. This information is usually registered and controlled outside of the vehicle registry information system, by other government entities such as the Department of Motor Vehicles. However, the entity responsible for issuing a new certificate of ownership is usually the National Registry Entity.

Considering a car registration system based on blockchain, the BP control mechanisms executed by government entities needs to be adapted. Thus, a car registration system based on blockchain may distribute the system's maintenance effort and control to the network nodes. However, the government entities of each European member state can still have control over the registered cars in that state. As part of the system's requirements, the following use cases should be considered. The main use cases are: *(i)* transferring of car ownership from a seller to a buyer; *(ii)* registering a lease contract to a client during a defined period on which the leasing company is responsible for paying vehicle expenses in exchange for a defined monthly payment by the client; *(iii)* registering a vehicle as a guarantee to a credit, to be executed in case the vehicle owner misses credit payments. Secondary use cases are: *(i)* registering a newly produced car by request of a car manufacturer; *(ii)* executing a judicial order to liquidate a vehicle owner's debt, resulting in a transfer of ownership to the entity responsible for collecting the debt payments.

Considering the main and the secondary use cases specified, four participants are defined. A Natural Person, a Legal Person, a Judicial Officer and a Registry Employee. A Natural Person participant type is defined by a name and an address which represents the official residence of the entity and its fiscal number. A Natural Person can execute the following operations when owning a vehicle: Register a lease contract; Accept or reject a lease contract; Request to cancel a lease contract; Accept or reject a lease contract cancellation; Start a vehicle's ownership change; Add the vehicle as a guarantee for a loan; Request for a vehicle guarantee to be cancelled. A Legal Person represents a set of Natural Persons and requires the same information as a Natural Person. Also, a Legal Person requires a list of entities. These entities are Legal Person's owners or entities entitled to perform actions on the car registration system on behalf of a Legal Person. Judicial Officers are entitled to execute operations as fulfilment of a judicial order. As expected, a Judicial Officer is required to be registered, in the car registration system, as tied to the judicial entity for whom the Judicial Officer works. A Registry Employee defines users of the car registration system, working for national registry entities. These participants have unrestricted access to and permissions over the car registries but are still accountable for their actions in the system, given the properties of blockchain technology. Registry Employees, as mentioned, play a role of supervisors and can solve issues regarding car registries present in the blockchain-based car registration system. Person and company are defined as follows in Hyperledger Composer:

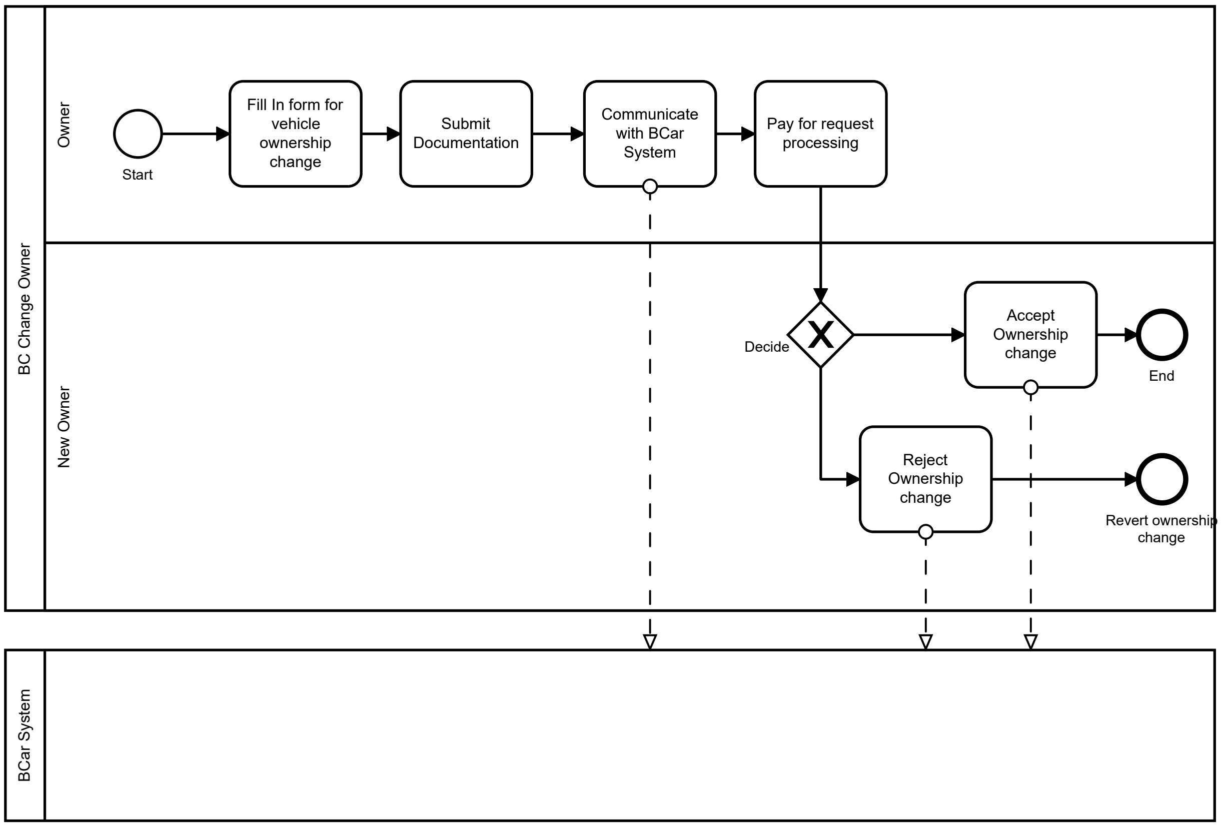


Figure 11: Business process for changing vehicle ownership.

```
participant Person
extends SingularEntity  {
}

participant Company
extends Entity {
    --> Person[] owners
}
```

The data model was created based on the European regulations (The Council of European Union 1999) and the use cases defined. A vehicle is an asset registered in the car registration system with the following data: a registration number, one or more owners, the vehicle identification number (VIN), a category (based on the classification in UNECE 2017), and several other attributes:

```
asset Vehicle identified by vin {
    o String registrationNumber
    --> Entity certificateHolder
    o Ownership[] owners
    o String vin
    o VehicleCategory category
    ...
}
```

The vehicle category is an enumerate with 4 possible values:

```
enum VehicleCategory {
    o M //Carrying passengers
    o N //Carrying goods
    o L // 2/3-wheel vehicles
        // and quadricycles
    o T //agricultural and forestry
        //tractors and trailers
}
```

Considering Fig. 11, a change of ownership needs to be initialized by the current owner wishing to give up his or her position on a certain vehicle. This operation is implemented in the proposed system, requiring the owner to issue a *Change Owner* transaction, specifying the vin, the registration number and the make of the vehicle as well as the list of new owners to which the current owner will give his or her share of the vehicle and the share percentage he or she wants to transfer to the new owners. This transaction is defined as follows:

```
transaction ChangeOwner {
    o String vin
    o String registrationNumber
    o String make
    o Ownership[] newOwners
}
```

When the blockchain receives that transaction, it executes the corresponding business logic, which is essentially a function (the full source code is in the annex):

```
function onChangeOwner(changeOwner) {
    ...
}
```

In order to complete the ownership change, the new owner registered in the initial *Change Owner* transaction, is required to issue a *Confirm Ownership* transaction, specifying the vehicle's VIN, registration number and make as well as the ownership share intended to be given to him or her. Only after this step, the ownership information is considered valid, regarding judicial obligations and the new owner is considered to be the legitimate owner.

Based on the data model described, a similar data model was created using Hyperledger Composer (Linux Foundation 2018) and latter deployed on a Hyperledger Fabric version 1.1 based blockchain. Each transaction described was also modelled in Hyperledger Composer Modeling Language. Every information regarding the created data model was stored in the Hyperledger Fabric blockchain with no off-chain database handling car registry records. Specific access control rules were defined using Hyperledger Composer access control language.

4.3 Results

We do not provide a full experimental evaluation of the car registration system, but only some basic measurements. We used a Virtual Machine with 8vCPU, 32 GB RAM and 40 GB HDD space, running Ubuntu 16.04.5 LTS. Regarding the software setup, tests were performed on top of Docker containers, running Hyperledger Fabric x86 64 v1.1.0. As database we used the Hyperledger Fabric CouchDB image version x86 64 v0.4.. The

Hyperledger infrastructure was configured with an orderer and two organizations with a single peer each. A set of functions was selected to assess the system's overall performance (cf. Annex A). Thus, we evaluated the performance of the *Create Vehicle*, *Change Vehicle State*, *Change Ownership*, *Issue Seizure* and *Register as Guarantee* functions with various blocksizes (1MB, 2MB and 4MB) and send rates (50 to 200 transactions per second).

As expected, a higher block size provided a higher throughput (Fig. 12 and 13). On the other hand, a higher latency is also obtained. The maximum throughput achieved was 7.33 tps, with a 4 MB block size and a 1 second timeout, when issuing a *Change Ownership* transaction. However, the same transactions present a latency of 25.45 seconds. Considering the complexity of car registry operations, it is plausible that the quantity of information stored in the blockchain might provide a reason for such latency and throughput results. Using Hyperledger Fabric chaincode to develop car registration smart contracts instead of Hyperledger Composer framework might contribute to performance improvements. During the development and performance tests of the proposed car registration system it was noticeable the access control mechanisms of Hyperledger Composer contributed to slower execution of smart contract's code.

5 Lessons Learned

Sect. 3 and 4 present two experiments, each one considering a business application, but both using the same technological solution (Hyperledger Fabric) and both with a common need to control the asset's life cycle (fruit or car registration). In this section, we assess the business process modelling languages used (DEMO and BPMN) in the context of the blockchain implementations.

Regarding DEMO modelling, the first experiment has shown an overhead related to the need to implement all the BP steps (even the implicit ones) in the blockchain. The number of blockchain assets is higher than in the second experiment. However, this fine-grained approach helps to trace all the misalignment situations, as reported in Sect.3.3. This over-modelling activity could be considered harmless if the DEMO standard business transaction pattern (cf. Fig. 2) is directly inherited in the modelling phase.

On the other hand, regarding BPMN modelling, the second experiment created a specific blockchain data model that conformed to the specific, car registration, application and, more specifically, to the process of changing vehicle ownership. Any other business process, or a change in this business process, would require a change in the blockchain data model, which is undesirable. However, this design decision allowed us to give more focus to the optimization of that operation, revealing that this is an issue that also demands effort in a blockchain implementation concerning the domain of business processes.

During the requirements elicitation phase of both projects, the BP concept offered a common language that abstracted the technological details of the operation. This is a recommendation for future developments that is aligned with other related work (Hornáčková et al. 2018).

We believe that combining both approaches (ontological and blockchain algorithm optimization) could boost a more balanced implementation where business process change flexibility could be optimized in terms of performance. Firstly, the ontological definition of business processes clarifies the expected outcomes from the blockchain application: actors are identified, communication between actors is anticipated and the assets to be stored in the chain clarified. In fact, ontology is a solution that could be applied when use case scenario discussions between stakeholders are needed. Nevertheless, as a second concern, the Blockchain implementation (e. g., smart contracts development) demands a detailed, technical, approach. In the end, some technological decisions need to be addressed (e. g., choice of operation project) and optimization recalls from those decisions.

In both experiments, an initial high learning effort to manage the complexity of deploying and programming Hyperledger Fabric was noticed. The documentation and APIs were evolving at

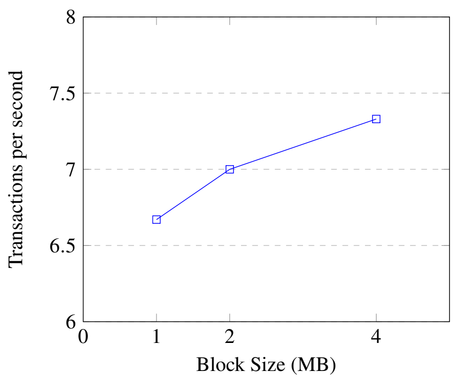


Figure 12: Throughput of Change Ownership function

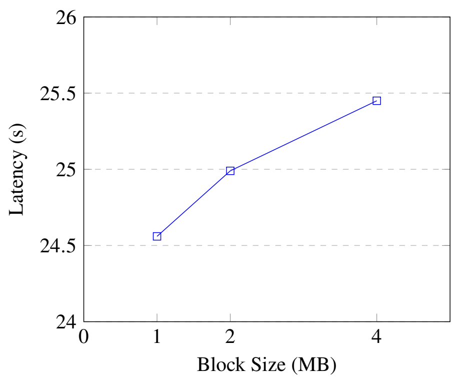


Figure 13: Latency of Change Ownership function

the same time that the development teams were working, which caused some turbulence during the development process. This issue needs to be taken into account for future operation application projects. Therefore, as a conclusion, the exact solution to be optimized should be considered from the beginning of the ontological design.

6 Related Work

This section presents a review of the knowledge available in the literature related to the topics of blockchain and business processes. To that end, a search has been conducted using the Web of Science Core Collection (WOS), KCI-Korean Journal Database (KJD), Russian Science Citation Index (RSCI), Current Contents Connect (CCC), SciELO Citation Index (SCI-ELO), and MEDLINE® databases, considering the following topic searches: TS1=(*"Blockchain"*), TS2=(*"Blockchain"* AND *"business process"*) and TS3=(*"Blockchain"* AND *"business process"* AND *"control"*), until 2019 (included). The returned number of hits were TS1=5486, TS2=43 and TS3=3; the yearly distribution is shown in Tab. 2. On the one hand, TS1 returns the first references to Blockchain as being introduced in 2013, although Nakamoto's seminal paper was introduced 5 years before (Nakamoto 2008). The recent years of 2018 and 2019 reveal a clear increasing interest in Blockchain. On the other hand, the application of Blockchain to business processes is still very short (TS2), and even less when the concept of control is included (TS3). However, if *"business process"* and *"control"* are considered without the *"Blockchain"* term, then a larger set of papers is retrieved. Business process control is a research area with many contributions (Guerreiro 2020) that may improve the more recent area of Blockchain. Joining TS2 and TS3 results and removing the duplicate entries, only 43 papers are obtained: 26 are from journals and 17 from scientific conferences. Moreover, at this moment, few citations (around 400) are recorded for these 43 papers, and only 7 papers have more than 10 citations. This low citation level may reveal the novelty of the field that is still in an early stage, and it is also aligned with the results presented by the work of (Rouhani and Deters 2019). The 43 papers were analysed and selected, by the researchers, and the main findings are presented below. The following classification was used to organize the findings: *(i)* Blockchain research challenges, *(ii)* Blockchain industrial applications and *(iii)* Implementation proposals of business processes supported in Blockchain.

From this set of 43 papers, some of them reflect upon the current research challenges and about the future research that has to be addressed towards blockchain support for inter-organizational business processes (Mendling 2018; Mendling et al. 2018b). One of the recommended research directions is to develop information systems that are able to execute and monitor business processes on Blockchain, taking benefit from the

Table 2: Yearly distribution of related literature for each interest topic using the databases of WOS, KJD, RSCI, CCC, SCIELO and MEDLINE®.

	TS1=("Blockchain")	*TS2*=("Blockchain" AND "business process")	*TS3*=("Blockchain" AND "business process" AND "control")
2019	2697	20	1
2018	1977	20	2
2017	638	2	0
2016	140	1	0
2015	22	0	0
2014	10	0	0
2013	2	0	0
total	*5486*	*43*	*3*

knowledge available from software engineering and distributed systems. Moreover, it is recommended to investigate blockchain-aware business processes (Sturm et al. 2019). Falazi et al. 2019 define this terminology as a way to capture the semantics of blockchain-based systems and assists in modelling fine-grained decisions when handling the uncertainty of blockchain transactions. Also, Madakam et al. 2019 refer that Blockchain could be a positive aid to the development of Robotics Process Automation. Similarly, Mendling et al. 2018a refer Machine Learning, Robotics Process Automation and Blockchain as emerging technologies for the business process management field that need to take the human factor in account when designing and implementing solutions.

Other papers are focused on the general terms of Blockchain and its application to the industry. The most cited paper referring to the importance of investing in Blockchain research is Xu et al. 2018 with its identification of Industry 4.0 trends. Viriyasitavat and Hoonsopon 2019 identify the following industries where Blockchain can have a positive impact: Banking and Payment, Insurance, Digital Supply Chain, Energy Management, Healthcare, Voting and Recruitment. Perboli et al. 2018 present a solution for a fresh food supply chain based on the previous knowledge that exists in the finance applications related to Blockchain. The main savings identified with the usage of Blockchain are related with data accuracy, optimization of operations, reduction of the waste of goods, counterfeit reduction, increase in brand image and reduction of the impact of the costs given by the recalls for possible contamination. Pourheidari et al. 2018 propose a solution for an Order Processing system, that besides the advantage of control approval for entities to join (offered by a permissioned ledger) uses a Role-based access control that can be used to define the access levels over parties' information, their assets, transactions, and the process flow's data. Regarding the integration of industrial application with business processes, Da Xu and Viriyasitavat 2019 detail an IoT (Internet of Things) solution to be integrated with industrial business processes. A smart contract for establishing the trust of process executions that fits into the IoT environment is presented.

In terms of implementation proposals that use Blockchain to support business processes, Weber et al. 2016 developed a technique to integrate Blockchain with BPMN (OMG 2011) choreography without a central authority but trust maintained. López-Pintado et al. 2019 use Ethereum (Wood 2014), in specific solidity smart contracts, to develop an open-source blockchain-based BPMN execution engine. This solution is expected to allow the execution of any BPMN model. Also, Yu et al. 2017 explore smart contracts as a way to execute business process models that include sequential, parallel and non-blocking constructs. Moreover, Roth and Djoua 2018 research the ability of a message exchange system supported by Blockchain to maintain the business process execution state. Viriyasitavat et al. 2019 propose an architecture of business processes in Blockchain to overcome the problems of time inconsistency and consensus bias. The authors state that some Blockchain characteristics could be beneficial for cross-organizational business processes in the aspects of persistence, validity, and auditability.

In this line, Kruijff and Weigand 2017 use enterprise ontology to explain Blockchain technology. The authors describe this technology on three levels: datalogical, infological and essential. To make this distinction, the authors use the distinction axiom from ψ theory (Dietz 2015;

Dietz and Mulder 2020). For each level, the authors have build a domain model using UML as ontology language to explain how the concepts of that level relate with each other. The datalogical level corresponds to the lowest or technical level where Blockchain is described as a set of blocks and code. This level considers concepts such as *blocks*, *miners*, *mainchains* (which is the blockchain), *sidechains* (which is a chain that communicates with the mainchain for enhanced functionality) because this level corresponds to the level of data structures and data manipulation. The infological level describes Blockchain technology, not as a set of blocks and data, but as a distributed ledger system where goods can be transferred between participants. This ledger is understood as a set of accounts where transactions act as inputs and outputs. The inputs and outputs represent intangible or tangible assets exchanged between accounts. However, transactions need to follow a set of rules of engagement, (i. e., *smart contracts*) that are implemented as blockchain code. The essential level or business level describes the Blockchain as a set of commitments that creates something. This level is focused on the creations directly or indirectly by communication. *Commitment* is a key concept on this level since they are evaluated or established by communicative acts, the engine of communication. A commitment bounds an actor to something for example a promise and when agreed between two actors it represents a change in social reality. According to the authors, the content of this change constitutes an essential business transaction and due to the fact of Enterprise Ontology be not specific about the change, they have combined it with the Business Ontology of REA (Hunka and Zacek 2015). Thus, on this level, the authors explain that processes are represented by economic events which are performed by agents and affect a specific resource. *Stock flows* represent the relationship between events and resources and can generate flows of conversion and exchange. On this level, the performance of transactions and commitments can originate these flows which are events that affect a resource. Here, the authors explain the difference between these concepts (although both need to follow smart contracts). A transaction changes the economic reality and can exist on its own. Commitments represent promises of future stock-flow events fulfilled within the execution of those events and some of them are executed automatically. On the other hand, committed transactions are only executed when certain conditions are met and are saved irreversibly in the blockchain.

The combination of Blockchain and DEMO is also reported by Guerreiro et al. 2017 that propose a meta-model for the interoperability of secure business transactions. The aim is to solve the security risks involved in business transactions executions increasing trust, authenticity, robustness and traceability against fraud. Blockchain solutions are concerned with technological aspects and not with social ones, like the existence of human interaction when performing business transactions in electronic networks. Therefore, Enterprise Operating System (Guerreiro et al. 2013) (EOS), a model-driven software system that supports the business process operation founded in the ψ theory based on DEMO, is integrated with Blockchain to increase trust between stakeholders and cyber-security in order to enable the operation of multiple business transactions. Each actor of each company is responsible to initiate and execute the business transactions, where the runtime control of business transactions execution is performed by the EOS at the application level. At the technological level, the authors propose a private blockchain to provide trust and security between the companies and to allow actors to consult the performed business transactions.

Rimba et al. 2017 compare the computation cost between the business process execution on the blockchain (Ethereum) with a popular cloud service for the same purpose (Amazon Simple Workflow Service). Authors conclude that the Ethereum cost can be two orders of magnitude higher than on Amazon SWF.

Tab. 3 summarizes the previous introduced related work. On the one hand, it is noted that many bibliographic references propose to model and/or

Table 3: Summary of the related work. ○ *refers to no conceptual coverage;* ● *to full conceptual coverage.*

Bibliographic reference	*Blockchain research challenges*	*Blockchain industrial applications*	*Business processes supported in blockchain*	*Blockchain Technologies*	*Usage of BPMN*	*Usage of DEMO*	*Other aspects*
Mendling 2018	●	○	●		●	○	
Mendling et al. 2018b	●	○	●		●	○	
Sturm et al. 2019	●	○	○		●	○	BP-aware blockchain
Falazi et al. 2019	●	○	○		●	○	BP-aware blockchain
Madakam et al. 2019	●	○	○		○	○	Robotic Process Automation
Mendling et al. 2018a	●	○	○		○	○	Human factor
Xu et al. 2018	○	●	○		○	○	
Viriyasitavat and Hoonsopon 2019	○	●	○		○	○	
Perboli et al. 2018	○	●	○	Hyperledger Fabric	○	○	Business model canvas
Pourheidari et al. 2018	○	●	○	Hyperledger Fabric	●	○	
Da Xu and Viriyasitavat 2019	○	●	○	AWS PostgreSQL	○	○	
Weber et al. 2016	○	○	●	Smart contracts	●	○	
López-Pintado et al. 2019	○	○	●	Ethereum	●	○	
Yu et al. 2017	○	○	●	Smart contracts	●	○	
Roth and Djoua 2018	○	○	●		○	○	
Viriyasitavat et al. 2019	●	○	●		○	○	
Kruijff and Weigand 2017	○	○	●		○	●	
Guerreiro et al. 2017	○	○	●		○	●	
Guerreiro et al. 2013	○	○	●		○	●	
Rimba et al. 2017	○	○	●	Ethereum	●	○	

to execute business processes using BPMN. Yet, some other references emphasize upon the need to future use of approaches with richer semantics, *e.g.*, ontologies or process-aware approaches. On the other hand, regarding technologies, many papers present prototypes where the aspects related to performance in production environments are undervalued. Therefore, the blockchain ontology-based solutions and the blockchain implementation' optimization seems as two pertinent research topics.

7 Conclusions and Future Work

The increasing interest in blockchain-based solutions applied to complex networks of business processes raised the problem of how to enforce decentralized control in such conditions. A single blockchain solution does not fit all the business process configurations and industries, e. g., business processes included in a food supply chain where many people interact, are different from old-fashioned paper-based processes. Each industry, and each organization in particular, already has its business process definition with its own context execution. Therefore, for example, a blockchain asset has to be defined with contextualization extracted from business process definitions that are already in place (orders, payments, *etc.*).

In this context, this paper reports the implementation of two applications in Hyperledger Fabric and Hyperledger Composer and uses its outcomes to help understanding the benefits and pitfalls of such an endeavour. Implementation details are described to offer a body of knowledge to the business processes integrated with a blockchain platform.

From the data and team experience obtained in both applications, blockchain data models need to accommodate reuse and adaption mechanisms to support the change in business process models. Controlling a single set of static business process models (e. g. business rules) does not guarantee the ability to support change throughout time. While, from other perspective, a blockchain implementation requires optimization to be efficient and effective. It is a mixture of both concerns: flexibility and performance that is recommended as an outcome of these two projects.

The following aspects are identified as future work. (A) Develop a script generator to automatically encode BPMN and DEMO specifications into Hyperledger Fabric. (B) Develop a tool to manage the versioning of Blockchain data models. (C) Apply this approach to more applications domains.

References

Alter S. (2015) A Workaround Design System for Anticipating, Designing, and/or Preventing Workarounds. In: nterprise, Business-Process and Information Systems Modeling BPMDS 2015/EMMSAD 2015. Lecture Notes in Business Information Processing Vol. 214. Springer, pp. 489–498

Androulaki E., Barger A., Bortnikov V., Cachin C., Christidis K., De Caro A., Enyeart D., Ferris C., Laventman G., Manevich Y. et al. (2018) Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the 13th ACM EuroSys Conference. ACM, pp. 1–15

Bork D., Sinz E. J. (2013) Bridging the gap from a multi-view modelling method to the design of a multi-view modelling tool. In: Enterprise Modelling and Information Systems Architectures (EMISAJ) 8(2), pp. 25–41

Christidis K., Devetsikiotis M. (2016) Blockchains and smart contracts for the internet of things. In: IEEE Access 4, pp. 2292–2303

CONSENSYS (2020) Quorum https://consensys.net/quorum/

Corda (2020) Open-source blockchain platform for business https://www.corda.net/

Correia M. (2019) From Byzantine Consensus to Blockchain Consensus. In: Essentials of Blockchain Technology. CRC Press, pp. 41–80

Da Xu L., Viriyasitavat W. (2019) Application of blockchain in collaborative Internet-of-Things services. In: IEEE Transactions on Computational Social Systems 6(6), pp. 1295–1305

Décosse C., Molnar W. A., Proper H. A. (2014) What does DEMO Do? A qualitative analysis about DEMO in practice: founders, modellers and beneficiaries. In: Enterprise Engineering Working Conference. Lecture Notes in Business Information Processing Vol. 174. Springer, pp. 16–30

Dietz J. L. (2015) The PSI theory understanding human collaboration. Technical Report TRFIT-15-05. Faculty of Information Technology Czech. http://ciaonetwork.org/uploads/eewc2017/resources/other_docs/TEE-03%5C%20PSI%5C%20ES%5C%20v4.2%5C%20EO.pdf

Dietz J. L., Mulder H. (2020) Enterprise ontology: A Human-Centric Approach to Understanding the Essence of Organisation. Springer

Dumas M., La Rosa M., Mendling J., Reijers H. A. (2017) Fundamentals of business process management. Springer

Falazi G., Hahn M., Breitenbücher U., Leymann F. (2019) Modeling and execution of blockchain-aware business processes. In: SICS Software-Intensive Cyber-Physical Systems 34(2-3), pp. 105–116

Ferstl O. K., Sinz E. J. (1998) Modeling of business systems using SOM. In: Handbook on architectures of information systems. Springer, pp. 347–367

Fickinger T., Recker J. C. (2013) Construct Redundancy in Process Modeling Grammars: Improving the Explanatory Power of Ontological Analysis. In: Proceedings of the 21st European Conference on Information Systems. AIS, pp. 1–12

Guerreiro S. (2020) Conceptualizing on dynamically stable business processes operation: a literature review on existing concepts. In: Business Process Management Journal ahead-of-print, pp. 1–30

Guerreiro S., Guédria W., Lagerström R., van Kervel S. J. H. (2017) A Meta Model for Interoperability of Secure Business Transactions - Using BlockChain and DEMO. In: Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management Vol. 2. SciTePress, pp. 253–260

Guerreiro S., van Kervel S. J., Babkin E. (2013) Towards Devising an Architectural Framework for Enterprise Operating Systems. In: Proceedings of the 8th International Conference on Software Technologies. SciTePress, pp. 578–585

Guerreiro S., Tribolet J. (2013) Conceptualizing Enterprise Dynamic Systems Control For Run-Time Business Transactions. In: Proceedings of ECIS 2013, Research in Progress Vol. 5. AIS

Guerreiro S., Vasconcelos A., Tribolet J. (2012) Enterprise Dynamic Systems Control Enforcement of Run-Time Business Transactions. In: EEWC 2012: Advances in Enterprise Engineering VI. Lecture Notes in Business Information Processing Vol. 110. Springer, pp. 46–60

Hornáčková B., Skotnica M., Pergl R. (2018) Exploring a Role of Blockchain Smart Contracts in Enterprise Engineering. In: EEWC 2018: Advances in Enterprise Engineering XII. Lecture Notes in Business Information Processing Vol. 334. Springer, pp. 113–127

Hunka F., Zacek J. (2015) A new view of REA state machine. In: Applied Ontology 10(1), pp. 25–39

Hyperledger (2019) Annual Report https://www.hyperledger.org/learn/publications/hyperledger-annual-report

Hyperledger (2020) Hyperledger Burrow https://www.hyperledger.org/use/hyperledger-burrow

de Kruijff J., Weigand H. (2017) Understanding the Blockchain Using Enterprise Ontology. In: CAiSE 2017: Advanced Information Systems Engineering. Lecture Notes in Computer Science Vol. 10253. Springer, pp. 29–43

Linux Foundation (2018) Hyperledger Composer https://hyperledger.github.io/composer

López-Pintado O., Garcıa-Bañuelos L., Dumas M., Weber I., Ponomarev A. (2019) Caterpillar: A Business Process Execution Engine on the Ethereum Blockchainn. In: Software: Practice and Experience 49(7), pp. 1162–1193

Madakam S., Holmukhe R. M., Jaiswal D. K. (2019) The Future Digital Work Force: Robotic Process Automation (RPA). In: JISTEM-Journal of Information Systems and Technology Management 16, pp. 1–17

Mendling J. (2018) Towards blockchain support for business processes. In: BMSD 2018: Business Modeling and Software Design. Lecture Notes in Business Information Processing Vol. 319. Springer, pp. 243–248

Mendling J., Decker G., Hull R., Reijers H. A., Weber I. (2018a) How do Machine Learning, Robotic Process Automation, and Blockchains Affect the Human Factor in Business Process Management? In: Communications of the AIS 43, pp. 297–320

Mendling J., Weber I., Aalst W. V. D., Brocke J. V., Cabanillas C., Daniel F., Debois S., Ciccio C. D., Dumas M., Dustdar S. et al. (2018b) Blockchains for Business Process Management - Challenges and Opportunities. In: ACM Transactions on Management Information Systems 9(1), 4:1–4:16

Nakamoto S. (2008) Bitcoin: A peer-to-peer electronic cash system https://git.dhimmel.com/bitcoin-whitepaper/

OMG (2011) BPMN 2.0 Specification https://www.omg.org/spec/BPMN/2.0.2/PDF

Peck M. E. (2017) Blockchains: How They Work and Why They’ll Change the World. In: IEEE Spectrum 54(10), pp. 26–35

Peffers K., Tuunanen T., Rothenberger M. A., Chatterjee S. (2007) A Design Science Research Methodology for Information Systems Research. In: Journal of Management Information Systems 24(3), pp. 45–77

Perboli G., Musso S., Rosano M. (2018) Blockchain in Logistics and Supply Chain: A Lean Approach for Designing Real-World Use Cases. In: IEEE Access 6, pp. 62018–62028

Pourheidari V., Rouhani S., Deters R. (2018) A Case Study of Execution of Untrusted Business Process on Permissioned Blockchain. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). IEEE, pp. 1588–1594

Rimba P., Tran A. B., Weber I., Staples M., Ponomarev A., Xu X. (2017) Comparing blockchain and cloud services for business process execution. In: 2017 IEEE International Conference on Software Architecture (ICSA). IEEE, pp. 257–260

Roth U., Djoua T. N. (2018) Message exchange on base of a blockchain-based layered architecture. In: it - Information Technology 60(5-6), pp. 253–261

Rouhani S., Deters R. (2019) Security, Performance, and Applications of Smart Contracts: A Systematic Survey. In: IEEE Access 7, pp. 50759–50779

Rozinat A., van der Aalst W. M. P. (2008) Conformance Checking of Processes Based on Monitoring Real Behavior. In: Information Systems 33(1), pp. 64–95

Sinz E. (2018) A Short Comparison of Business Process Modelling Methods under the Perspective of Structure and Behaviour. In: Enterprise Modelling and Information Systems Architectures (EMISAJ) 13, pp. 63–68

Sousa J., Bessani A., Vukolic M. (2018) A Byzantine Fault-Tolerant Ordering Service for Hyperledger Fabric. In: Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. IEEE, pp. 51–58

Sturm C., Scalanczi J., Schönig S., Jablonski S. (2019) A Blockchain-based and resource-aware process execution engine. In: Future Generation Computer Systems 100, pp. 19–34

The Council of European Union (1999) Council Directive 1999/37/EC of 29 April 1999 on the registration documents for vehicles https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX%3A31999L0037

Underwood S. (2016) Blockchain beyond Bitcoin. In: Communications of the ACM 59(11), pp. 15–17

UNECE (2017) Consolidated Resolution on the Construction of Vehicles (R.E.3), ECE/TRANS/WP.29/78/Rev.6. United Nations Economic and Social Councli https://www.unece.org/fileadmin/DAM/trans/main/wp29/wp29resolutions/ECE-TRANS-WP.29-78r6e.pdf

Van Der Aalst W. (2011) Process mining: Discovery, Conformance and Enhancement of Business Processes. Springer

Viriyasitavat W., Da Xu L., Bi Z., Pungpapong V. (2019) Blockchain and Internet of Things for Modern Business Process in Digital Economy-the State of the Art. In: IEEE Transactions on Computational Social Systems 6(6), pp. 1420–1432

Viriyasitavat W., Hoonsopon D. (2019) Blockchain characteristics and consensus in modern business processes. In: Journal of Industrial Information Integration 13, pp. 32–39

Weber I., Xu X., Riveret R., Governatori G., Ponomarev A., Mendling J. (2016) Untrusted Business Process Monitoring and Execution Using Blockchain. In: BPM 2016: Business Process Management. Lecture Notes in Computer Science Vol. 9850. Springer, pp. 329–347

Wilson N. (1996) Supply chain management: a case study of a dedicated supply chain for bananas in the UK grocery market. In: Supply Chain Management: An International Journal 1(2), pp. 28–35

Winter R. (2008) Design science research in Europe. In: European Journal of Information Systems 17(5), pp. 470–475

Wood G. (2014) Ethereum: A secure decentralised generalised transaction ledger. In: Ethereum Project Yellow Paper 151, pp. 1–32

Xu L. D., Xu E. L., Li L. (2018) Industry 4.0: state of the art and future trends. In: International Journal of Production Research 56(8), pp. 2941–2962

Yu L., Tsai W.-T., Li G., Yao Y., Hu C., Deng E. (2017) Smart-contract execution with concurrent block building. In: 2017 IEEE Symposium on Service-Oriented System Engineering (SOSE). IEEE, pp. 160–167

A Car Registration Application

The car registration application was built using the Hyperledger Fabric as platform. Furthermore, the Hyperledger Composer v0.19.0 framework was used because it allows focusing on the business logic and data model implementation to solve the business problems, without the need to deep dive into platform specific configurations.

The car registration platform's data model can be split in three parts:

- The data model, mainly defining a vehicle, which is the main asset of our application (Sect. A.1);
- The participants that are not only modeled in the data model but also represent the only available roles/participants in the car registration system (Sect. A.2). We need to define those participants and their permissions or access and controls in order for the system to properly work (Sect. A.4).
- The transactions, i. e., the transactions supported in the system and the information required for each transaction to be processed (Sect. A.3).

Aside from the data model for the car registration system, the business logic is enforced through a set of functions (A.5). These functions are designed to process transactions and make changes to the data model's state based on the issuance of the transactions by the assigned participants. At last, an access control list (ACL) specifies the rules to which each transaction must comply to, in order to be validated by the network peers. Furthermore, those rules also define which participants have permissions to execute specific transactions and which elements of the data model they have access to.

A.1 DataModel

```
enum VehicleCategory {
    o M //Carrying passengers
    o N //Carrying goods
    o L // 2/3-wheel vehicles
        // and quadricycles
    o T //agricultural and forestry
        //tractors and trailers
}

// For VehicleCategory M or N
enum VehicleClass {
    o LIGHT //passenger cars and vans
    o HEAVY // trucks,
            //buses and coaches
}

enum VehicleState {
    o STOLEN
    o DESTRUCTED
    o INACTIVE
    o SUSPENDED
    o ACTIVE
}

enum LoanType{
    o COLLATERAL
    o MORTGAGE
}

enum State {
    o VALID
    o WAITING
    o WAITINGCANCELATION
}

asset Vehicle identified by vin {
    o String registrationNumber
    --> Entity certificateHolder
    o Ownership[] owners
    o LeaseInfo lease optional
    o Loan loan optional
    o Seizure seizure optional
    o String make
    o String typeVariant optional
    o String typeVersion optional
    o String ComercialDesc optional
    // Vehicle Identification Number
    o String vin
    o Integer maxLadenMass optional
    o VehicleCategory category
    //restricted to M or N
    o VehicleClass vClass optional
    o Engine engine
    // power/weight ration (moto)
    o Double pwRatio optional
    o Emissions emissions
    o VehicleState state
}

/*
*   Concepts
*/

concept Engine {
    o Integer capacity
    o Integer maxNetPower optional
    o String fuelType
    o Integer ratedSpeed
    o String ein
}

concept Emissions {
    o Double emCO
    o Double emHC
    o Double emNOx
    o Double emHCNOx
    //diesel only
    o Double emParticles optional
    //diesel only
    o Double emAbsortionCoefficient
optional
    o Double emCO2
    o Double emFuelConsumption
}

concept Ownership {
    --> Entity owner
    --> Entity newOwner optional
    o Double share
    o State state
}

concept LeaseInfo {
    --> Entity issuer
    --> Entity lessee
    o DateTime startDate
    o DateTime endDate
    o Double totalValue
    o State state
}

concept Loan {
    --> Entity creditor
    o LoanType type
    o Double totalValue
    o Double penalty
    o Boolean waitingCancelation
optional
}

concept Seizure {
    --> Entity owner
    --> Entity creditor
    o Double totalValue
    o State status
}
```

A.2 Participants

```
abstract participant Entity
identified by fn {
    o String name
    o String address
    o String fn //fiscal Number
}

abstract participant SingularEntity
extends Entity {
    o String surname
    o String idNumber optional
}

participant Person
extends SingularEntity  {
}

participant Company
extends Entity {
    --> Person[] owners
}

participant JudicialOfficer
extends SingularEntity {
    o String court
}

participant ExternalEntity
extends SingularEntity {

}

participant RegistryEmployee
extends SingularEntity {
    o String employeeNr
}
```

A.3 Transactions

```
transaction CreateVehicle {
  o Vehicle vehicle
}

transaction ChangeOwner {
    o String vin
    o String registrationNumber
    o String make
    o Ownership[] newOwners
}

transaction ConfirmOwnership {
    o String vin
    o String registrationNumber
    o String make
    --> Entity newOwner
}

transaction CancelOwnership
extends ConfirmOwnership {
}

transaction CreateLease {
  o String vin
  o String registrationNumber
  o String make
  o DateTime startDate
  o DateTime endDate
  o Double totalValue
  --> Entity lessee
}

transaction ConfirmLease {
  o String vin
  o String registrationNumber
  o String make
  o Double totalValue
  --> Entity lessee
  --> Entity lessor

}

transaction CancelLease
extends ConfirmLease {

}


transaction ConfirmLeaseTermination {
    o String vin
    o String make
    o String registrationNumber
}

transaction CancelLeaseTermination
extends ConfirmLeaseTermination {

}

transaction RegisterAsGuarantee {
    --> Entity creditor
    o LoanType type
    o String vin
    o String registrationNumber
    o String make
    o Double totalValue
    o Double penalty
}

transaction CancelGuarantee{
    o String vin
    o String registrationNumber
    o String make
}

transaction ConfirmGuaranteeCancelation
extends CancelGuarantee {
}
transaction RejectGuaranteeCancelation
extends CancelGuarantee {
}

transaction IssuePendingSeizure{
    --> Entity owner
    o Double totalValue
    o String vin
    o String registrationNumber
    o String make
    --> Entity creditor
}

transaction IssueSeizure
extends IssuePendingSeizure{
    o DateTime date
    o String orderNumber
}

transaction CancelSeizure
extends IssuePendingSeizure{
}

transaction ChangeState{
    o String vin
    o String registrationNumber
    o String make
    o VehicleState newState
}
```

A.4 Access Control List

```
/*
        NETWORK ACCESS RULES
*/

rule networkControlPermission {
  description:
"networkControl␣can␣access␣network
commands"
  participant:
"org.hyperledger.composer.system
.NetworkAdmin"
  operation: ALL
  resource:
"org.hyperledger.composer.system.**"
  action: ALLOW
}

rule EntityHasAllReadAccess {
        description: "Allow␣all
participants␣read␣access␣to␣all
network␣resources"
        participant: "com.bcar.Entity"
        operation: ALL
        resource:
"org.hyperledger.composer.system.**"
        action: ALLOW
}

/*
        ASSET OPERATION RULES
*/
rule networkAdminHasAllAcess {
        description: "Allow␣network
admin␣to␣access␣all␣resources"
        participant:
"org.hyperledger.composer.system
.NetworkAdmin"
        operation: ALL
        resource: "com.bcar.**"
        action: ALLOW
}

rule EntityHasAllReadVehicle {
        description:
"Allow␣Persons␣read␣access␣to␣Vehicles"
        participant: "com.bcar.Entity"
        operation: READ
        resource: "com.bcar.Vehicle"
        action: ALLOW
}

rule ExternalEntityHasAllRead {
  description: "Allow␣trusted␣external
entities␣to␣access␣Entity␣information"
  participant: "com.bcar.ExternalEntity"
  operation: READ
  resource: "com.bcar.Entity"
  action: ALLOW
}

rule EntityReadParticipantsGuarantee {
        description: "Allow␣Persons␣read␣
            access␣to
Participant␣list␣when␣registering␣a␣
    Guarantee"
        participant: "com.bcar.Entity"
        operation: READ
        resource: "com.bcar.Entity"
        transaction: "com.bcar.
            RegisterAsGuarantee"
        action: ALLOW
}
rule EntityReadParticipantsCancelGuarantee
     {
        description: "Allow␣Persons␣read␣
            access␣to
Participant␣list␣when␣canceling␣a␣
    Guarantee"
        participant: "com.bcar.Entity"
        operation: READ
        resource: "com.bcar.Entity"
        transaction: "com.bcar.
            CancelGuarantee"
        action: ALLOW
}

rule EntityHasAllReadParticipantsLease {
        description: "Allow␣Persons␣read␣
            access␣to
Participant␣list␣when␣registering␣a␣Lease"
        participant: "com.bcar.Entity"
        operation: READ
        resource: "com.bcar.Entity"
        transaction: "com.bcar.CreateLease
            "
        action: ALLOW
}
```

```
/*
        PERMISSIONS ON TRANSACTIONS
*/

// Change Vehicle Transaction
rule RegistryEmployeeCanCreateVehicleTx{
  description: "Description␣of␣the␣
      Transactional
ACL␣rule"
  participant: "com.bcar.RegistryEmployee"
  operation: ALL
  resource: "com.bcar.CreateVehicle"
  action: ALLOW
}

rule RegistryEmployeeCanCreateVehicle {
  description: "Description␣of␣the␣
      Transactional
ACL␣rule"
  participant: "com.bcar.RegistryEmployee"
  operation: ALL
  resource(res): "com.bcar.Vehicle"
  transaction(tx): "com.bcar.CreateVehicle
      "
  condition: (tx.vehicle.type() === res.
      type())
  action: ALLOW
}

//      Change Owner Transaction

rule EntityCanCreateChangeOwnerTrans {
        description: "Allow␣Persons␣to␣
            create
ChangeOwner␣transactions"
        participant: "com.bcar.Entity"
        operation: CREATE
        resource: "com.bcar.ChangeOwner"
        action: ALLOW
}

rule OwnerCanChangeOwner {
        description: "Allow␣a␣owner␣to␣
            change
ownership␣of␣his␣car␣through␣ChangeOwner␣
    transaction"
        participant: "com.bcar.Entity"
        operation:  READ, UPDATE
        resource: "com.bcar.Vehicle"
        transaction: "com.bcar.ChangeOwner
            "
        action: ALLOW
}

// Confirm Ownership Transaction

rule EntityConfirmOwnershipTrans {
        description: "Allow␣all␣
            participants␣to
create␣confirmOwnership␣transaction"
        participant: "com.bcar.Person"
        operation: CREATE
        resource: "com.bcar.
            ConfirmOwnership"
        action: ALLOW
}

rule newOwnerUpdateVehicle {
        description: "Allow␣new␣owners␣to␣
            update␣vehicle␣information␣
            regarding␣ownership␣
            confirmation"
        participant: "com.bcar.Person"
        operation: UPDATE
        resource(r): "com.bcar.Vehicle"
        transaction(tx): "com.bcar.
            ConfirmOwnership"
  condition: (tx.vin == r.vin && tx.
      registrationNumber == r.
      registrationNumber && tx.make == r.
      make)
  action: ALLOW
}

// Create Lease
rule EntityCanCreateLease {
        description: "Allow␣all␣entities␣
            to␣create␣lease␣contract␣
            transaction"
        participant: "com.bcar.Entity"
        operation: CREATE
        resource: "com.bcar.CreateLease"
        action: ALLOW
}

rule EntityCanLeaseCar {
        description: "Allow␣all␣owners␣can
            ␣lease␣their␣vehicles"
        participant: "com.bcar.Entity"
        operation: UPDATE
        resource(r): "com.bcar.Vehicle"
        transaction(tx): "com.bcar.
            CreateLease"
  condition: (tx.vin == r.vin && tx.
      registrationNumber == r.
      registrationNumber && tx.make == r.
      make)
        action: ALLOW
}

// Confirm Lease
rule PersonConfirmLeaseTransaction {
        description: "Allow␣all␣
            participants␣read␣access␣to␣
            all␣resources"
        participant: "com.bcar.Person"
        operation: CREATE
        resource: "com.bcar.ConfirmLease"
    action: ALLOW
}

rule PersonUpdateVehicleConfirmLease {
    description: "Allow␣all␣participants␣
        read␣access␣to␣all␣resources"
    participant(p): "com.bcar.Person"
    operation: READ, UPDATE
    resource(r): "com.bcar.Vehicle"
    transaction(tx): "com.bcar.
        ConfirmLease"
    condition:
        (r.vin == tx.vin &&
            r.make == tx.make &&
            r.lease != undefined &&
            r.lease.totalValue == tx.
                totalValue)
    action: ALLOW
}

rule PersonReadEntityConfirmLease {
    description: "Allow␣all␣participants␣
        read␣access␣to␣all␣resources"
    participant: "com.bcar.Person"
    operation: READ
    resource: "com.bcar.Entity"
    transaction: "com.bcar.ConfirmLease"
    action: ALLOW
}

// Cancel Lease Transaction
rule PersonCancelLeaseTransaction {
        description: "Allow␣all␣
            participants␣read␣access␣to␣
            all␣resources"
        participant: "com.bcar.Person"
        operation: CREATE
        resource: "com.bcar.CancelLease"
    action: ALLOW
}

rule JudicialOfficerCancelLeaseTransaction
    {
    description: "Allow␣all␣participants␣
        read␣access␣to␣all␣resources"
    participant: "com.bcar.JudicialOfficer
        "
    operation: CREATE
    resource: "com.bcar.CancelLease"
    action: ALLOW
}

rule
    JudicialOfficerUpCancelLeaseTransaction
     {
    description: "Allow␣all␣participants␣
        read␣access␣to␣all␣resources"
    participant(p): "com.bcar.
        JudicialOfficer"
    operation: READ, UPDATE
    resource(r): "com.bcar.Vehicle"
    transaction(tx): "com.bcar.CancelLease
        "
    condition:
        (r.vin == tx.vin &&
            r.make == tx.make &&
            r.lease != undefined &&
            r.lease.totalValue == tx.
                totalValue)
    action: ALLOW
}

rule
    JudicialOfficerReadEntitiesCancelLease
     {
    description: "Allow␣all␣participants␣
        read␣access␣to␣all␣resources"
    participant: "com.bcar.JudicialOfficer
        "
    operation: READ
    resource: "com.bcar.Entity"
    transaction: "com.bcar.CancelLease"
    action: ALLOW
}

rule
    RegistryEmployeeUpdateVehicleCancelLease
     {
    description: "Allow␣all␣participants␣
        read␣access␣to␣all␣resources"
    participant(p): "com.bcar.
        RegistryEmployee"
    operation: READ, UPDATE
    resource(r): "com.bcar.Vehicle"
    transaction(tx): "com.bcar.CancelLease
        "
    condition:
        (r.vin == tx.vin &&
            r.make == tx.make &&
            r.lease != undefined &&
            r.lease.totalValue == tx.
                totalValue)
    action: ALLOW
}

rule
    RegistryEmployeeReadEntitiesCancelLease
     {
    description: "Allow␣all␣participants␣
        read␣access␣to␣all␣resources"
    participant: "com.bcar.
        RegistryEmployee"
    operation: READ
    resource: "com.bcar.Entity"
    transaction: "com.bcar.CancelLease"
    action: ALLOW
}


rule PersonUpdateVehicleCancelLease {
    description: "Allow␣all␣participants␣
        read␣access␣to␣all␣resources"
    participant(p): "com.bcar.Person"
    operation: READ, UPDATE
    resource(r): "com.bcar.Vehicle"
    transaction(tx): "com.bcar.CancelLease
        "
    condition:
        (r.vin == tx.vin &&
            r.make == tx.make &&
            r.lease != undefined &&
            r.lease.totalValue == tx.
                totalValue)
    action: ALLOW
}

rule PersonReadEntitiesCancelLease {
    description: "Allow␣all␣participants␣
        read␣access␣to␣all␣resources"
    participant: "com.bcar.Person"
    operation: READ
    resource: "com.bcar.Entity"
    transaction: "com.bcar.CancelLease"
    action: ALLOW
}


// Register a vehicle as Guarantee

rule EntityCanRegisterVehicleAsGuarantee {
        description: "Allow␣all␣
            participants␣to␣register␣a␣
            vehicle␣as␣Guarantee␣for␣a␣
            loan"
        participant: "com.bcar.Person"
        operation: CREATE
        resource: "com.bcar.
            RegisterAsGuarantee"
        action: ALLOW
}

rule EntityCanUpdateVehicleAsGuarantee {
        description: "Allow␣participants␣
            to␣read␣and␣update␣vehicle␣
            info␣regarding␣collatrals"
        participant: "com.bcar.Entity"
        operation: READ, UPDATE
        resource(r): "com.bcar.Vehicle"
        transaction(tx): "com.bcar.
            RegisterAsGuarantee"
        condition: (tx.vin == r.vin && tx.
            registrationNumber == r.
            registrationNumber && tx.make
            == r.make)
        action: ALLOW
}

// Cancel a vehicle as Guarantee

rule EntityCanCancelVehicleAsGuarantee {
        description: "Allow␣all␣
            participants␣to␣cancel␣a␣
            vehicle␣as␣Garantee␣for␣a␣loan
            "
        participant: "com.bcar.Entity"
        operation: CREATE
        resource: "com.bcar.
            CancelGuarantee"
        action: ALLOW
}

rule EntityUpdateVehicleCancelGuarantee {
        description: "Allow␣participants␣
            to␣read␣and␣update␣vehicle␣
            info␣regarding␣collatrals"
        participant: "com.bcar.Entity"
        operation: READ, UPDATE
        resource(r): "com.bcar.Vehicle"
        transaction(tx): "com.bcar.
            CancelGuarantee"
        condition: (tx.vin == r.vin && tx.
            registrationNumber == r.
            registrationNumber && tx.make
            == r.make)
        action: ALLOW
}

// Confirm guarantee cancelation

rule EntityConfirmGuaranteeCancelVehicle {
        description: "Allow␣all␣
            participants␣to␣cancel␣a␣
            vehicle␣as␣Garantee␣for␣a␣loan
            "
        participant: "com.bcar.Entity"
        operation: CREATE
        resource: "com.bcar.
            ConfirmGuaranteeCancelation"
        action: ALLOW
}

rule
    EntityUpdateVehicleConfirmGuaranteeCancel
     {
        description: "Allow␣participants␣
            to␣read␣and␣update␣vehicle␣
            info␣regarding␣collatrals"
        participant: "com.bcar.Entity"
        operation: READ, UPDATE
        resource(r): "com.bcar.Vehicle"
        transaction(tx): "com.bcar.
            ConfirmGuaranteeCancelation"
        condition: (tx.vin == r.vin && tx.
            registrationNumber == r.
            registrationNumber && tx.make
            == r.make)
        action: ALLOW
}


// Reject guarantee cancelation

rule EntityRejectGuaranteeCancelVehicle {
        description: "Allow␣all␣
            participants␣to␣cancel␣a␣
            vehicle␣as␣Guarantee␣for␣a␣
            loan"
        participant: "com.bcar.Entity"
        operation: CREATE
        resource: "com.bcar.
            RejectGuaranteeCancelation"
        action: ALLOW
}

rule
    EntityUpdateVehicleRejectGuaranteeCancel
     {
        description: "Allow␣participants␣
            to␣read␣and␣update␣vehicle␣
            info␣regarding␣collateral"
        participant: "com.bcar.Entity"
        operation: READ, UPDATE
        resource(r): "com.bcar.Vehicle"
        transaction(tx): "com.bcar.
            RejectGuaranteeCancelation"
        condition: (tx.vin == r.vin && tx.
            registrationNumber == r.
            registrationNumber && tx.make
            == r.make)
        action: ALLOW
}
```

```
// IssuePendingSeizure

rule JudicialOfficerCanIssuePendingSeizure
     {
        description: "Allow␣judicial␣
            officer␣to␣create␣a␣pending␣
            seizure␣transaction"
        participant: "com.bcar.
            JudicialOfficer"
        operation: CREATE
        resource: "com.bcar.
            IssuePendingSeizure"
        action: ALLOW
}

rule JudicialOfficerIssuePendingSeizure{
        description: "Allow␣judicial␣
            officer␣to␣update␣a␣vehicle␣
            via␣pending␣seizure␣
            transaction"
        participant: "com.bcar.
            JudicialOfficer"
        operation: UPDATE
        resource(r): "com.bcar.Vehicle"
        transaction(tx): "com.bcar.
            IssuePendingSeizure"
        condition: (tx.vin == r.vin && tx.
            registrationNumber == r.
            registrationNumber && tx.make
            == r.make)
        action: ALLOW
}

// CancelVehicleSeizure

rule JudicialOfficerCanCancelSeizure {
        description: "Allow␣judicial␣
            officer␣to␣cancel␣a␣seizure"
        participant: "com.bcar.
            JudicialOfficer"
        operation: CREATE
        resource: "com.bcar.CancelSeizure"
        action: ALLOW
}

rule JudicialOfficerVeihcleCancelSeizure {
        description: "Allow␣judicial␣
            officer␣to␣update␣a␣vehicle␣
            via␣cancel␣seizure␣transaction
            "
        participant: "com.bcar.
            JudicialOfficer"
        operation: UPDATE
        resource(r): "com.bcar.Vehicle"
        transaction(tx): "com.bcar.
            CancelSeizure"
        condition: (tx.vin == r.vin && tx.
            registrationNumber == r.
            registrationNumber && tx.make
            == r.make)
        action: ALLOW
}

// Confirm Lease Termination Transaction

rule
    PersonCanCreateConfirmLeaseTermination
     {
        description: "Allow␣all␣
            participants␣read␣access␣to␣
            all␣resources"
        participant: "com.bcar.Person"
        operation: CREATE
        resource: "com.bcar.
            ConfirmLeaseTermination"
    action: ALLOW
}

rule PersonVehicleConfirmLeaseTermination
    {
    description: "Allow␣all␣participants␣
        read␣access␣to␣all␣resources"
    participant(p): "com.bcar.Person"
    operation: READ, UPDATE
    resource(r): "com.bcar.Vehicle"
    transaction(tx): "com.bcar.
        ConfirmLeaseTermination"
    condition:
        (r.vin == tx.vin &&
            r.make == tx.make &&
            r.registrationNumber == tx.
                registrationNumber)
    action: ALLOW
}

rule PersonReadEntitiesConfirmLeaseTerm {
    description: "Allow␣all␣participants␣
        read␣access␣to␣all␣resources"
    participant: "com.bcar.Person"
    operation: READ
    resource: "com.bcar.Entity"
    transaction: "com.bcar.
        ConfirmLeaseTermination"
    action: ALLOW
}

rule RegistryEmployeeChangeState {
    description: "Allow␣registry␣employees
        ␣to␣create␣change␣vehicle␣state␣
        transactions"
    participant: "com.bcar.
        RegistryEmployee"
    operation: CREATE
    resource: "com.bcar.ChangeState"
    action: ALLOW
}

rule RegistryEmployeeVehicleChangeState {
    description: "Allow␣registry␣employees
        ␣to␣update␣vehicle␣state␣through␣
        change␣state␣transactions"
    participant: "com.bcar.
        RegistryEmployee"
    operation: UPDATE
    resource(r): "com.bcar.Vehicle"
    transaction(tx): "com.bcar.ChangeState
        "
    condition:
      (r.vin == tx.vin &&
        r.registrationNumber == tx.
            registrationNumber &&
        r.make == tx.make)
    action: ALLOW

}
```

A.5 Business Logic

```
"use␣strict";

var namespace = "com.bcar";
var JudicialOfficer = "JudicialOfficer";
var RegistryEmployee = "RegistryEmployee";
var Person = "Person";

var vehicleType = "Vehicle";
var vehicleClasses = ["LIGHT", "HEAVY"];
var DIESEL = "DIESEL";
var LeaseInfo = "LeaseInfo";
var Loan = "Loan";
var Company = "Company";
var Seizure = "Seizure";
var STATE_WAITING = "WAITING";
var STATE_VALID = "VALID";
var STATE_WAITINGCANCELATION = "
    WAITINGCANCELATION";
var STATE_ACTIVE = "ACTIVE";

function onCreateVehicle(createVehicle) {
  var newVehicle = createVehicle.vehicle;

  var qry = buildQuery(
    "SELECT␣" +
      namespace +
      "." +
      vehicleType +
      "␣WHERE␣(vin␣==␣␣$vin␣OR␣" +
      "engine.ein␣==␣␣$ein␣OR␣" +
      "registrationNumber␣==␣
          _$registrationNumber)"
  );

  return query(qry, {
    vin: newVehicle.vin,
    ein: newVehicle.engine.ein,
    registrationNumber: newVehicle.
        registrationNumber,
  }).then(function (result) {
    if (result.length >= 1) {
      throw new Error(
        "Vehicle␣with␣same␣vin␣or␣ein␣or␣
            registration␣" +
          "number␣already␣exists."
      );
    }

    switch (newVehicle.category) {
      case "M":
      case "N":
        if (vehicleClasses.indexOf(
            newVehicle.vClass) < 0) {
          throw new Error(
            "Vehicle␣from␣categories␣M␣or␣
                N␣" + "need␣to␣have␣a␣
                vehicle␣class."
          );
        }

      case "T":
        if (typeof newVehicle.pwRatio != "
            undefined") {
          throw new Error(
            "Vehicle␣from␣categories␣M,␣N␣
                or␣T␣" + "can␣not␣have␣
                pwRatio."
          );
        }
        if (typeof newVehicle.maxLadenMass
             === "undefined") {
          throw new Error(
            "Vehicle␣from␣categories␣M,␣N␣
                or␣T␣" + "need␣to␣have␣
                maxLadenMass."
          );
        }
        break;

      case "L":
        if (
          newVehicle.pwRatio <= 0 ||
          typeof newVehicle.pwRatio === "
              undefined"
        ) {
          throw new Error("Vehicle␣from␣
              category␣L␣" + "need␣to␣have
              ␣pwRatio.");
        }
        if (typeof newVehicle.maxLadenMass
             != "undefined") {
          throw new Error(
            "Vehicle␣from␣category␣L␣" + "
                can␣not␣have␣maxLadenMass.
                "
          );
        }
    }

    if (
      ["L", "T"].indexOf(newVehicle.
          category) >= 0 &&
      vehicleClasses.indexOf(newVehicle.
          vClass) >= 0
    ) {
      throw new Error(
        "Vehicle␣from␣categories␣L␣or␣T␣"
            + "can␣not␣have␣a␣vehicle␣
            class."
      );
    }

    if (
      newVehicle.engine.fuelType == DIESEL
          &&
      (typeof newVehicle.emissions.
          emParticles === "undefined" ||
        typeof newVehicle.emissions.
```

```
          emAbsorptionCoefficient === "
          undefined")
    ) {
      throw new Error(
        "Vehicle␣with␣DIESEL␣fuelType␣
            engines␣need␣" +
          "to␣have␣emParticles␣and␣
              emAbsorptionCoefficient."
      );
    }

    if (
      newVehicle.engine.fuelType != DIESEL
           &&
      typeof newVehicle.emissions.
          emParticles != "undefined" &&
      typeof newVehicle.emissions.
          emAbsorptionCoefficient != "
          undefined"
    ) {
      throw new Error(
        "Vehicle␣with␣non-DIESEL␣fuelType␣
            engines␣can␣not" +
          "␣have␣emParticles␣and␣
              emAbsorptionCoefficient."
      );
    }

    return getAssetRegistry(namespace + ".
        " + vehicleType).then(function (
      vehicleRegistry
    ) {
      newVehicle.state = STATE_ACTIVE;
      vehicleRegistry.add(newVehicle);
    });
  });
}

/**
 * Ownership change transactions
 */

function onChangeOwner(changeOwner) {
  var factory = getFactory();

  return getAssetRegistry(namespace + "."
      + vehicleType).then(function (
    registry
  ) {
    return registry.get(changeOwner.vin).
        then(function (result) {
      var vehicleOwnership = new Array();
      var transactionIssuer =
          getCurrentParticipant();

      var ownerShare = 0;

      for (
        var i = 0;
        i < result.owners.length && result
            .owners.length != 0;
        i++
      ) {
        var ownerRelation = result.owners[
            i];

        if (
          ownerRelation.owner.
              getFullyQualifiedIdentifier
              () ==
            transactionIssuer.
                getFullyQualifiedIdentifier
                () ||
          (ownerRelation.owner.
              getFullyQualifiedType() ==
            namespace + "." + Company &&
            isCompanyOwner(
                transactionIssuer,
                ownerRelation.owner))
        ) {
          vehicleOwnership.push(
              ownerRelation);
          ownerShare += ownerRelation.
              share;
          result.owners.splice(i, 1);
          i--;
        }
      }

      if (
        result.make != changeOwner.make ||
        result.registrationNumber !=
            changeOwner.registrationNumber
      ) {
        throw new Error(
          "Change␣owner␣data␣does␣not␣
              match" + "␣the␣vehicle␣
              registry."
        );
      } else if (ownerShare > 0) {
        // get sum of new shares added and
        // pushes new shares to vehicle to
             be later updated

        var newTotalShare = changeOwner.
            newOwners.reduce(function (
            total, obj) {
          if (vehicleOwnership[0].owner.
              $identifier == obj.owner.
              $identifier) {
            obj.state = STATE_WAITING;
            result.owners.push(obj);
            return total + obj.share;
          }
        }, 0);

        if (newTotalShare != ownerShare) {
          throw new Error(
            "Can␣not␣give␣more␣shares␣than
                ␣the␣original" + "␣owner␣
                has"
          );
        }
        return registry.update(result);
      } else {
        throw new Error(
          "Only␣the␣owner␣is␣capable␣of␣
              changing␣" + "vehicle␣
              ownership."
        );
      }
    });
  });
}

function onConfirmOwnership(
    confirmOwnership) {
  return getAssetRegistry(namespace + "."
      + vehicleType).then(function (
    registry
  ) {
    return registry.get(confirmOwnership.
        vin).then(async function (vehicle)
         {
      var issuer = getCurrentParticipant()
      .getFullyQualifiedIdentifier();
      if (
        issuer != confirmOwnership.
            newOwner
        .getFullyQualifiedIdentifier() ||
        (confirmOwnership.newOwner
        .getFullyQualifiedType() ==
          namespace + "." + Company &&
          !(await isCompanyOwner(issuer,
              confirmOwnership.newOwner)))
      ) {
        throw new Error(
          "Only␣the␣new␣owner␣can␣perform"
               + "␣confirm␣ownership."
        );
      }
      var needsUpdate = false;
      vehicle.owners.forEach(function (
          ownership) {
        if (
          ownership.newOwner != undefined
              &&
          (ownership.newOwner
          .getFullyQualifiedIdentifier()
          == issuer ||
            (confirmOwnership.newOwner
            .getFullyQualifiedIdentifier()
                 ==
              namespace + "." + Company &&
              isCompanyOwner(issuer,
                  confirmOwnership.
                  newOwner)))
        ) {
          ownership.owner = ownership.
              newOwner;
          ownership.state = STATE_VALID;
          ownership.newOwner = undefined;
          needsUpdate = true;
        }
      });
      if (!needsUpdate) {
        throw new Error(
          "There␣is␣no␣ownership␣
              confirmation" +
            "␣to␣be␣done␣for␣vehicle␣with␣
                vin:␣" +
            vehicle.vin
        );
      }
        return registry.update(vehicle);
      });
    });
}

function onCancelOwnershipChange(
    cancelOwnershipChange) {
  return getAssetRegistry(namespace + "."
      + vehicleType).then(function (
    registry
  ) {
    return registry
      .get(cancelOwnershipChange.vin)
      .then(async function (vehicle) {
        var issuer = getCurrentParticipant
            ();
        if (
      issuer !=
      cancelOwnershipChange.owner
      .getFullyQualifiedIdentifier() &&
      cancelOwnershipChange.owner
      .getFullyQualifiedType() ==
        namespace + "." + Company &&
      !(await
      isCompanyOwner(issuer,
          cancelOwnershipChange.owner)) &&
      !(await isOwner(issuer, vehicle))
        ) {
          throw new Error(
            "Only␣the␣new␣owner␣can␣
                perform" + "␣confirm␣
                ownership."
          );
        }
        var needsUpdate = false;
        vehicle.owners.forEach(function (
            ownership) {
          if (
    ownership.newOwner != undefined &&
    (ownership.newOwner.
        getFullyQualifiedIdentifier() ==
      issuer.getFullyQualifiedIdentifier()
           ||
      (cancelOwnershipChange.owner.
          getFullyQualifiedIdentifier() ==
        namespace + "." + Company &&
        isCompanyOwner(issuer,
            cancelOwnershipChange.owner))
            ||
      ownership.owner.
          getFullyQualifiedIdentifier() ==
        issuer.getFullyQualifiedIdentifier
            ())
          ) {
            ownership.state = STATE_VALID;
            ownership.newOwner = undefined
                ;
            needsUpdate = true;
          }
        });
        if (!needsUpdate) {
          throw new Error(
            "There␣is␣no␣ownership␣
                confirmation" +
              "␣to␣be␣done␣for␣vehicle␣
                  with␣vin:␣" +
              vehicle.vin
            );
          }

          return registry.update(vehicle);
        });
      });
    }

    /**
     *  Lease transactions
     */

    function onCreateLease(createLease) {
      return getAssetRegistry(namespace + "."
          + vehicleType).then(function (
        registry
      ) {
        return registry.get(createLease.vin).
            then(async function (result) {
          var factory = getFactory();

          var now = new Date();

          var issuer = getCurrentParticipant()
              ;
          if (!(await isOwner(issuer, result))
              ) {
            throw new Error("Only␣the␣owner␣
                can␣create␣a␣lease␣contract");
          }

          if (createLease.startDate.getTime()
              >= createLease.endDate.getTime()
              ) {
            throw new Error("End␣date␣must␣be␣
                greater␣than␣start␣date.");
          }
          if (now >= createLease.endDate.
              getTime()) {
            throw new Error("End␣date␣must␣be␣
                greater␣than␣current␣date.");
          }
          if (
            result.registrationNumber !=
                createLease.registrationNumber
                 ||
            result.make != createLease.make
          ) {
            throw new Error("Create␣Lease␣data
                ␣does␣not␣match␣the␣registry."
                );
          }

          var lease = factory.newConcept(
              namespace, LeaseInfo);
          lease.issuer = factory.
              newRelationship(
            namespace,
            Person,
            issuer.$identifier
          );
          lease.lessee = createLease.lessee;
          lease.startDate = createLease.
              startDate;
          lease.endDate = createLease.endDate;
          lease.totalValue = createLease.
              totalValue;
          lease.state = STATE_WAITING;
          result.lease = lease;

          return registry.update(result);
        });
      });
    }

    function onConfirmLease(acceptLease) {
      var issuer = getCurrentParticipant();
      return getAssetRegistry(namespace + "."
          + vehicleType).then((registry) => {
        return registry.get(acceptLease.vin).
            then(async (vehicle) => {
          if (
            vehicle.lease == undefined ||
            (!(await isLessee(issuer, vehicle)
                ) &&
              !(await isRegistryEmployee(
                  issuer)))
          ) {
            throw Error("Only␣authorized␣
                participants␣can␣perform␣this␣
                action.");
          }

          vehicle.lease.state = STATE_VALID;
          return registry.update(vehicle);
        });
      });
    }

    function onCancelLease(cancelLease) {
      var issuer = getCurrentParticipant();
      return getAssetRegistry(namespace + "."
          + vehicleType).then((registry) => {
        return registry.get(cancelLease.vin).
            then(async (vehicle) => {
          if (
            (await isJudicialOfficer(issuer))
                ||
            (await isRegistryEmployee(issuer))
          ) {
            vehicle.lease = undefined;
            return registry.update(vehicle);
          }

          if (
            !((await isOwner(issuer, vehicle))
                 || (await isLessee(issuer,
                vehicle)))
          ) {
            throw new Error(
              "Only␣the␣owner␣or␣the␣lessee␣
                  can␣issue␣a␣lease" + "␣
                  termination."
            );
          }
          if (vehicle.lease == undefined) {
```

```
        throw new Error("Vehicle␣does␣not␣
            have␣any␣active␣lease.");
      }
      vehicle.lease.issuer = getFactory().
          newRelationship(
        namespace,
        Person,
        issuer.$identifier
      );

      if (vehicle.lease.state ==
          STATE_WAITING) {
        vehicle.lease = undefined;
      } else {
        vehicle.lease.state =
            STATE_WAITINGCANCELATION;
      }
      return registry.update(vehicle);
    });
  });
}

function onConfirmLeaseTermination(
    confirmLeaseTermination) {
  var issuer = getCurrentParticipant();
  return getAssetRegistry(namespace + "."
      + vehicleType).then((registry) => {
    return registry.get(
        confirmLeaseTermination.vin).then(
        async (vehicle) => {
      if (
        !(await isOwner(issuer, vehicle))
            &&
        !(await isLessee(issuer, vehicle))
      ) {
        throw new Error(
          "Only␣the␣owner␣or␣the␣lessee␣
              can␣confirm␣a␣lease" + "␣
              termination."
        );
      }
      if (vehicle.lease == undefined) {
        throw new Error("Vehicle␣does␣not␣
            have␣any␣active␣lease.");
      }

      if (vehicle.lease.state !=
          STATE_WAITINGCANCELATION) {
        throw new Error(
          "A␣Cancel␣Lease␣Transaction␣
              needs␣to␣be" + "␣issued␣
              first."
        );
      }

      if (
        vehicle.lease.issuer.
            getFullyQualifiedIdentifier()
            ==
        issuer.getFullyQualifiedIdentifier
            ()
      ) {
        if (
          vehicle.lease.lessee.
              getFullyQualifiedIdentifier
              () ==
          issuer.
              getFullyQualifiedIdentifier
              ()
        ) {
          throw new Error("This␣action␣
              needs␣to␣be␣taken␣by␣the␣
              leaser.");
        } else {
          throw new Error("This␣action␣
              needs␣to␣be␣taken␣by␣the␣
              lessee.");
        }
      }

      vehicle.lease = undefined;
      return registry.update(vehicle);
    });
  });
}

function onCancelLeaseTermination(
    cancelLeaseTermination) {
  var issuer = getCurrentParticipant();
  return getAssetRegistry(namespace + "."
      + vehicleType).then((registry) => {
    return registry.get(
        cancelLeaseTermination.vin).then(
        async (vehicle) => {
      if (
        !((await isOwner(issuer, vehicle))
            || (await isLessee(issuer,
            vehicle)))
      ) {
        throw new Error(
          "Only␣the␣owner␣or␣the␣lessee␣
              can␣confirm␣a␣lease" + "␣
              termination"
        );
      }

      if (vehicle.lease == undefined) {
        throw new Error("Vehicle␣does␣not␣
            have␣any␣active␣lease.");
      }

      if (
        vehicle.lease.state !=
            STATE_WAITINGCANCELATION ||
        vehicle.lease.issuer.
            getFullyQualifiedIdentifier()
            !=
          issuer.
              getFullyQualifiedIdentifier
              ()
      ) {
        throw new Error(
          "A␣lease␣termination␣operation␣
              must" + "␣be␣issued␣first."
        );
      }

      vehicle.lease.state = STATE_VALID;
      vehicle.lease.issuer = getFactory().
          newRelationship(
        namespace,
        Person,
        issuer.$identifier
      );
      return registry.update(vehicle);
    });
  });
}

/*
    Guarantee transactions
*/

function onRegisterAsGuarantee(
    registerAsGuarantee) {
  return getAssetRegistry(namespace + "."
      + vehicleType).then(function (
    registry
  ) {
    return registry.get(
        registerAsGuarantee.vin).then(
        async function (result) {
      var issuer = getCurrentParticipant()
          ;
      if (!(await isOwner(issuer, result))
          ) {
        throw new Error(
          "Only␣the␣owner␣is␣capable␣of" +
            "␣registering␣the␣vehicle␣as␣
                loan␣garantee."
        );
      }

      if (result.seizure != undefined) {
        throw new Error("Vehicle␣should␣
            not␣have␣a␣pending␣seizure␣in␣
            place");
      }

      var factory = getFactory();
      var coll = factory.newConcept(
          namespace, Loan);
      coll.type = registerAsGuarantee.type
          ;
      coll.creditor = registerAsGuarantee.
          creditor;
      coll.totalValue =
          registerAsGuarantee.totalValue;
      coll.penalty = registerAsGuarantee.
          penalty;
      if (result.loan == null) {
        result.loan = coll;
        return registry.update(result);
      } else {
        throw new Error(
          "Vehicle␣with␣registration␣
              number␣" +
            registerAsGuarantee.
                registrationNumber +
            "␣was␣already␣registered␣as␣
                loan␣garantee."
        );
      }
```

```
    });
  });
}

function onCancelGuarantee(cancelGuarantee
    ) {
  var issuer = getCurrentParticipant();
  return getAssetRegistry(namespace + "."
      + vehicleType).then(function (
    registry
  ) {
    return registry.get(cancelGuarantee.
        vin).then(async (vehicle) => {
      var needsUpdate = false;
      if (
        vehicle.registrationNumber !=
            cancelGuarantee.
            registrationNumber &&
        vehicle.make != cancelGuarantee.
            make
      ) {
        throw new Error("Vehicle␣does␣not␣
            match␣the␣registy.");
      }
      if (await isOwner(issuer, vehicle))
          {
        if (vehicle.loan == undefined) {
          throw new Error("This␣Vehicle␣is
              ␣not␣registered␣" + "as␣a␣
              Garantee.");
        }
        needsUpdate = true;
        vehicle.loan.waitingCancelation =
            true;
      }
      if (await isCreditor(issuer, vehicle
          )) {
        needsUpdate = true;
        vehicle.loan = null;
      }
      if (needsUpdate) {
        return registry.update(vehicle);
      } else {
        return null;
      }
    });
  });
}

function onConfirmGuaranteeCancelation(
    confirmGuaranteeCancelation) {
  var issuer = getCurrentParticipant();
  return getAssetRegistry(namespace + "."
      + vehicleType).then(function (
    registry
  ) {
    return registry
      .get(confirmGuaranteeCancelation.vin
          )
      .then(async (vehicle) => {
        if (
          !(await isCreditor(issuer,
              vehicle)) &&
          !(await isRegistryEmployee(
              issuer))
        ) {
          throw new Error("Only␣the␣
              Creditor␣can␣execute␣this␣
              action");
        }

        if (
          vehicle.registrationNumber !=
            confirmGuaranteeCancelation.
                registrationNumber &&
          vehicle.make !=
              confirmGuaranteeCancelation.
              make
        ) {
          throw new Error("Vehicle␣does␣
              not␣match␣the␣registry.");
        }

        if (
          vehicle.loan == undefined ||
          vehicle.loan.waitingCancelation
              == false
        ) {
          throw new Error(
            "No␣Loan␣defined␣or␣the␣
                garantee␣is␣" +
              "not␣in␣cancelation␣process.
                  "
          );
        }
        vehicle.loan = null;
        return registry.update(vehicle);
      });
  });
}

function onRejectGuaranteeCancelation(
    rejectGuaranteeCancelation) {
  var issuer = getCurrentParticipant();
  return getAssetRegistry(namespace + "."
      + vehicleType).then(function (
    registy
  ) {
    return registy.get(
        rejectGuaranteeCancelation.vin).
        then(async (vehicle) => {
      if (
        !(await isCreditor(issuer, vehicle
            )) &&
        !(await isRegistryEmployee(issuer)
            )
      ) {
        throw new Error("Only␣the␣Creditor
            ␣can␣execute␣this␣action");
      }

      if (
        vehicle.registrationNumber !=
          rejectGuaranteeCancelation.
              registrationNumber &&
        vehicle.make !=
            rejectGuaranteeCancelation.
            make
      ) {
        throw new Error("Vehicle␣does␣not␣
            match␣the␣registry.");
      }

      if (
        vehicle.loan == undefined ||
        vehicle.loan.waitingCancelation ==
            false
      ) {
        throw new Error(
          "No␣Loan␣defined␣or␣the␣garantee
              ␣is␣" + "not␣in␣cancelation␣
              process."
        );
      }
      vehicle.loan.waitingCancelation =
          false;
      return registy.update(vehicle);
    });
  });
}

/**
 * Seizure transactions
 */

function onIssuePendingSeizure(
    issuePendingSeizure) {
  return getAssetRegistry(namespace + "."
      + vehicleType).then((registry) => {
    return registry.get(
        issuePendingSeizure.vin).then((
        vehicle) => {
      if (
        vehicle.registrationNumber !=
            issuePendingSeizure.
            registrationNumber &&
        vehicle.make !=
            issuePendingSeizure.make
      ) {
        throw new Error("Vehicle␣does␣not␣
            match␣the␣registry.");
      }
      if (vehicle.seizure != undefined) {
        throw new Error("Vehicle␣already␣
            has␣a␣pending␣Seizure");
      }

      if (
        vehicle.loan != undefined &&
        vehicle.loan.creditor.
            getFullyQualifiedIdentifier()
            !=
          issuePendingSeizure.creditor.
              getFullyQualifiedIdentifier
              () &&
        vehicle.loan.type == "COLLATERAL"
      ) {
        throw new Error("Vehicle␣is␣
            subject␣to␣a␣loan␣as␣
            collateral");
      }
      var factory = getFactory();
      var seizure = factory.newConcept(
          namespace, Seizure);
      seizure.owner = issuePendingSeizure.
          owner;
      seizure.creditor =
          issuePendingSeizure.creditor;
      seizure.totalValue =
          issuePendingSeizure.totalValue;
      seizure.status = STATE_WAITING;

      vehicle.seizure = seizure;
      return registry.update(vehicle);
    });
  });
}

function onIssueSeizure(issueSeizure) {
  return getAssetRegistry(namespace + "."
      + vehicleType).then((registry) => {
    return registry.get(issueSeizure.vin).
        then((vehicle) => {
      if (vehicle.loan != undefined) {
        if (
          vehicle.loan.creditor.
              getFullyQualifiedIdentifier
              () !=
            issueSeizure.creditor.
                getFullyQualifiedIdentifier
                () &&
          vehicle.loan.type == "COLLATERAL
              "
        ) {
          throw new Error("Vehicle␣is␣
              subject␣to␣a␣loan␣as␣
              collateral");
        } else {
          vehicle.loan = undefined;
        }
      }
      var oldOwner;
      var creditor;
      if (
        vehicle.seizure != undefined &&
        vehicle.seizure.status ==
            STATE_WAITING
      ) {
        oldOwner = vehicle.seizure.owner;
        creditor = vehicle.seizure.
            creditor;

        if (
      oldOwner.getFullyQualifiedIdentifier
          () !=
      issueSeizure.owner
      .getFullyQualifiedIdentifier() ||
      creditor.getFullyQualifiedIdentifier
          () !=
        issueSeizure.creditor
        .getFullyQualifiedIdentifier()
        ) {
          throw new Error(
            "A␣Pending␣Seizure␣is␣in␣place
                ␣for␣a␣different␣Owner␣or␣
                Creditor."
          );
        }
        vehicle.seizure = undefined;
      } else {
        oldOwner = issueSeizure.owner;
        creditor = issueSeizure.creditor;
      }

      vehicle.owners.forEach(function (
          ownership) {
        if (
      ownership.owner.
          getFullyQualifiedIdentifier() ==
      oldOwner.getFullyQualifiedIdentifier
          ()
        ) {
          ownership.owner = creditor;
          ownership.state = STATE_VALID;
          ownership.newOwner = undefined;
        }
      });

      return registry.update(vehicle);
    });
  });
}

function onCancelSeizure(cancelSeizure) {
  return getAssetRegistry(namespace + "."
      + vehicleType).then((registry) => {
    return registry.get(cancelSeizure.vin)
        .then((vehicle) => {
      vehicle.seizure = undefined;
      return registry.update(vehicle);
    });
  });
}

/**
 * Change State transactions
 */

function onChangeState(changeState) {
  return getAssetRegistry(namespace + "."
      + vehicleType).then((registry) => {
    return registry.get(changeState.vin).
        then((vehicle) => {
      vehicle.state = changeState.newState
          ;

      return registry.update(vehicle);
    });
  });
}

/**
 * Permissions checking functions
 */

async function isOwner(person, vehicle) {
  var finalResult = false;
  var promiseList = [];
  for (
    var i = 0;
    vehicle.owners != undefined && i <
        vehicle.owners.length;
    i++
  ) {
    var owner = vehicle.owners[i].owner;
    if (
      owner.getFullyQualifiedIdentifier()
          ==
      person.getFullyQualifiedIdentifier()
    ) {
      finalResult = true;
    } else if (owner.getFullyQualifiedType
        () == namespace + "." + Company) {
      var promise = isCompanyOwner(person,
           owner).then((result) => {
        if (result) {
          finalResult = result;
        }
      });
      promiseList.push(promise);
    } else {
      finalResult = false;
    }
  }
  await Promise.all(promiseList);
  return finalResult;
}

async function isLessee(person, vehicle) {
  return (
    vehicle.lease != undefined &&
    (vehicle.lease.lessee.
        getFullyQualifiedIdentifier() ==
      person.getFullyQualifiedIdentifier()
          ||
      (vehicle.lease.lessee.
          getFullyQualifiedType() ==
        namespace + "." + Company &&
        (await isCompanyOwner(person,
            vehicle.lease.lessee))))
  );
}

function isCompanyOwner(person, company) {
  return getParticipantRegistry(namespace
      + "." + Company).then((registry) =>
      {
    return registry.get(company.
        $identifier).then((cmpy) => {
      var result = false;
      cmpy.owners.forEach((owner) => {
        if (
          owner.
              getFullyQualifiedIdentifier
              () ==
          person.
              getFullyQualifiedIdentifier
              ()
        ) {
          result = true;
        }
      });

      return result;
    });
  });
}

async function isRegistryEmployee(person)
    {
  return (
    person.getFullyQualifiedIdentifier()
        == namespace + "." +
        RegistryEmployee
  );
}
async function isCreditor(person, vehicle)
     {
  return (
    vehicle.loan != undefined &&
    ((vehicle.loan.creditor.
        getFullyQualifiedType() ==
      namespace + "." + Person &&
      vehicle.loan.creditor.
          getFullyQualifiedIdentifier() ==
        person.getFullyQualifiedIdentifier
            ()) ||
      (await isCompanyOwner(person,
          vehicle.loan.creditor)))
  );
}

async function isJudicialOfficer(person) {
  return person.getFullyQualifiedType() ==
       namespace + "." + JudicialOfficer;
}
```