Using Software Product Lines to Create Blockchain Products: Application to Supply Chain Traceability

Nicolas Six, Nicolas Herbaut, Roberto Erick Lopez-Herrejon, Camille Salinesi

Using Software Product Lines to Create Blockchain Products: Application to Supply Chain Traceability

Nicolas Six
nicolas.six@univ-paris1.fr
Université Paris 1 Panthéon-Sorbonne
Paris, France

Nicolas Herbaut
nicolas.herbaut@univ-paris1.fr
Université Paris 1 Panthéon-Sorbonne
Paris, France

Roberto Erick Lopez-Herrejon
roberto.lopez@etsmtl.ca
École de Technologie Supérieure
Montreal, Canada

Camille Salinesi
camille.salinesi@univ-paris1.fr
Université Paris 1 Panthéon-Sorbonne
Paris, France

ABSTRACT

In recent years, blockchain has been growing rapidly from a niche technology to a promising solution for many sectors, due to its unique properties that empower the design of innovative applications. Nevertheless, the development of blockchain applications is still a challenge. Due to the technological novelty, only a few developers are familiar with blockchain technologies and smart contracts. Others might face a steep learning curve or difficulties to reuse existing code to build blockchain applications. This study proposes a novel approach to tackle these issues, through software product line engineering. To support the approach, a web platform to configure and generate a blockchain application for on-chain traceability is introduced. First, a feature model has been designed to model core features of the chosen domain, based on the existing literature. Then, a configurator has been implemented to support the feature selection phase. Finally, a generator is able to ingest such configurations to generate on-the-shelf blockchain products. The generalizability of the contribution is validated by reproducing on-chain traceability applications proposed in the literature by using the platform. This work provides the first evidence that the implementation of blockchain applications using software product lines enhances the quality of produced applications and reduces the time to market.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; **Software notations and tools**; *Patterns*; • **Computer systems organization → Peer-to-peer architectures**.

KEYWORDS

blockchain, software product line, code generation

1 INTRODUCTION

While blockchain's popularity growth coupled with its unique capabilities has attracted many companies to start blockchain projects, the road going from their drive to innovate to the materialization into production-ready applications remains challenging. Three issues hinder the adoption of blockchain: organizational, legal, and technological [29]. For the latter, one reason is the difficulties met by software developers along the software engineering process. The design of the application can be tedious due to the novelty of components employed inside, such as smart contracts, or cryptographic wallets. A bad design can lead to higher operation and maintenance costs at best and vulnerabilities and flaws at worst (e.g. *The DAO* attack [25]).

Reusing existing code is one solution to solve this issue (so-called clone-and-own) and is a common practice in the blockchain field [8]. Some of these solutions have even been formalized as design patterns to ease their reuse. For instance, as smart contracts cannot query data from outside the blockchain, developers have to apply a design pattern named *Oracle pattern* [39]. An oracle includes two components: a smart contract capable of emitting an event when new data is required, and an off-chain service listening to these events to inject fresh data when needed.

This reuse of existing code is a first step in addressing the difficulties of implementing a blockchain application, but it could be further systematized with a software product line (SPL) approach. Software product line engineering (SPLE) is based on the reuse of various software artifacts (e.g., requirements, models, code, and tests) designed for this purpose, to create (software) products that have common elements [28]. By leveraging an SPL approach, developers could easily configure and generate blockchain applications by reusing existing knowledge and artifacts. First, by reusing domain requirements to guide the selection of features in a feature model (introduced in Section 4). Then, by reusing design artifacts, such as existing design patterns or models. Finally, by reusing configurable code (in this case, templates) that may also implement the aforementioned design patterns. Instead of using the "clone-and-own" strategy, configurable code can be tailored to fit into the project systematically. Yet, the combination of SPL and blockchain technology is still an unexplored area. This study attempts to highlight the relevance of coupling SPLs and blockchain by addressing the following research questions:

- **RQ1** - Is SPLE applicable to the blockchain field?
- **RQ2** - Do blockchain applications created following a standard software development engineering differs from applications derived from a SPL?

To answer these questions, this contribution proposes the creation from scratch of an SPL for blockchain applications. It results in a web platform that allows the configuration and the generation of a blockchain product. The generation is performed by assembling code templates (e.g., smart contracts), based on the configuration

given by the user. A feature model guides the configuration process, by describing existing features and their constraints with others. This feature model has been devised by extracting features found in studies of a specific domain, that is, blockchain-based traceability. We evaluated the capacity to generalize our approach by reproducing existing blockchain-based traceability applications using exclusively the web platform. Also, the source code of the web platform and the templates is available on Github[1]

The paper is organized as follows: Section 2 introduces background on blockchain technologies, then Section 3 discusses related works on blockchain code generation and variability. Section 4 and 5 introduce the platform, with describing the construction of the feature model and its reuse through the web platform. An evaluation is performed in Section 6, and Section 7 discusses those results along with lessons learned and open research challenges. Finally, Section 8 concludes the paper.

2 BACKGROUND

In this section, some background on blockchain technologies and smart contracts is given. A blockchain is a data structure where each block is linked to the previous one with a cryptographic hash. Each block also contains a list of transactions that represent user-to-blockchain interactions. A network of peers, known as nodes, is in charge of adding new blocks to the blockchain. First-generation blockchains, such as Bitcoin [27], only had the purpose of cryptocurrency exchange between users. Nevertheless, the release of Ethereum [38] in 2015 allowed its usage in a wider range of use cases through Turing-complete smart contracts.

A smart contract (also called DApp) is a computer program that executes predefined actions when certain conditions within the system are met. Blockchain smart contracts can be deployed and interacted with through transactions. The nodes responsible for the inclusion of new transactions into the blockchain are responsible for executing the smart contract with provided parameters. Each smart contract is constituted of two distinct parts: its state, and its logic. By interacting with smart contract functions using transactions, it is possible to alter the state of a smart contract.

The usage of smart contracts combined with blockchain allows the development of applications that differ from conventional software engineering. These applications benefit from blockchain decentralization, as there is no central actor to control the network. Smart contract data are also immutable by nature, as it is theoretically impossible to alter a block after its addition into the network. Finally, full transparency of the application is possible, allowing the traceability of its data and usages. Many usages of blockchain have already been explored in the literature. For instance, Tian et al. propose to use blockchain for transparent and trustable traceability of supply chain [34]. Decentralized finance (DeFi) leverages blockchain to exchange value between users without any intermediary [30]. Blockchain can also be used as a platform for peer-to-peer systems, such as smart grids [26].

3 RELATED WORK

Several works in the literature have been proposed to assist practitioners in designing, generating, and deploying blockchain-based solutions, starting from low-level code generation tools to Model-driven Engineering (MDE) and proposals that take blockchain variability into account.

3.1 Smart Contract Code Generation

The most recent blockchain solution supports general-purpose programming languages, such as Java/Kotlin for Corda [16], or Go/Node.js/Java for Hyperledger Fabric [2]. Yet, the vast majority of the literature presenting blockchain-based solutions still rely on Ethereum and its specific languages (e.g. Solidity) to demonstrate the feasibility of their proposal. For this reason, several papers focus on helping developers write smart contracts with Ethereum.

Wöhrer et al. propose a Contract Modeling Language (CML) to simplify the writing of smart contracts [37]. CML defines contract-specific concepts such as Party, Asset, or Event, and decorators to indicate the usage of blockchain-based design patterns in specific functions. A parser is also proposed for CML-to-Solidity conversion. However, this approach requires developers to become proficient in CML in addition to Solidity, as only learning CML might limit developers in the development of smart contracts. Other approaches in the literature focus on reusing existing models to generate code. For instance, Zupan et al. propose a framework to generate smart contracts based on Petri nets [40]. The generation of code is made through their translation engine, which is able to convert Petri nets into Solidity smart contracts. López-Pintado et al. use Business Process Model and Notation (BPMN) to generate a suite of Solidity smart contracts, that are able to run the corresponding business process on the blockchain with a solution called Caterpilar [22]. Generated smart contracts are used to start business process instances, manage business process activities, and handle the business process workflow. Choudhury et al. use a different model for smart contract generation composed of an ontology with classes linked together, and constraints expressed as a set of rules [9].

3.2 Blockchain and Model-Driven Engineering

Smart contract code generation is useful for use cases where all the processed data happens to be on the blockchain. However, these approaches fall short when dealing with the integration of other domain-specific components into the blockchain solution at different architectural levels. Several authors propose relying on MDE to help grasp the complexity of integrating blockchain-based solutions within information systems.

Lu et al. propose a tool called Lorikeet that extends the BPMN modeling capabilities already proposed in Caterpillar with the support of asset registry management [23]. Both business process modeling and asset registry modeling are used to generate smart contracts making the developers more productive, the operators able to monitor smart contracts execution, and the domain experts capable of understanding how their ideas are represented in the system. De Sousa et al. present MDE4BBIS, a framework to incorporate MDE in the development of Blockchain-based IS [12]. They demonstrate their solution to support cross-organizational business processes. Górski et al. propose new UML stereotypes in a UML profile for distributed ledger deployment and incorporated their solution in a modeling tool to automate the deployment to Corda [14].

[1] https://github.com/harmonica-project/BANCO

3.3 Blockchain and SPL

Finally, a few proposals have been made to use SPLs for blockchain. Kim et al. present a feature model to allow organizations to build their blockchain platform by selecting its features (e.g., smart contract language, consensus algorithm, etc.) [18]. They present a feature model for blockchain platforms allowing the selection of the desired features, without, however, supporting feature binding or code generation. Liaskos et al. introduce a meta-model for the derivation of specialized blockchain network simulators, emphasizing the importance of SPLE and MDE [21]. As we have seen from this section, even if code generation and MDE have been proposed to support the creation of blockchain applications, this paper is the first attempt at building an SPL for blockchain applications.

4 FEATURE MODEL DESIGN

The first step in the SPLE process is the domain analysis [10], where the result is often a feature model. A feature model (FM) is a widely adopted notation to describe allowed variability between products of the same family and feature dependencies [32]. The main advantage of using a FM is the increased ease of reusing existing features, with an accurate mapping that can be shared between stakeholders. In this study, the FM has been created following the standard feature model notation with FeatureIDE, an open-source framework [33]. It is composed of different notation elements. It allows the definition of concrete/abstract features that can be optional or mandatory. It also supports *and-* and *xor-* decomposition of features, to either select multiple subfeatures (but at least one) among a given set linked to a feature, or select only one subfeature in the selection. Finally, FMs include constraints between features, preventing for instance the selection of two conflicting features. The standard FM has been chosen as it satisfies our needs for the construction of an on-chain traceability FM.

4.1 Construction Method

The construction of a FM requires extensive knowledge of its associated domain. In this study, this knowledge has been extracted from 5 different works that propose on-chain traceability solutions, called foundational set, shown in Table 1 [3, 6, 13, 19, 36]. From these papers, the features that were at least present twice (2-of-5) were included in the FM. In some cases, they have been refined manually by adding subfeatures (e.g., adding CRUD methods to manage application participants). This results in a FM, presented in the following subsections. Note that this FM is not meant to be a complete representation of existing on-chain traceability features, but provides its most salient characteristics. A complete analysis through a systematic literature review is left for future work.

The resulting FM is composed of 53 different features, split across three different core features. The first core feature, `SmartContracts`, gather features included in the smart contracts. The selection of the subfeatures of `SmartContracts` represents the configuration of the on-chain part of the application. The second core feature, named `Storage`, regroups the features that address how and where traceability data is stored. Finally, the last core feature, `Frontend`, represents the off-chain part of the application.

4.2 Feature Smart Contracts

The first feature of the model is `SmartContracts` (Figure 1). It represents the on-chain part of the traceability application, composed of a collection of smart contract instances. This part of the FM also involves three different constraints, expressed in Table 2.

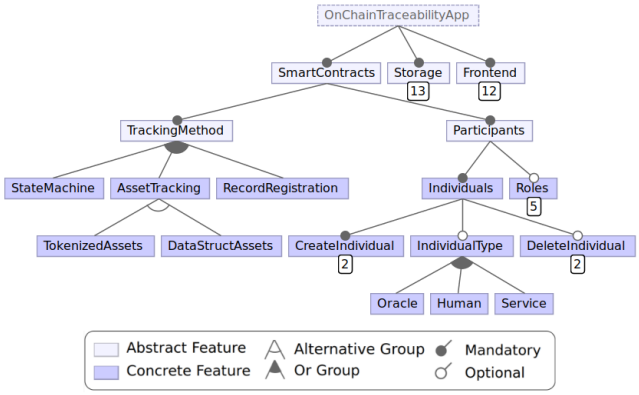

Figure 1: Focused view of the SmartContract FM

This feature is divided into two subfeatures: the management of participants, and the traceability methods used. The feature `Participants` distinguish two important aspects: individuals, that will interact with the traceability smart contracts and are identified by a public address, and roles, that can be assigned to individuals. Using roles is optional in the model, as access control can be done using only public addresses (e.g., only a given set of individuals can add records in a given record collection). However, they can be useful to implement Role-based Access Control (i.e., in a supply chain, identify the suppliers, carriers, and buyers). Besides roles, individuals can be classified through types: they can either be human, service, or oracle (from the *Oracle pattern* [39]).

Three traceability methods can be selected in conjunction or as standalone in the model. The subfeature `StateMachine` allows tracking state changes on-chain. A state machine is defined by a set of state variables and commands, that transform its state [31]. For each transition, it is possible to define a set of individuals and roles that are entitled to trigger the transition between two states. The current implementation behind the subfeature `StateMachine` only allows the creation of basic state machines where each state only has at most one previous and one following state. Nonetheless, this aspect will be improved in future works. `AssetTracking` consists of storing data on real-world assets. Each asset has a set of owners and a set of "entitled" individuals and roles that can modify it. A state machine can be attached to an asset: for instance, a batch can be stored, shipped, or delivered. Assets can either simply be stored as a simple data structure, or as tokens (as proposed in [19]). Storing assets as tokens facilitates their transfer between individuals. For instance, a batch can be sent from the supplier to the carrier. Tokenization is a common blockchain-based design pattern [39], standardized for many blockchains such as the ERC721 standard for Ethereum[2]. Finally, `RecordCollections` allows bulk storage

[2]https://eips.ethereum.org/EIPS/eip-721

Table 1: Blockchain traceability research used to design and test the FM

Ref	Title	Authors	Item	Part of
[3]	Ensuring transparency and traceability of food local products: A blockchain application to a Smart Tourism Region.	Baralla et al.	Food	Foundational set
[6]	Blockchain-based traceability in Agri-Food supply chain management: A practical implementation.	Caro et al.	Food	
[13]	A blockchain implementation prototype for the electronic open source traceability of wood along the whole supply chain.	Figorilli et al.	Wood	
[19]	Blockchain-based application for the traceability of complex assembly structures	Kuhn et al.	Manufactured items	
[36]	Blockchain-based data traceability platform architecture for supply chain management	Wei et al.	Goods	
[15]	Blockchain-based solution for the traceability of spare parts in manufacturing	Hasan et al.	Spare parts	Test set
[7]	Blockchain-based food supply chain traceability: a case study in the dairy sector	Casino et al.	Food	

Table 2: Feature model constraints

Range	Operator	Target
DeleteIndividualByRole	$\Rightarrow$	Roles
IndividualsSetup	$\Longleftrightarrow$	CreateIndividualAtSetup
RolesSetup	$\Longleftrightarrow$	CreateRoleAtSetup
RecordRegistration	$\Longleftrightarrow$	RecordHistory
RecordHistory	$\Longleftrightarrow$	RecordsCollectionSetup
AssetTracking	$\Longleftrightarrow$	AssetsData
AssetsData	$\Longleftrightarrow$	AssetsSetup
StateMachine	$\Longleftrightarrow$	StateMachineData
StateMachineData	$\Longleftrightarrow$	StateMachineSetup

of records in arrays. These records are stored as described in the feature `Storage`. As with others, a collection has a set of "entitled" individuals and roles that can append new records.

4.3 Feature Storage

The second feature of this model is `Storage` (Figure 2), divided in two aspects. For the first aspect, data can be stored in multiple formats. In some applications, it is a suite of timestamped records. These records can be either data on a specific event that occurred in the traceability process or regularly pushed traceability data (e.g., real-time temperature).

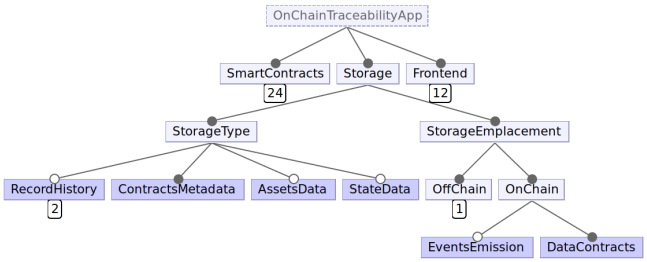

Figure 2: Focused view of the Storage FM.

The FM further refines the subfeature `RecordHistory` in two: `StructuredRecords` and `HashedRecords` (not shown in Figure 2). Where the first one can contain any type of data, the second one is a timestamped hash of a `StructuredRecord`. These records can be used when it is not desirable to store data on-chain for confidentiality reasons or storage limitations. In this case, each structured record is stored off-chain in a database, then hashed and stored on-chain as it. This storage strategy is a common blockchain design pattern named *Off-chain data storage Pattern* [39]. Traceability data can also be stored as objects representing `AssetsData`, or as a set of states and the transition history between them when using a `StateMachine`. These dependencies between storage type and traceability methods imply a set of constraints (Table 2). Indeed, the selection of a specific traceability method should automatically select the related setup form and storage type features. Finally, a mandatory feature named `ContractMetadata` is in charge of storing the address of every smart contract deployed for a traceability process. This feature includes the usage of the *Factory Pattern* [39], as the factory deploys and keeps track of existing contract instances.

Regarding the storage emplacement, data can either be stored on-chain or off-chain. On-chain data is stored in smart contracts following the *Data Contract Pattern*, that separates data storage from logic contracts (e.g., controllers) [39]. Events can also be emitted when something occurs (e.g., storing a new record, firing a transition). Traceability data can also be stored off-chain, in databases. The `Database` feature is mandatory in the FM, as smart contract metadata must at least be stored off-chain to allow retrieving the address of existing contracts. However, traceability data can either be stored off-chain, on-chain, or both.

4.4 Feature Frontend

The last feature is `Frontend` (Figure 3). The frontend application can be used to set up the traceability process through the feature `DeploymentView`.

Individuals, roles, and traceability assets/states/collections are not defined statically in the code but dynamically as parameters passed when instantiating the smart contracts. Thus, the user has to specify these data to set up the traceability process. One feature that is `BlockchainNetwork`, specifies the targeted network: in this model, either the Ethereum testnet (for testing purposes, free to use) or mainnet (in production). Users can then interact with deployed smart contracts through the application to leverage the aforementioned features.

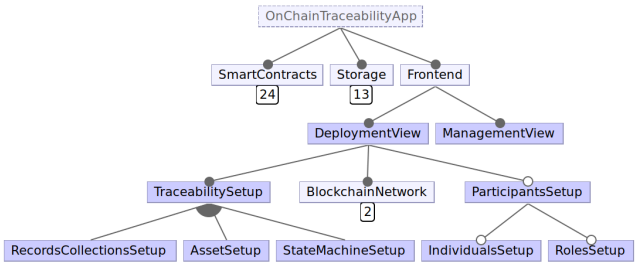

Figure 3: Frontend FM.

5 PLATFORM CONSTRUCTION

The FM guides the possible selection of features by the user when configuring products. However, this task is burdensome when performed manually. In this work, a web platform has been built to ease the configuration of a product. It notably integrates a configurator and a FM visualizer, that was adapted from Kuiter et al. work [20]. This platform also integrates a generator based on templates to output a working product from the user configuration. The following subsections respectively discuss the construction of the configurator and the generator.

5.1 Product Configuration

We implemented two different panels on the web platform to ease the configuration. The first panel displays a tree of features, generated using as input the on-chain traceability FM. Some of the features are already pre-selected, as the FM contains mandatory features. The user can either select the inclusion or the exclusion of a feature by selecting the corresponding box. Each selection will trigger the constraint engine, which will automatically include or exclude features based on the constraints formulated along the FM.

The configuration also has two different states: its validity and its completeness. The first one indicates if a configuration is valid, i.e., if constraints are satisfied. As the configurator prevents selecting two conflictual features, the user cannot make a selection that results in an invalid configuration. The second one indicates if the configuration is complete, i.e., all the features are either selected or deselected. The user can rely on these indicators to know if the configuration step is complete or not.

The second panel shows the FM itself, to visualize the on-chain traceability domain and its available features. The visualizer guides the user during the configuration by changing the color of selected or deselected features respectively in green or red. It allows to quickly notice the impact of selecting one feature on others, and the features that remain to be selected.

5.2 Product Generation

From a valid and complete product configuration, the web platform is capable of generating a working product leveraging a generator based on Template-Based Code Generation (TBCG). TBCG is a technique from the MDE field that consists of generating code based on templates, constituted of static text with embedded dynamic portions that are evaluated by a template engine to output functioning code [17]. Such evaluation also requires providing data to fill the dynamic portions of the text.

In this work, the task of evaluating templates is performed by Mustache, a logic-less web template system[3]. Mustache is capable of evaluating any provided text input that contains a series of tags (i.e., dynamic portions), providing it a suitable JSON object to populate the tags with data. This template system handles features such as optional code blocks, text completion, and loops.

From the configuration made by the user on the web platform, a JSON object is generated containing all of its choices, then ingested by Mustache to process the templates. For the on-chain part, the smart contract templates are written in Solidity [11], a language to implement Ethereum smart contracts. The default Mustache tag has been modified from the default notation (*{{ }}*) to the block comment symbols used in Solidity (/* */), to allow writing Mustache instructions in Solidity comments. This allows for developing and testing smart contract templates without raising any errors caused by the Mustache notation and preserves productivity-enhancing features of Integrated Development Environments such as static code analysis. The approach taken to develop the templates is based on subtractive code generation: all of the features are included in the templates, and Mustache removes or modifies them according to the configuration. For instance, the following code block (Listing 1) will be conditionally rendered in the final product only if the feature *AddRoleDynamically* has been selected by the user.

```
1 /* #AddRoleDynamically */
2 function addRoleToP(address _p, string _rName) public
      {
3   [...]
4 }
5 /* /AddRoleDynamically */
```

Listing 1: Solidity template code sample

Figure 4 describes the chosen architecture for the on-chain part of the application. At first, the user deploys a single factory contract (1). A factory contract, designed following the *Factory pattern* [39], is in charge of creating other contract instances at instantiation (e.g., participant contracts) (2). The factory contract also acts as a contract registry, by storing created contracts' addresses. Once this deployment is completed, the user can interact with controllers (3). Each on-chain feature is implemented as a pair of two contracts: a data contract in charge of holding data collections and getters/setters to manipulate them, and a controller contract to interact with data contracts. These controllers also enforce specific conditions to modify the data (e.g., verifying asset ownership before updating it). The separation between logic and data is a common blockchain design pattern that also increases upgradeability: controllers can be changed without having to migrate the data from one contract to another [39]. Otherwise, this operation would be very expensive in terms of storage and costs and tedious to perform.

The different features defined in the FM can be traced to this architecture. One controller/data smart contract pair is in charge of participants and roles, whereas three controllers/data contract smart pairs are responsible for the different traceability methods defined in the FM. As the user can select one to three different traceability methods, some of these contracts might not be present

[3] https://mustache.github.io/

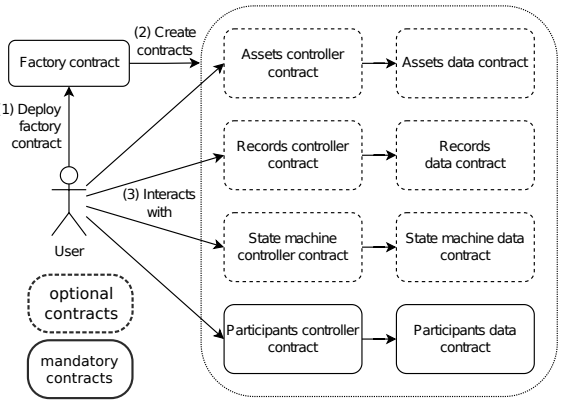

Figure 4: Smart contract architecture

in the final product. However, the participant data/controller smart contract pair will always be present, although the role features might not be depending on the configuration.

For the off-chain part, a web application has been developed, where pages are conditionally included in the final product depending on the configuration. For instance, if the user does not select the `Roles` feature, the web page to configure or allocate roles to users will not be included in the final product.

6 EVALUATION

The main motivations for using SPLs are the reduction of development costs, the reduction of time needed to create an application and an increased code quality [28]. However, the produced artifacts using the SPL still have to satisfy the application requirements. In this section, the requirements satisfaction and the cost of generated products are evaluated by being compared to reference implementations from a set of existing studies.

6.1 Protocol

The protocol used in the evaluation of the contribution is the following. First, a sample of two studies has been chosen, called the Test Set in Table 1. This selection has been performed following two criteria: (1) the source code is available online and (2) the functional requirements of the application proposed in the study can be clearly identified and extracted. Then, for each study, the following steps have been conducted:

(1) Extract main functional requirements[4] formulated by the authors for their on-chain traceability application.
(2) Configure and generate a product using the web platform from these requirements.
(3) Assess the satisfaction of formulated requirements towards the produced blockchain application.
(4) Compare the operating cost of deploying the on-chain part of the product and the implementation proposed by the authors.

During the second step, the configuration of a product is guided by the functional requirements. However, some features are left to be configured at the end, as these features do not have any impact on the satisfaction of the requirements. To arbitrate on these features, the source code of the reference paper implementation has been used to extend these requirements. Finally, if some features remain unselected after the configuration, they are automatically deselected. Indeed, in the Ethereum ecosystem, implementing additional features increases the operational cost of smart contracts. Only one product is generated for each reference paper, however, many more products that also suit the specified requirements could have been generated. Nonetheless, this generated product is the closest from the reference implementation proposed in each reference paper.

Regarding the fourth step, the operating cost will be measured in gas, a unit that represents the cost of performing an atomic operation on an EVM-compatible[5] blockchain. It is computed by summing all the low-level operations performed during the operation (so-called opcodes). As the templates of the SPL have been written using Solidity, this metric is very relevant to assessing and comparing the performance of blockchain applications. However, other metrics might be considered for other technologies. This aspect is discussed in Subsection 7.1.

The evaluation of the proposed SPL will be considered satisfying if the products generated from the web platform sufficiently match the requirements formulated by the authors of reproduced applications, and if the gas cost for the deployment and the execution of the generated smart contracts is satisfactory compared to the reference papers implementations. For the latter, the gas cost of each implementation is shown in Figure 5.

6.2 Spare Part Study Comparison

The first study chosen for the evaluation discusses a blockchain-based traceability system for spare parts purchasing in manufacturing [15]. The main motivation for this study is the lack of reliable tracing and tracking of spare parts and their ownership, especially when they are employed in sensible domains, such as aeronautics. From this study, a set of 13 functional requirements have been identified and classified (Table 3). Then, a configuration has been created based on these requirements, and the corresponding product has been generated and deployed to assess its performance.

6.2.1 Feature Selection. Two traceability features have been selected. The first feature is `AssetTracking`, as a representation of a spare part must be created by an OEM (Original Equipment Manufacturer) for ownership traceability purposes. As there is no need for modeling tokenized assets, the `StructuredAssets` subfeature is used. Then, the second chosen feature is `StateMachine`, as it is required to trace the current state of purchasing new spare parts. Regarding the `Participants` feature, only individuals have been included in the configuration. Indeed, there is no need to create groups of individuals (e.g., roles) in this scenario. The configuration does not include individual types either, as there are no oracle or external services specified.

For storage concerns, the spare parts study does not specify any off-chain storage. However, events are emitted along the process of refilling spare parts. Thus, the `EventsEmission` feature has

[4]For the sake of brevity, we extracted only a subset of functional requirements that involve writing/modifying data.

[5]The EVM (Ethereum Virtual Machine) is used by nodes to execute smart contracts.

Table 3: Spare parts study functional requirements (SR: satisfied in reference paper, SP: satisfied in the generated product).

Category	ID	Requirement	SR	SP
Purchase request	R.1.1	The engineer shall be able to submit a purchase request.	Yes	Yes
	R.1.2	The line manager shall be able to approve a purchase request.	Yes	Yes
	R.1.3	The procurement manager shall be able to approve a purchase request if the requested spare part within the request is missing from the inventory.	Yes	Yes
Purchase quotation	R.1.4	The procurement manager shall be able to submit purchase quotations for a requested spare part.	Yes	Yes
	R.1.5	The engineer shall be able to select a purchase quotation for a requested spare part.	Yes	Yes
	R.1.6	The procurement manager shall be able to confirm the availability of the requested spare part.	Yes	Partially
Purchase order	R.1.7	The engineer shall be able to submit a purchase order for a requested spare part.	Yes	Yes
	R.1.8	The line manager shall be able to approve a purchase order.	Yes	Yes
	R.1.9	The purchase manager shall be able to purchase the spare part specified by the purchase order.	Yes	Yes
	R.1.10	The engineer shall be able to request a spare part from the inventory.	Yes	Yes
	R.1.11	The engineer shall be able to submit a purchase order for a requested spare part.	Yes	Yes
Spare part transfer	R.1.12	An OEM (Original Equipment Manufacturer) shall be able to create a spare part entry.	No	Yes
	R.1.13	Any participant shall be able to transfer the spare part ownership to another.	No	Yes

been included. Also, the `StateData` and the `AssetsData` storage type subfeatures have also been included, due to the specified constraints between features.

6.2.2 Requirements Satisfaction. After the generation of a product based on this configuration, the satisfaction of requirements can then be assessed (Table 3). In total, 12 of the 13 specific requirements are marked as satisfied. Indeed, the generated product is able to support these requirements by leveraging a state machine to track the state of spare parts refilling, and the ownership of spare parts through assets. However, one requirement has been marked as partially filled. The requirement R6 is difficult to satisfy with the current implementation of the product, as it requires establishing a communication system between the OEM and the procurement manager to ask for spare part availability. As we only evaluate the on-chain part of both applications (i.e., smart contracts), R.1.4 and R.1.12 have been marked as satisfied. These requirements demand to store some documents on IPFS (Inter-Planetary File System), a decentralized storage system [4], then store the document reference (so-called tag) in the smart contract. Both the spare-part study implementation and the generated product can do that, however, they do not propose a frontend feature to store a document on IPFS for the moment. Note that requirements R.1.12 and R.1.13 have been marked as unsatisfied in the spare-part study implementation. Indeed, only one hardcoded spare part has been found in the spare-part study implementation code, and no function allows the transfer of a spare part from one participant to another.

6.2.3 Performance Assessment. To evaluate the performance ratio between the smart contract proposed in the spare-part study and the generated product, we designed a test scenario for spare part refilling, from the request to the purchase. This scenario covers the functional requirements specified in Table 3. Figure 5 compiles the differences from 1 to 8 executions.

The process to compute these metrics is the following. At first, the cost of deploying the smart contracts is assessed. This cost is separated from others as usually it is paid only once by the user, at deployment. However, this is not the case for the spare-part study architecture. Then, each function was executed, both in the

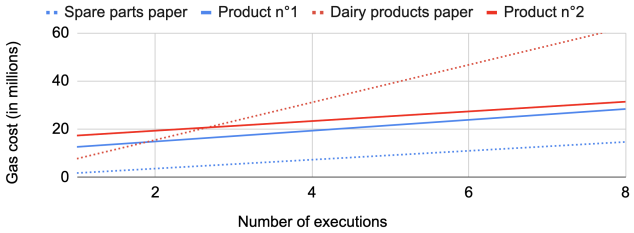

Figure 5: Gas cost of executing several times the reference implementations and generated products.

smart contract proposed in the spare-part study and the generated product. For the latter, the followed scenario involved the creation of an on-chain state machine using the same states as the spare-part study, then transitioning from one state to another providing the same parameters as the first spare part study.

The cost of deploying the generated product is up to 10 431 963 gas, whereas the smart contract proposed in the spare-part study costs 1 513 078 gas to be deployed. However, the generated product allows the creation of a new traceability process using already deployed contracts, where the smart contract proposed in the spare-part study has to be redeployed to be used when starting a new traceability process. Thus, the deployment of smart contracts is not a one-time cost in the spare-part study and has to be paid for each traceability process created. Also, the implementation cost of the generated product includes features for asset management, specified in the spare-part study. However, these features are missing from the spare-part study implementation. Regarding the cost of executing the scenario once the deployment is performed, the spare-part study cumulates a gas cost of 329 840, where the generated product adds up to 2 248 064 gas. Note that two features specified in the requirements are missing from the spare-part study implementation, thus the cost of the generated product for the first 11 requirements can be adjusted to 1 970 268 gas.

Figure 5 displays a tendency of the execution cost of both spare-part study implementation and generated products. To extend these

results, we also computed the cost of executing the scenario several times. As for the spare-part study implementation, the cost is obtained for N execution by summing N time for both the deployment and the function execution cost. In the product generated, the cost is obtained by summing N times the function execution cost and only then adding the deployment cost. Even if these cost models appear simplistic at first sight, they correctly represent the actual cost for EVM-based languages such as Solidity, since EVM imposes a deterministic execution model and fixed code deployment costs. This difference in calculation method is explained as the generated product's smart contracts do not have to be redeployed to create a new process. More rationale on identified cost differences between these two implementations is given in Section 7.

6.3 Dairy Products Study Comparison

For the second chosen study, a blockchain-based food supply chain traceability for dairy products is introduced [7]. As safety is a critical aspect of food supply chains, blockchain and smart contracts can be used to build a secure and trustworthy architecture for food supply chain traceability. In their work, Casino et al. propose such architecture through a concrete use case for the traceability of dairy products. Eleven functional requirements have been identified and classified in this study (Table 4). From these requirements, a configuration has been made on the web platform, and the corresponding product has been generated for comparison.

6.3.1 Feature Selection. This case study both involves the tracking of asset ownership, records, and state changes in a process. The three different tracking methods have been selected to address these requirements: `AssetTracking`, `RecordsHistory`, and `StateMachine`. Also, as there is no need for modeling tokenized assets, the `StructuredAssets` subfeature is used.

For the `Participants` main feature, the paper describes the need to define two roles. The first one is the *Stakeholders* role: stakeholders are involved in the milk transformation process. The second one is the *Administrators* role. Members of this role group are employees from the dairy company and oversee the blockchain traceability application. They are able to perform administration operations, such as adding new stakeholders. The presence of roles in this application justifies the selection of the `Role` feature. According to the dairy products study, it is also possible to create new stakeholders or delete them at any moment. This involves the following features and their subfeatures: `CreateIndividual`, `DeleteIndividual`, and `AddRole`. However, `Individual-Types` have not been added into the configuration, as there is no explicitly mentioned oracles or services.

Regarding the `Storage` feature, the study does not mention the emission of events. As this is an expensive feature in terms of gas cost, `EventEmission` has been excluded from the configuration. The other storage subfeatures, notably the ones related to the traceability data, were automatically selected.

6.3.2 Requirements Satisfaction. Once the configuration step is finished, we assessed the satisfaction of the extracted requirements (Table 4). In total, 8 out of 11 requirements have been marked as satisfied, 2 requirements marked as partially satisfied, and one requirement marked as unsatisfied. In our SPL, an asset can only be weakly attached to a process (here, state machine instances), using the additional data field to reference the instance. Thus, the requirements R.2.8 and R.2.9 have also both been marked as partially satisfied. Regarding the requirement R.2.7, it is not satisfied as it demands a feature to stale an ongoing process: this aspect is not handled by the generated product. Also, as we only evaluate smart contracts in the evaluation, the requirement R.2.5 is satisfied. Indeed, as in the first study, it is possible to attach an IPFS tag to an asset in the generated product, to link it to a document. However, this requires uploading the document beforehand, a feature not handled by the web platform at the moment.

6.3.3 Performance Assessment. The cost of deploying the smart contracts is assessed, then each function was executed, both in the smart contract proposed in the first reference paper and the generated product. Figure 5 shows the resulting costs from 1 to 8 executions. The cost of deploying the generated product is up to 15 400 174 gas, whereas the smart contract proposed in the dairy products study costs 6 748 484 gas to be deployed. As in the first paper, this is not a one-time cost for the dairy products study implementation: smart contracts have to be redeployed for each legal agreement signed between stakeholders and the dairy company in charge of the application. For the functions-related costs, the dairy products study implementation sums up a gas cost of 1 044 928, and the generated product a gas cost of 2 004 322. As in the spare parts study comparison, we also computed the cost of executing the scenario several times. Like the first paper, the implementation cost is added only once to the generated product total gas cost, whereas it is added N times for the dairy products study implementation.

7 DISCUSSION

7.1 Research Questions

In the evaluation section, the relevance of the approach is assessed by replicating on-chain traceability applications found in other works using the web platform. The gas cost of generated products was also assessed by comparing it to the gas cost found for the reference studies implementations. This section discusses these results in the light of formulated research questions (Section 1).

7.1.1 RQ1. To address the first research question, we compared the requirements satisfaction rate of the generated products and the reference papers code. Indeed, if it is possible to replicate most of the existing blockchain-based traceability applications by only using the web platform, the SPL approach is relevant. It has been shown that the web platform was able to produce blockchain applications that satisfy most of the requirements expressed by the studies that were used as reference. Yet, some of the requirements were not fully satisfied. A reason is the genericity of the products that can be generated by the web platform. Indeed, the templates have been designed to be flexible rather than implementing specific domain-oriented features. An illustration of this flexibility is the management of roles: rather than using data structures tailored after the possible roles in a traceability application, a generic data structure named Role is implemented. Also, domain-specific features might be missing from the generated product. This has been faced during the evaluation (Section 6), where some requirements

Table 4: Dairy products study functional requirements (SR: satisfied in reference paper, SP: satisfied in generated product).

Category	ID	Requirement	SR	SP
Product management	R.2.1	An administrator or a stakeholder shall be able to create a product.	Yes	Yes
	R.2.2	An administrator shall be able to change the stakeholder of a product.	Yes	Yes
	R.2.3	A stakeholder shall be able to change the stakeholder of a product if owned.	Yes	Yes
	R.2.4	An administrator or a stakeholder shall be able to push a new record for a given product.	Yes	Yes
	R.2.5	An administrator or a stakeholder shall be able to attach a chemical test to a product.	Yes	Yes
Milk transformation process management	R.2.6	An administrator or a stakeholder shall be able to create a new milk transformation process.	Yes	Yes
	R.2.7	An administrator or a stakeholder shall be able to disable a milk transformation process.	Yes	No
	R.2.8	An administrator shall be able to link a product to a milk transformation process.	Yes	Partially
	R.2.9	A stakeholder shall be able to link a product to a milk transformation process if owned.	Yes	Partially
Stakeholder management	R.2.10	An administrator shall be able to disable a stakeholder.	Yes	Yes
	R.2.11	An administrator shall be able to create a new stakeholder.	Yes	Yes

need to verify a specific condition or execute a defined operation before changing the traceability process state. Nevertheless, the design of the SPL facilitates the integration of new domain-oriented features. In this case, the generated product is solid ground to start implementing more complex features on it.

7.1.2 RQ2. The second research question consists of evaluating if differences between applications generated from an SPL or implemented using a traditional software engineering approach exist. For these applications, the gas cost of deploying and then executing several times a defined scenario has been measured. Then, the divergence of design and code between these applications has been studied to explain the measured gas costs. In the spare parts study, the generated product was more expensive to deploy and execute several times, and for the dairy products study, less expensive after 3 executions. This difference is mainly due to two architectural aspects: the redeployment of smart contracts when willing to relaunch a new traceability process, and the deployment of numerous contracts to facilitate contract upgradeability. Indeed, redeploying a contract requires reallocating a large amount of storage to initialize state variables and store the source code. The products generated by the web platform are designed to avoid this issue: a new state machine (by extension, a traceability process) can be created using a dedicated function. The separation of concerns between data and logic also addresses this issue, as a new controller can be deployed to upgrade some features in generated products rather than redeploying everything. However, this approach has a drawback: the logic required for the separation of concerns and easier upgradeability requires the deployment of larger smart contracts. This results in a more expensive deployment for generated products.

The implementation of the generated products features and reference study implementations also differs. For the latter, hardcoded values were found, notably for the definition of participants and roles. This leads to decreased gas costs, as there is no additional feature for participants dynamic management. On the opposite, the generated products derived from our SPL are flexible and foster maintainability and upgradeability. The flexibility of this approach increases the operating costs of the application. Nevertheless, the high gas costs observed during the evaluation of the SPL might be reduced in future works by implementing features to reduce the code and needed storage size, to the detriment of upgradeability. Also, the deployment of smart contracts and the execution of functions is free on private blockchains networks, such as Proof-of-Authority-based Ethereum networks. In this context, it is not always necessary to optimize the application for gas-cost reduction.

It should also be noted that although the gas cost is an accurate metric to describe Ethereum-based smart contract performance, it is not systematically generalizable to any blockchain technology. Indeed, there is no gas cost at all on other non-EVM-based blockchains, such as Hyperledger Fabric. Other metrics might be considered to assess the performance of blockchain applications in future works using these technologies. For instance, the resource usage of an application (e.g., CPU, RAM, storage size, etc.) could be monitored. The cost of executing the features themselves could also vary depending on the blockchain used. As an example, a feature for data confidentiality requires to implement a function to encrypt data on Ethereum-based blockchains. On Hyperledger Fabric, this is unnecessary as it is possible to restrict the read access of a contract to a defined set of participants, using channels [2].

7.2 Lessons Learned

The main advantage identified during the completion of the study was the time saved compared to manually developing traceability applications. Indeed, after the identification of desired requirements in these works, the configuration and the generation of blockchain applications can be done in minutes. Also, the quality of generated products benefits from the integration of good practices, design patterns, and standards in core assets. However, the main drawback to this approach is the time overhead needed to set up the SPL (feature analysis, FM development, templates development). Compared to a traditional software engineering approach, additional time has been needed to extract the features from the literature, structure them in a feature model, and implement the templates taking in mind the possible combinations between the features as well as the organization between smart contracts (e.g. separating logic and data). Furthermore, this work does not address the challenges of SPL evolution, as the web platform was built over a new feature model. Additional tasks not carried out in this work might be needed to handle the SPL evolution, as illustrated in [1]. This approach might not be tailored for a company willing to implement

a particular blockchain solution, but might suit IT services companies that want to provide a wide range of blockchain products with shared commonalities to their customers.

The templating engine used in this contribution was enough to illustrate the capability of generating blockchain products from configurations. However, a domain engineer may feel limited by the templating engine when implementing many templates for large-scale SPLs. The implementation of the different features within templates might also be tedious, as it must take into account all the possible combinations of features and possible nestings.

Nonetheless, this issue can be mitigated in the blockchain field by different means. First, smart contracts can be designed in a way that the resulting architecture is a set of loosely coupled smart contracts. This approach eases the addition of new features to the SPL. Such architecture is notably introduced by Tonelli et al. [35], as they implement a microservices system with blockchain smart contracts. Consequently, the smart contract architecture proposed in this work was designed with modularity as a main concern. Second, many design patterns, standards, and commonly reused code blocks already exist. As identified by Chen et al., 26% of Ethereum smart contracts code block are from reused sources, notably, ERC20-related contracts [8]. Indeed, ERC20[6] is a standard for the creation of fungible tokens on Ethereum. This existing code can be easily bundled into a feature, reusable in many SPLs.

7.3 Research Challenges

Using SPLs to create blockchain applications raises new research challenges to address. In this paper, the Solidity language has been chosen to develop smart contracts. However, a wider range of languages exists to develop smart contracts for one or other blockchain technologies (e.g., Solidity, Rust, etc.). Future FMs of blockchain products could contain a feature for the selection of a specific smart contract language. This feature could yield SPLs that are able to produce the same application for multiple blockchain technologies. It would allow developers to focus on the application to build rather than the blockchain target behind and its technical specificities. Still, there is an issue with the implementation of such features: the programming model might differ between different blockchains. For instance, Ethereum is account-based, whereas other blockchains such as Bitcoin, rely on a UXTO model [5]. A consequence of these different paradigms could be the impossibility to design some features with specific blockchain technologies. Therefore, more research is needed to assess the generalisability of this approach to other domains or other SPL paradigms.

Also, this paper proposes a domain-oriented FM (on-chain traceability), yet another type of FM could be created around existing blockchain features. The resulting SPL could allow the creation of generic blockchain applications that provide a solid ground for developers to start implementing domain features above. The evolution of SPLs, when core assets (e.g., templates, FMs) evolve over time to address newer requirements or changes in the technology used [24], is also a challenge for blockchain SPLs. This issue is very relevant to blockchain: due to the novelty of the field, many existing standards, patterns, and commonly reused block codes might change in the future, impacting existing features. As mentioned in Subsection 7.2, this is an issue that companies might face when experimenting SPLs for blockchain applications. Future research on blockchain-based SPLs should consider this issue and include mechanisms to handle the evolution of blockchain core assets.

[6]https://ethereum.org/en/developers/docs/standards/tokens/erc-20

8 CONCLUSION

As the development of blockchain applications is still tedious and error-prone, the usage of an SPL can help in the systematic reuse of existing code, good practices, and standards (e.g., Ethereum ERCs) to build robust and efficient applications. This paper denotes the relevance of leveraging SPLs for the design and implementation of blockchain-based applications with an exemplified approach. First, a feature model for on-chain traceability applications is introduced, built by extracting features from 5 different works in this field. From this model, a web platform is proposed to allow the configuration of an on-chain application. The web platform also includes a code generator that reuses this configuration to feed a templating engine that produces a working blockchain application, without any coding. By specifying its desired features, the user is capable of generating an application for on-chain traceability that suits its needs. Also, the produced code is designed to be highly modular, thus easing the addition of new features, either through adding extra features in the feature model or manually. This approach is validated by using the web platform to recreate existing on-chain traceability applications proposed in the literature.

Many research challenges still have to be addressed, such as the management of the SPL evolution considering the rapid pace of blockchain development. Yet, this paper paves the way for blockchain-backed solutions created with the SPL method.

OPEN SCIENCE

Following open science principles, the source code associated to web platform introduced in this paper and the results obtained in the evaluation of the contribution are available on GitHub[7]. A rationale is also given to change the proposed feature model and its related code templates to other domains and purposes.

ACKNOWLEDGEMENTS

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) grant RGPIN-2017-05421, and the Mitacs Globalink Research Award grant IT30564.

REFERENCES

[1] Muhammad Abbas, Robbert Jongeling, Claes Lindskog, Eduard Paul Enoiu, Mehrdad Saadatmand, and Daniel Sundmark. 2020. Product line adoption in industry: an experience report from the railway domain. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*. 1–11.

[2] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*. 1–15.

[3] Gavina Baralla, Andrea Pinna, Roberto Tonelli, Michele Marchesi, and Simona Ibba. 2021. Ensuring transparency and traceability of food local products: A blockchain application to a Smart Tourism Region. *Concurrency and Computation: Practice and Experience* 33, 1 (2021), e5857.

[4] Juan Benet. 2014. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561* (2014).

[7]https://github.com/harmonica-project/BANCO

[5] Lars Brünjes and Murdoch J Gabbay. 2020. UTxO-vs account-based smart contract blockchain programming paradigms. In *International Symposium on Leveraging Applications of Formal Methods*. Springer, 73–88.

[6] Miguel Pincheira Caro, Muhammad Salek Ali, Massimo Vecchio, and Raffaele Giaffreda. 2018. Blockchain-based traceability in Agri-Food supply chain management: A practical implementation. In *2018 IoT Vertical and Topical Summit on Agriculture-Tuscany (IOT Tuscany)*. IEEE, 1–4.

[7] Fran Casino, Venetis Kanakaris, Thomas K Dasaklis, Socrates Moschuris, Spiros Stachtiaris, Maria Pagoni, and Nikolaos P Rachaniotis. 2021. Blockchain-based food supply chain traceability: a case study in the dairy sector. *International Journal of Production Research* 59, 19 (2021), 5758–5770.

[8] Xiangping Chen, Peiyong Liao, Yixin Zhang, Yuan Huang, and Zibin Zheng. 2021. Understanding code reuse in smart contracts. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 470–479.

[9] Olivia Choudhury, Nolan Rudolph, Issa Sylla, Noor Fairoza, and Amar Das. 2018. Auto-generation of smart contracts from domain-specific ontologies and semantic rules. 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, 963–970.

[10] Krzysztof Czarnecki and Eisenecker Ulrich. 2000. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Reading, MA, USA. 864 pages.

[11] Chris Dannen. 2017. *Introducing Ethereum and Solidity*. Vol. 1. Springer.

[12] Victor Amaral de Sousa and Corentin Burnay. 2021. MDE4BBIS: A Framework to Incorporate Model-Driven Engineering in the Development of Blockchain-Based Information Systems. In *2021 Third International Conference on Blockchain Computing and Applications (BCCA)*. IEEE. https://doi.org/10.1109/bcca53669.2021.9657015

[13] Simone Figorilli, Francesca Antonucci, Corrado Costa, Federico Pallottino, Luciano Raso, Marco Castiglione, Edoardo Pinci, Davide Del Vecchio, Giacomo Colle, Andrea Rosario Proto, et al. 2018. A blockchain implementation prototype for the electronic open source traceability of wood along the whole supply chain. *Sensors* 18, 9 (2018), 3133.

[14] Tomasz Gorski and Jakub Bednarski. 2020. Applying Model-Driven Engineering to Distributed Ledger Deployment. *IEEE Access* 8 (2020), 118245–118261. https://doi.org/10.1109/access.2020.3005519

[15] Haya R. Hasan, Khaled Salah, Raja Jayaraman, Raja Wasim Ahmad, Ibrar Yaqoob, and Mohammed Omar. 2020. Blockchain-Based Solution for the Traceability of Spare Parts in Manufacturing. *IEEE Access* 8 (2020), 100308–100322. https://doi.org/10.1109/ACCESS.2020.2998159

[16] Mike Hearn and Richard Gendal Brown. 2016. Corda: A distributed ledger. *Corda Technical White Paper* 2016 (2016).

[17] Sven Jörges. 2013. *Construction and evolution of code generators: A model-driven and service-oriented approach*. Vol. 7747. Springer. 29–31 pages.

[18] Suntae Kim, Sooyong Park, Young Beom Park, Jeong Ah Kim, Young-Hwa Cho, Jae-Young Choi, and Chin-Chol Kim. 2018. A Feature based Content Analysis of Blockchain Platforms. In *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE. https://doi.org/10.1109/icufn.2018.8436843

[19] Marlene Kuhn, Felix Funk, Guanlai Zhang, and Jörg Franke. 2021. Blockchain-based application for the traceability of complex assembly structures. *Journal of Manufacturing Systems* 59 (2021), 617–630.

[20] Elias Kuiter, Jacob Krüger, Sebastian Krieter, Thomas Leich, and Gunter Saake. 2018. Getting rid of clone-and-own: moving to a software product line for temperature monitoring. In *Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1*. 179–189.

[21] Sotirios Liaskos, Tarun Anand, and Nahid Alimohammadi. 2020. Architecting blockchain network simulators: a model-driven perspective. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE. https://doi.org/10.1109/icbc48266.2020.9169413

[22] Orlenys López-Pintado, Luciano García-Bañuelos, Marlon Dumas, Ingo Weber, and Alexander Ponomarev. 2019. Caterpillar: a business process execution engine on the Ethereum blockchain. *Software: Practice and Experience* 49, 7 (2019), 1162–1193. https://doi.org/10.1002/spe.2702

[23] Qinghua Lu, An Binh Tran, Ingo Weber, Hugo O'Connor, Paul Rimba, Xiwei Xu, Mark Staples, Liming Zhu, and Ross Jeffery. 2020. Integrated model-driven engineering of blockchain applications for business processes and asset management. *Software: Practice and Experience* 51, 5 (nov 2020), 1059–1079. https://doi.org/10.1002/spe.2931

[24] Maíra Marques, Jocelyn Simmonds, Pedro O Rossel, and María Cecilia Bastarrica. 2019. Software product line evolution: A systematic literature review. *Information and Software Technology* 105 (2019), 190–208.

[25] Muhammad Izhar Mehar, Charles Louis Shier, Alana Giambattista, Elgar Gong, Gabrielle Fletcher, Ryan Sanayhie, Henry M Kim, and Marek Laskowski. 2019. Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack. *Journal of Cases on Information Technology (JCIT)* 21, 1 (2019), 19–32.

[26] Muhammad Baqer Mollah, Jun Zhao, Dusit Niyato, Kwok-Yan Lam, Xin Zhang, Amer MYM Ghias, Leong Hai Koh, and Lei Yang. 2020. Blockchain for future smart grid: A comprehensive survey. *IEEE Internet of Things Journal* 8, 1 (2020), 18–43.

[27] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008), 21260.

[28] Klaus Pohl, Günter Böckle, and Frank Van Der Linden. 2005. *Software product line engineering: foundations, principles, and techniques*. Vol. 1. Springer.

[29] Kyleen W Prewett, Gregory L Prescott, and Kirk Phillips. 2020. Blockchain adoption is inevitable—Barriers and risks remain. *Journal of Corporate accounting & finance* 31, 2 (2020), 21–28.

[30] Fabian Schär. 2021. Decentralized finance: On blockchain-and smart contract-based financial markets. *FRB of St. Louis Review* (2021).

[31] Fred B Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)* 22, 4 (1990), 299–319.

[32] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. 2007. Generic semantics of feature diagrams. *Computer networks* 51, 2 (2007), 456–479.

[33] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming* 79 (2014), 70–85.

[34] Feng Tian. 2016. An agri-food supply chain traceability system for China based on RFID & blockchain technology. In *2016 13th international conference on service systems and service management (ICSSSM)*. IEEE, 1–6.

[35] Roberto Tonelli, Maria Ilaria Lunesu, Andrea Pinna, Davide Taibi, and Michele Marchesi. 2019. Implementing a microservices system with blockchain smart contracts. In *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, 22–31.

[36] Yihang Wei. 2020. Blockchain-based Data Traceability Platform Architecture for Supply Chain Management. In *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. IEEE, 77–85.

[37] Maximilian Wohrer and Uwe Zdun. 2020. Domain Specific Language for Smart Contract Development. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE. https://doi.org/10.1109/icbc48266.2020.9169399

[38] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

[39] Xiwei Xu, Cesare Pautasso, Liming Zhu, Qinghua Lu, and Ingo Weber. 2018. A pattern collection for blockchain-based applications. In *Proceedings of the 23rd European Conference on Pattern Languages of Programs*. 1–20.

[40] Nejc Zupan, Prabhakaran Kasinathan, Jorge Cuellar, and Markus Sauer. 2020. Secure smart contract generation based on petri nets. In *Blockchain Technology for Industry 4.0*. Springer, 73–98.