University of Camerino

School of Advanced Studies

Doctor of Philosophy in Computer Science and Mathematics - XXXV Cycle

Inter-Organisational Business Processes on Blockchain

Supervisor

Prof. Barbara Re

Co-Supervisor

Dr. Andrea Morichetta

PhD Candidate

Alessandro Marcelletti

Academic Year 2022-2023

ABSTRACT OF THE DISSERTATION

Inter-organisational business processes involve distributed organisations collaborating to reach a common objective. Such kinds of processes can be described by the Business Process Model and Notation (BPMN) choreography diagram, which allows the representation of their interactions from a high-level perspective. However, there is still a lack of support for execution due to the need for a trustworthy infrastructure managing the interactions that occur among distributed parties. Indeed, in a collaborative scenario, trust is subjective and its perception is influenced by technical and organisational aspects. In this context, blockchain is a prominent solution to enable the creation of trustworthy distributed infrastructures.

For this reason, the first research objective of the thesis faces the challenge of trust by proposing a model-driven methodology and a related framework, named ChorChain, relying on blockchain to support the development of choreographies from their modelling to execution. The proposed solution uses a BPMN choreography model as starting specification and automatically translates it into a smart contract deployed on the blockchain. This contract enforces the interactions among the cooperating components as prescribed by the choreography model.

As well as trust, the thesis identifies other relevant research objectives related to the execution of inter-organisational business processes on blockchain. In particular, the thesis focuses also on auditability, flexibility, multiplicity, privacy, and confidentiality. All of them are crucial aspects to consider especially when relying on a blockchain infrastructure. For each objective, the thesis presents a specific contribution, elaborated from the ChorChain framework, that was conceptually and practically extended.The proposed methodology and derived frameworks are assessed through a running example together with real-world and synthetic scenarios.

LIST OF PUBLICATIONS

- Tommaso Cippitelli, Alessandro Marcelletti and Andrea Morichetta. ChorSSI: a BPMN-Based Execution Framework for Self-Sovereign Identity Systems on Blockchain. Submitted to BPM 2023 Blockchain Forum

- Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta, Andrea Polini, Barbara Re, and Francesco Tiezzi. Supporting Multiple Instances in Blockchain-based Model-Driven Engineering. Submitted to ACM Distributed Ledger Technologies: Research and Practice

- Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta, Andrea Polini, Barbara Re, and Francesco Tiezzi. A Flexible Approach to Multi-party Business Process Execution on Blockchain. Future Generation Computer Systems, 147, pages 219-234, 2023

- Francesco Donini, Alessandro Marcelletti, Andrea Morichetta, Andrea Polini. RESTChain: a blockchain-based mediator for REST interactions in Service Choreographies. Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, pages 245–248, 2023

- Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta, Andrea Polini, Barbara Re, and Francesco Tiezzi. Flexible Execution of Multi-Party Business Processes on Blockchain. 5th IEEE/ACM WETSEB Workshop, pages 25-32, 2022

- Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta, Andrea Polini, Barbara Re, and Francesco Tiezzi. A Choreography-Driven Approach for Blockchain-Based IoT Applications. 3rd IEEE BRAIN Workshop, pages 255-260, 2022

- Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta, Andrea Polini, Barbara Re, and Francesco Tiezzi. ChorChain: A Blockchain-Based Framework for Executing and Auditing BPMN Choreographies.

Proceedings of the BPM Demo Track. CEUR Workshop Proceedings, 2022

- Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta, Andrea Polini, Barbara Re, and Francesco Tiezzi. ChorChain: A model-driven framework for choreography-based systems using blockchain. Proceedings of the 1st Italian BPM Forum, CEUR Workshop Proceedings 2952, pages 26-31, 2021

- Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta, Andrea Polini, Barbara Re, and Francesco Tiezzi. Engineering Trustable and Auditable Choreography-Based Systems Using Blockchain. ACM Transactions on Management Information Systems, 13 (3) (2022) 31:1–31:53.

- Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta, Andrea Polini, Barbara Re, Emanuele Scala and Francesco Tiezzi. Model-Driven Engineering for Multi-Parties Business Processes on Multiple Blockchains. Blockchain: Research and Application, Elsevier, 2 (3) (2021) 100018.

- Alessandro Marcelletti and Barbara Re. FabNet: an Automatic Hyperledger Fabric Network Wizard. In Proceedings of 1st BES Workshop CEUR Workshop Proceedings 2749, pages 59-67, 2020

- Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta, Andrea Polini, Barbara Re, and Francesco Tiezzi. Engineering trustable choreography-based systems using blockchain. In 35th ACM/SIGAPP SAC, ACM, 2020, pp. 1470–1479.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

PART I

INTRODUCTION & BACKGROUND

CHAPTER 1 INTRODUCTION

Inter-organisational business processes deal with distributed organisations, controlling different components (e.g., software services, business units, and departments) to perform certain interactions to reach a common final objective [17, 117, 22]. This kind of business involving many distributed parties is becoming crucial since it involves complex and distributed systems difficult to coordinate and execute [67, 22].

The remaining part of the introduction is structured as follows. Section 1.1 introduces the motivation of the thesis while in Section 1.2 the identified research objectives are presented. Finally, Section 1.3 describes the structure of the thesis.

1.1 Motivation

Inter-organisational systems have found in the BPMN standard [89] a prominent modelling language to describe their distributed interactions. In addition to the need for coordination, the involved parties also ask to keep private their internal behaviour [113]. For this reason, BPMN *choreography* diagrams are a valid solution to represent such kind of coordination in a proper way. These diagrams allow to describe system interactions in terms of the exchange of messages from a global perspective, without exposing internal behaviour. In particular, this thesis considers prescriptive choreographies, representing scenarios for future cooperation between stakeholders. However, while the use of choreographies has gathered momentum for modelling and specification purposes, there is still a lack of concrete realisation. Indeed, there is little availability of run-time infrastructures supporting the interactions defined in a choreography specification [81]. The cause is mainly related to the high demand for trust during the run-time execution. In this thesis, the term trust refers to the general notion in business relationships, as defined in [104]:

> "the belief that a party's word or promise is reliable and that a party will fulfil his/her obligations in an exchange relationship".

Trust is a major concern for an organisation that has to engage in a multi-party protocol, possibly with previously unknown partners [120]. However, trust is subjective and somehow different from user to user [82]. Its perception can be positively influenced by all those aspects having an impact during a collaboration. In inter-organisational contexts, it is possible to recognise different trust concerns that influence the trust perceived by organisations [82, 83, 86]. In principle, most of the time trust strictly depends on security, since it is crucial to prevent possible attacks and to provide capabilities to monitor and audit in a reliable and secure way certain events. Another perspective regards access control mechanisms to limit rights only to authorised users. In some situations, restriction to data should also consider privacy and confidentiality requirements. Other security attributes can be found in the integrity of both the application and the produced data.

From a business perspective instead, uncertainty is one of the most influencing aspects and it can be found at different phases of a process. In general, if a business process mostly depends on a single organisation, it can cause uncertainty for the other involved parties due to centralisation. When using a software infrastructure instead, usability and openness should be considered so to make a system easily accessible and transparent on how it works. During the execution, it is important to ensure the correct control flow so that each involved party respects the agreed standard, especially under certain anomalous or unexpected situations. Non-repudiation of resulting traces and events guarantees then that organisations cannot deny the occurrence of certain actions.

All the aforementioned attributes can so impact the execution of a choreography which somehow represents a contract specification that the participant organisations agree to follow. Indeed, this kind of collaboration takes place in a distributed and inter-organisational context, in which it is generally difficult, if not impossible, to have enough guarantees on the fact that the other participants actually abide by rules, or to prove that they did not follow the choreography specification practically. This occurs also because choreography participants have trust concerns about the underlying infrastructure which does not provide enough guarantees on the correct execution of the specified interactions.

In current approaches, this problem is generally delegated to a trusted third party that acts as a central guarantor for the fair behaviour of each participant. With the passing of time and the evolving of scenarios, this solution is not considered suitable anymore since it represents a point of failure bringing also additional costs to the involved organisations [69]. Other solutions as traditional databases are instead recommended when parties mutually trust each other, without considering any participant as malicious. In this case, single or shared writers in a database are assessed valid [124]. When considering instead other innovative technologies, cloud computing can introduce new vulnerabilities and threats caused by collaboration and data exchange over the Internet [82]. For these reasons, this thesis faces the challenge of **trusted execution** of inter-organisational business processes. In this context, the thesis considers also the different trust concerns in a collaborative scenario providing novel solutions for their resolution.

Blockchain is an emerging distributed ledger technology for decentralised and transactional data sharing across a network of untrusted participants [128]. Thanks to its infrastructure, blockchain provides different characteristics enabling the development of new forms of multi-party distributed systems [100, 94, 8]. The adoption of blockchain brings several advantages compared to traditional technologies. Indeed, blockchain contributes to mitigating trust concerns, thanks also to its strong verifiability, whenever multiple mutually mistrusting parties want to collaborate and change the state of a system, without agreeing on a trusted third party [124, 101]. Indeed, the distributed environment allows the creation of systems that do not rely anymore on **central authorities** [28, 76]. Moreover, the consensus algorithm and the cryptographic link between blocks provide the **immutability** of exchanged data and a strong **security** layer. This permits to have a system in which participants can interact in a direct and autonomous way, without the need for a central authority that guarantees the correct execution. This is supported also by the **enforcing** characteristic of the smart contracts that contain the code to execute in an immutable way without the possibility of updating it at run-time. This ensures that each participant is enforced to behave as defined in the initial specification, avoiding the execution of malicious interactions. Finally, in public blockchains, the **transparent** nature of the network allows all the participants to have a clear view of the ongoing system execution and can have tangible proof of the actions performed by their counterparts.

Thanks to all these characteristics, blockchain is advocated for creating novel solutions providing trusted environments.

For this reason, this thesis investigates the use of blockchain facing the challenge of trust during the execution of inter-organisational business processes. Indeed, blockchain can be exploited to allow organisations, represented by choreography participants, to achieve trust without a central authority. This can be done by enabling a distributed and transparent execution between parties in a non-repudiable manner. However, if on one hand, the trusted execution of inter-organisational business processes can leverage the blockchain, the adoption of such technology in those contexts opens new challenges to face such as auditability, flexibility, multiplicity, privacy, and confidentiality.

In addition to the trusted execution, the second challenge is related to the **auditing** of blockchain-based systems. While blockchain provides a transparent environment in which transactions are visible to everyone, the execution of business processes requires more complex mechanisms for auditing purposes. Indeed, single executions are connected to the initial agreement and to specific participants so they must be processed together in order to understand the overall state of the process. Furthermore, the enforcing mechanisms based on smart contracts can guarantee the correct execution step to be performed (for example, that a participant has sent/received a given message and with which content) but can not avoid the sending of incorrect information or human mistakes while using dedicated instruments. The conduction of auditing procedures is indeed useful not only to monitor the execution of the process and resolve potential disputes (e.g., if the payload of a message is different from what is expected) but also to extract some data used for optimising the next executions. For this, novel and concrete instruments are required to support auditing strategies extracting data from the blockchain.

In the third challenge, while the immutable nature of blockchain permits achieving trustworthiness by providing transparent and secure proof of past interactions, at the same time it hinders the **flexibility** of the business process execution [105]. Flexibility, in fact, refers to the possibility of reacting to an unexpected situation at run-time and it is a crucial property due to the high dynamism of the interactions in business scenarios [33]. Indeed, flexibility property could be required to deal with factors exogenous and endogenous to the blockchain-based business process (e.g., new laws, market dynamics, or changes in customers' attitudes). In these situations, a blockchain-based solution does not allow updating the underlying infrastructure, thus requiring a new version of the implementation. This brings additional costs in terms of time and money, and the loss of connection between the data already registered in the blockchain and the new smart contracts. In this situation, a new mechanism has to be defined in order to enable the run-time update of business process logic inside the blockchain.

The fourth challenge is connected to the **multiplicity** aspect of the modelled and implemented scenarios. In the simplest case indeed, the development of trustable systems refers to scenarios where single organisations are willing to collaborate exchanging a single message for each step defined in the choreography. However, this is limiting when considering more complex behaviours, requiring the expression of multiplicity aspects of both participants and their actions. Some examples are the retailing and auction processes, in which roles can be associated with more participants performing multiple actions such as the producers providing different quotations to a retailer or a buyer bidding multiple times. In these situations, the issue of representing multiplicity reflects in the starting model that in those cases results limiting. For this reason, it is necessary to support multiple-instance elements inside the choreography and create associated mechanisms inside the resulting smart contract coordinating multiple interactions.

The last challenge comes from the permissionless nature of public blockchains (e.g., Ethereum [26]). Those have a public and open network without any restriction regarding access to the recorded transactions or the identity of the participants that can join the blockchain. However, while these categories of blockchain suit untrusted environments, where transparency is one of the fundamental requirements to ensure trust, they do not consider **privacy** and **confidentiality** aspects [57]. Indeed, in some situations, parties do not want to expose both the content of exchanged messages (privacy) and their partner (confidentiality), since those provide a potential business advantage with respect to the competitors. For this purpose, alternative permissioned blockchain infrastructures (e.g., Hyperledger Fabric [9]) can be adopted to ensure access control mechanisms and restrict access to the stored data. In this way, the usage of both a permissionless and permissioned blockchain lead to different scenarios about the engineering of the multi-party business processes the thesis targeted. To support this, it is necessary to consider the different technological requirements and peculiarities and integrate them into the proposed model-driven methodology.

1.2 Research Objectives

This thesis aims at providing a novel solution for supporting inter-organisational systems using blockchain technology and relying on a model-driven strategy for their development. To this aim, the thesis also faces the challenges arising from the usage of blockchain for the execution of inter-organisational systems and related to auditability, flexibility, multiplicity, privacy, and confidentiality. More specifically five research objectives were identified and the following were presented:

1. provide a trusted execution environment for inter-organisational business processes;

2. guarantee the auditing of inter-organisational business processes on blockchain;

3. provide run-time flexibility in inter-organisational business processes on blockchain;

4. support multiplicity aspects in inter-organisational business processes on blockchain;

5. support privacy and confidentiality in inter-organisational business processes on blockchain.

1.3 Thesis Structure

The thesis is organised into three parts and eight chapters and they are described in the following.

Background. Part I introduces the motivation of the thesis and the background concepts. In particular, Chapter 2 provides all the fundamentals required for the thesis starting with a description of the different blockchains used, and other relevant technologies passing then to the BPMN notation highlighting the choreography diagram. Finally, the Retail Process running example is shown, providing different contexts in which the defined challenges arise.

Blockchain for inter-organisational business processes. In Part II the thesis describes the main contributions by first providing an insight into the general ChorChain framework focusing then on the other approaches related to the faced challenges:

- Chapter 3 presents the ChorChain framework and tool exploiting a model-driven methodology for the development and execution of trustable inter-organisational business processes. The framework supports the entire choreography life-cycle. In particular, the execution is supported by the use of a public blockchain providing a trusted environment in which users can interact in a distributed manner without relying on a central authority;

- Chapter 4 focuses on the auditing challenge and its implementation. The thesis defines a set of audit strategies and concrete instruments supporting the monitoring of inter-organisational business processes. In particular, the ChorChain framework was extended to support the auditing phase, by exploiting the immutability of information exchanged inside the blockchain;

- Chapter 5 describes the flexibility aspect and the mechanisms for supporting run-time process updates in the FlexChain framework. To this purpose, the business logic is divided from the process state, thus allowing to modify part of the process according to internal or external situations without losing the already exchanged information;

- Chapter 6 introduces the multiplicity concept and the support of multiple-instance elements in the proposed MIChain framework supporting the modelling and development of multiple-instance participant and task elements. Thanks to this, it is possible to represent multiplicity aspects related to the number of participants performing a certain task multiple times;

- Chapter 7 faces the privacy and confidentiality challenge resulting in the MultiChain framework taking advantage of a permissioned blockchain to permit the generation of a different target infrastructure starting from the same model. MultiChain was integrated into ChorChain enabling the choice of a certain blockchain setting at run-time according to different needs of privacy and confidentiality.

Conclusions and Future Works. Finally, Part III concludes the thesis resuming the contributions starting from the usage of blockchain for the execution of inter-organisational systems and the challenges related to auditability, flexibility, multiplicity, privacy, and confidentiality. Then, an overview of future works is provided.

CHAPTER 2 BACKGROUND

This part introduces all the concepts used during the thesis and that will be presented in the next parts. In particular, a first description of blockchain technology and its characteristics is provided. Here both the Ethereum and the Hyperledger Fabric blockchains are analysed, highlighting their peculiarities and usage scenarios.Then a section introduces the Business Process Management (BPM) discipline and the Business Process Modeling and Notation (BPMN) specification with a focus on choreography diagrams. The concluding part of the Chapter reports the Retail Process scenario which will be used as a running example in the remainder of the thesis. In particular, the case study is analysed from different perspectives, emphasising the challenges in using blockchain for a trusted execution. For each challenge, a different context of the retail process is presented, with a focus on those characteristics requiring a novel approach.

2.1 Blockchain

A blockchain is a distributed ledger composed of a linked list (cf. chain) of records called blocks. Each block contains a limited number of transactions in its body, while the header includes, among the others, the number of the current block and the hash of the previous block. This hash cryptographically ensures that previous data (i.e., blocks) is not changed since this would corrupt the related hashes. Transactions can be included in an append-only manner, thus, it is not possible to delete old interactions, but only update them, keeping track of all past events. New blocks are added to the chain at regular intervals of time by the so-called "miners". These are computational nodes, related to the blockchain infrastructure, that are needed to derive the hash of a block. The entire blockchain data structure is instead stored and replicated among some nodes of the network called "full" [130].

The mining process and the use of consensus protocols permit the verification of the genuineness of the transactions included in each block. In addition, the replication of the chain in the network guarantees decentralisation and trustworthiness, without the need for an independent third-party authority.

Blockchain ideas have been initially proposed to support payment systems based on cryptocurrencies. However, in the last years, blockchain technologies have been adopted in many different contexts, especially after the introduction of *smart contracts*. These can be considered special programs that are deployed in the blockchain and executed in transactions. Smart contracts code is deterministic and once deployed it is immutable. The code execution is validated in each node of the blockchain, providing verified proof of the obtained results for the final users [59]. Calls of smart contract functions produce public transactions that are stored in the blockchain, thus making their execution auditable. A contract has a state and it can store both user-defined data and cryptocurrency. Over the years, the growing interest in blockchain has led to the creation of many implementations. From the initial Bitcoin solution, the newest platforms support the creation of complex software and applications [110].

2.1.1 Ethereum

In this context, one of the most prominent and mature blockchain technologies is certainly Ethereum [26]. Each node of the Ethereum network includes an Ethereum Virtual Machine (EVM) supporting the execution of the smart contracts. The operations executed in the EVM, like storage of information or contract instructions, have an associated complexity measure defined in terms of *gas*. The multiplication of this measure for the gas price results in a fee to be sustained by the user requesting the execution of the operation. The gas price is defined by the user and corresponds to the amount to be paid for each unit of gas; a higher value guarantees a faster inclusion of the transaction in the blockchain. The execution fees encourage also mining activities by network participants and, hence, permit keep the overall system working [44]. Indeed, miners are rewarded for each block they mine with a default amount of Ethers (i.e., the Ethereum cryptocurrency) plus the sum of the transaction fees included in the block. To bound the computation for a smart contract and for a block, a *gasLimit* sets the maximum amount of gas used by transactions. However, if from one side this parameter avoids potential denial-of-service attacks and errors, it limits the complexity of blockchain and the overall throughput. To write smart contracts, Ethereum provides the

Solidity[1] programming language. Once the code is generated, it is compiled into a low-level bytecode executed deterministically inside the EVM. After the smart contract is deployed in the blockchain through a dedicated transaction, it becomes available for user interaction. It is worth mentioning that a hex address uniquely identifies any user account in the Ethereum network. An account can also correspond to a smart contract deployed on the network hosting code. This feature is a fundamental aspect during the choreography execution to identify participants. Indeed, in Ethereum, Externally Owned Address (EOA) identifies public accounts storing cryptocurrency that can send transactions inside the blockchain. Those addresses are 42-character hexadecimal corresponding to part of the public key of an Ethereum account. This, combined with a unique private key, form the cryptographic proof of an account. Indeed, having these couple of artefacts, an account signs a transaction with its private key permitting the other participants to verify it with the sender's public key.

Consensus Protocol A fundamental aspect of public permissionless blockchains is the consensus protocol. In 2022, Ethereum switched from Proof-of-Work (PoW) to Proof-of-Stake (PoS) protocol thanks to less energy consumption, better security and the possibility for scaling mechanisms. In PoS, nodes validating blocks (i.e., validators) stake their ETH into a smart contract as collateral in case of malicious or erroneous behaviours. Indeed, validators are responsible for checking the correctness of propagated blocks and sometimes also for creating new ones. This procedure starts with the validator receiving new blocks from other peers and the contained transactions are executed again. In case the block is considered valid, the validator sends a vote across the network in favour of it. In case instead of a new block proposal, a validator is randomly selected and it is responsible for creating and delivering the new block to the other nodes. At this point, a committee of validators is selected and their vote will determine the validity of the block. Differently from PoW, PoS has a fixed block interval time which is divided into slots and epochs respectively of 12 seconds and 32 slots.

Layer 2 and Polygon Ethereum is currently one of the most used implementations for building blockchain-based applications. However, the continuous growth of its usage and popularity has led to an increase in costs associated with it. For this reason, there is a demand for scaling solutions that enable developers to build sustainable (in terms of costs) and performing (in terms of transaction speed and throughput) projects on top of Ethereum. In this context, **Layer 2** approaches are nowadays gaining interest and many implementations are currently available. In general, a Layer 2 approach

[1]`https://solidity.readthedocs.io/`

takes advantage of Ethereum's underlying infrastructure (Layer 1) while extending it taking advantage of its security and decentralisation. Alongside, the concept of **Sidechain** has been developed [16]. Sidechains are indeed blockchains that run independently from Ethereum and are connected to it through a *bridge*. Those chains can have different consensus algorithms and block settings, thus improving the network performance. One of the most stable is **Polygon**, a sidechain-based scaling solution using a Proof-of-Stake (PoS) consensus and enabling the creation of EVM applications. The native currency is named MATIC and the Polygon architecture is divided into three-layer that can be synthesised into:

- node layer producing blocks;
- PoS layer that runs parallel to Ethereum and aggregates the procured blocks into a Merkle tree whose root is periodically committed to the Ethereum chain (checkpoints);
- Ethereum layer handling smart contracts and data from other layers.

Thanks to this architecture and relying on Ethereum, Polygon provides fast, low-cost, secure and high-throughput transactions.

2.1.2 Hyperledger Fabric

Hyperledger Fabric [9, 68, 27, 23] differs from the previously described public blockchain paradigm, due to its architecture and permissioned aim. It is a modular system more versatile for enterprise applications, providing features such as consensus management, private channels, and contracts, full-featured programming languages in smart contracts, and access control policies. It introduces the execute-order-validate architecture, that allows distributed execution of untrusted code in an untrusted environment [10]. Indeed, Fabric executes transactions before reaching a final agreement on their order, then all peers validate transactions in the same order with a deterministic validation. Execute-order-validate paradigm represents the main innovation in Fabric architecture, making Fabric a scalable system for permissioned blockchains supporting flexible trust assumptions.

This kind of system allows providing flexibility, scalability, and privacy in contexts that require these features [68]. The absence of a mining mechanism enables a fast validation and confirmation of transactions. This is achieved thanks to the consensus management system, which allows configuring an arbitrary consensus algorithm, taking into account the requirements of the system intended to implement. Indeed, the arbitrary consensus and the permissioned nature of Fabric guarantee a faster protocol respecting the Ethereum Proof of Stake.

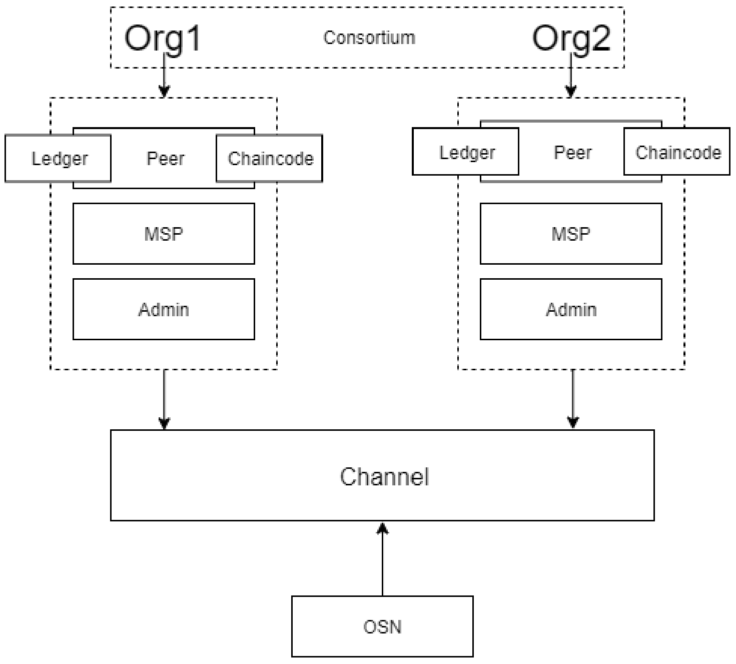


Figure 2.1: Example of 2 organisations network components.

The privacy aspect concerns the confidentiality of transactions and data. This is achieved in Fabric by using its channel architecture and membership service to restrict the distribution of confidential information exclusively to authorised nodes. This is achieved by defining channels inside the network in which only a set of chosen nodes can participate. Nodes and channels are also regulated through policies. Consequently, the channel's ledger is only accessible to its members and the channel's organisations must approve each peer's membership to the channel. Authentication and identity management are guaranteed through a flexible infrastructure based on the Public Key Infrastructure cryptographic scheme and Certificate Authority, which facilitates the joining of a new organisation in the private network. Also, Transport Layer Security cryptographic protocol is used to provide communications security over the network nodes.

The Fabric network is made of different components, reported in Fig. 2.1 and described below.

Peers are the nodes grouped into **organisations**, defined as a trusted domain in which a peer trusts all peers only within its organisation. Peers execute and/or validate transactions, and maintain the ledger. The group of defined organisations participating in a channel is called **consortium**. The ordering service, composed of a set of **Ordering Service Nodes** (OSNs), establishes consensus and atomically broadcasts state updates. The ordering is stateless and decoupled from the peers, it does not take part in the transaction execution and validation process. The **Membership Service Provider** (MSP), maintains the identities of all the nodes (clients, peers, OSNs) inside an organisation. It comprises mechanisms for authenticating transactions, verifying the integrity of transactions, signing and validating endorsements, key management, and registration of nodes. Smart contracts with system **chaincodes** define the transaction logic and the blockchain settings. They are defined in the channel and stored in each organisation's peer. The **ledger** maintains the transactions history, there is one ledger for a channel in which copies are stored in the organisations' peers. In addition, a snapshot of the most recent state is stored in a key-value world state. Finally, the **administrators** have permissions for different operations, like creating and assigning peers, creating network configuration files, and modifying policies through system files.

2.1.3 Blockchain Comparison

The current trend towards introducing different blockchain technologies, with different characteristics, has led to a proliferation of technologies to be acquired and learned. For our purpose, it is possible to distinguish among the following characterisations.

- **Permissionless vs Permissioned**: a permissionless blockchain is an open network where participants can join, and leave the network without the need for any authorisation. A permissioned blockchain runs a ledger among a set of previously identified and authorised peers.

- **Auditability vs Confidentiality**: an auditable blockchain has an immutable and transparent nature, and it naively allows independent auditing over the stored data. On the contrary, a permissioned blockchain introduces confidentiality so that stored data are not visible to anyone. Moreover, it restricts the distribution of information only to authorised nodes.

- **High decentralisation vs Performance and scalability**: the usage of strong consensus algorithms allows to trust nodes previously unknown or not trusted, in a decentralised context. On the contrary, the introduction of access control mechanisms leads to a trusted network with higher scalability and transaction throughput

- **Anonymity vs Identity**: a blockchain technology could permit anyone to join the network without putting in place any access control mechanism. Trust over the stored data will be in any case guaranteed by the consensus algorithm. On the other hand, access to a blockchain can be restricted to authorised users introducing specific mechanisms.

Ethereum implementation	**Hyperledger Fabric implementation**
Permissionless	Permissioned
Auditability	Confidentiality
High Decentralisation	Performance and Scalability
Anonymity	Identity

Table 2.1: Ethereum and Fabric comparison based on blockchain characterisations.

Consequently, it will be possible to associate identities with participants, and cryptographic credentials can be issued to new members. All communications can also be made use of authentication mechanisms.

In particular, this thesis considers two specific blockchain implementations: Ethereum and Hyperledger Fabric. The two technologies present rather orthogonal characteristics and have been conceived for different application domains. Table 2.1 compares Ethereum and Hyperledger Fabric with respect to the list of properties presented above, as confirmed in [93, 118, 102, 127, 119]. The permissionless characteristic of blockchain, like Ethereum, guarantees trusted and verifiable communication between untrusted and unknown organisations. At the same time, Ethereum lacks privacy, performance, and access controls. Permissioned blockchains, like Hyperledger Fabric, cover these aspects, leaving more freedom to the users in the network's organisation. In particular, this suits well when partial trust relationships between parties can be assumed.

In most cases, the right selection of the underlining blockchain technology for a given choreography scenario does not depend only on the system's behaviour. It is also influenced by the context in which the system will have to operate. This means that the same choreography model could be deployed in different situations within different blockchain technologies, depending on the level of trust required by the considered scenarios.

2.2 Business Process Management

Business Process Management is the discipline overseeing the work conducted by organisations to ensure consistent outcomes and to take advantage of improvement opportunities [46]. In particular, BPM aims to manage the entire chain of events, activities, and decisions connected to an organisation. These chains are called *business processes* and BPM includes concepts, methods, and techniques to support their design, administration, configuration, enactment, and analysis [121]. Those processes can be used to represent many kinds of interactions as in the case of collaborative ones. Those are called in general multi-party business processes and usually take place when many parties are involved in collaborations where there is a global awareness of the relevant interactions [25].

Business process activities can be enacted automatically by information systems, without any human involvement. Hence, inter-organisational business processes are an important concept in facilitating collaboration between companies and information systems. More and more business processes also play an important role in the design and realisation of flexible information systems. These information systems are essential for providing the technical basis useful for a quick implementation of new functionalities that realise new products or services.

BPM is characterised by a set of steps that occur cyclically in order to adapt and improve the model. Hence, BPM involves a continuous cycle, comprising the following phases: *modelling*, *analysis*, *execution*, and *monitoring*.

2.2.1 BPMN Choreography Diagram

Nowadays, a prominent modelling language to describe collaborative distributed systems is the BPMN standard [89]. BPMN is a graphical notation to represent the graphical layout of business processes [32, 123], it has been standardised by the Object Management Group (OMG) and it is actually widely accepted both in industry and academia. BPMN permits the representation of business processes with different levels of detail through the use of diagrams. Each diagram consists of a set of modelling elements expressing different behaviours.

Although it has been initially introduced to define and document business processes, the use of BPMN models has successively gathered momentum as a starting point for model-driven engineering of distributed systems [91, 5]. For such a purpose, BPMN provides a set of flowchart-based notations, permitting the representation of different distributed systems perspectives, starting from the internal behaviour of the composing sub-systems, till their interaction. In inter-organisational cooperations, in addition to the need for coordination in a distributed setting, the involved parties also ask to keep private their internal behaviour [113]. For this reason, organisations have found in BPMN *choreography* diagrams a valid solution to represent such kind of coordination in a proper way [36, 6]. These diagrams allow to describe system interactions in terms of the exchange of messages from a global perspective, without exposing internal behaviour. In a distributed environment, organisations wishing to collaborate can refer to specific choreographies that describe in detail how the different parties should interact to achieve common objectives. The integration of processes in this way leads to more peer-to-peer collaboration, shifting responsibility for each execution step of the collaborative process to the individual nodes. Consequently, in a choreography approach, each participant is responsible for partial orchestration, based on its individual rules without a central coordinator, and the final behaviour is specified as a family of permitted message exchange sequences. Figure 2.2 depicts the most used modelling elements that can be included in a BPMN choreography diagram. On the left, elements are used to define the control flow, while on the right, the elements are used to represent communication tasks.

- **Events** are used to represent something that can happen. An event can be a *Start Event* representing the point from which the choreography starts, or an *End Event* representing the choreography termination. Events are drawn as circles.

- **Sequence Flows** are edges used to connect events, gateways and tasks, permitting to specify the choreography execution flow.

- **Gateways** are used to drive the flow of a choreography. They can act either as join nodes (merging incoming sequence edges) or split nodes (forking the flow into multiple outgoing edges). Different types of gateways are available. An *exclusive gateway (XOR)* permits to represent choices. In particular, a XOR-split gateway is used after a decision to fork the flow into branches. When executed, it activates exactly one outgoing edge. A XOR-join gateway acts as a pass-through, meaning that it is activated each time the gateway is reached. A XOR gateway is drawn with a diamond marked with the symbol "×" A *parallel gateway (AND)* enables parallel execution flows. An AND-split gateway is used to model the parallel execution of two or more branches, as all outgoing sequence edges are activated simultaneously. An AND-join gateway synchronises the execution of two or more parallel branches, as it waits for all incoming sequence edges to complete before triggering the outgoing flow. An AND gateway is drawn with a diamond marked with the symbol "+". An *event-based gateway* is similar to the XOR-split gateway, but its outgoing branches activation depends on taking place of catching messages. Basically, such messages are in a race condition, where the first event that is triggered wins and disables the other ones. An event-based gateway is drawn with a diamond marked with the symbol of a double-rounded pentagon.

- **Tasks** are used to specify the message exchange between two participants. They are drawn as rectangles divided into three bands: the central one includes the name of the task, while the other two refer to the involved participants (the one in white is the initiator, while the grey one is the recipient). Messages can be sent either by one participant (One-Way tasks) or by both participants (Two-Way tasks).

When creating tasks and participants it is also possible to specify the **Multiplicity** aspect by including parallel Multi-instance markers. The resulting behaviour indicates that a participant can send/receive many messages or that a single message can be sent/received from/to many participants.

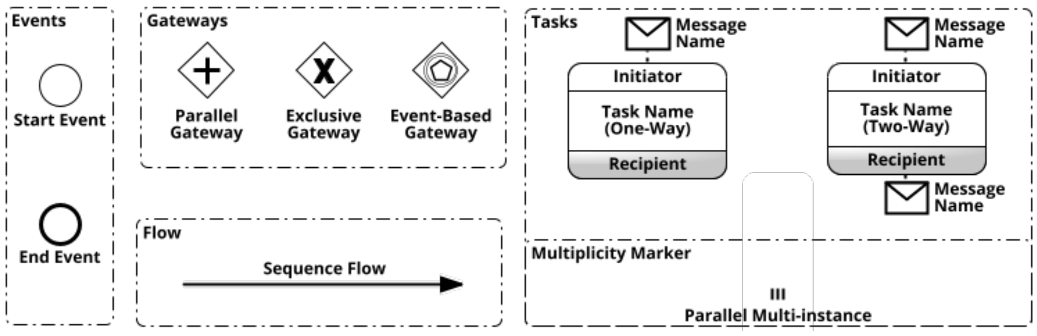


Figure 2.2: BPMN choreography core elements.

2.2.2 Running Example

To clarify the motivations and research challenges faced in this thesis, this part reports a running example consisting of a multi-party business process that allows a customer to buy goods from a retailer. The proposed example refers to warehousing management, and in particular, the considered policy aims at reducing warehousing costs. The retailer does not keep in the warehouse a high volume of goods and generally starts the acquisition process as soon as it receives a specific request. It is also possible that the customer's request indicates a specific producer to involve in the provisioning. This example highlights an interaction scenario in which the requirements of trust and privacy change according to the contexts in which the system operates.

Running example description The choreography reported in Fig. 2.3 represents the communications that should take place among the participants for the scenario described above. The model starts with the request by the customer for a quotation of goods. In case the goods are available, the customer proceeds with the payment, and the retailer commits to deliver the goods. In the other case, the retailer has to buy goods from the producer that the customer could have indicated, which then proposes a quotation. This quotation will be followed by the payment and the shipment of goods to the retailer that can close the customer's order with the final shipment.

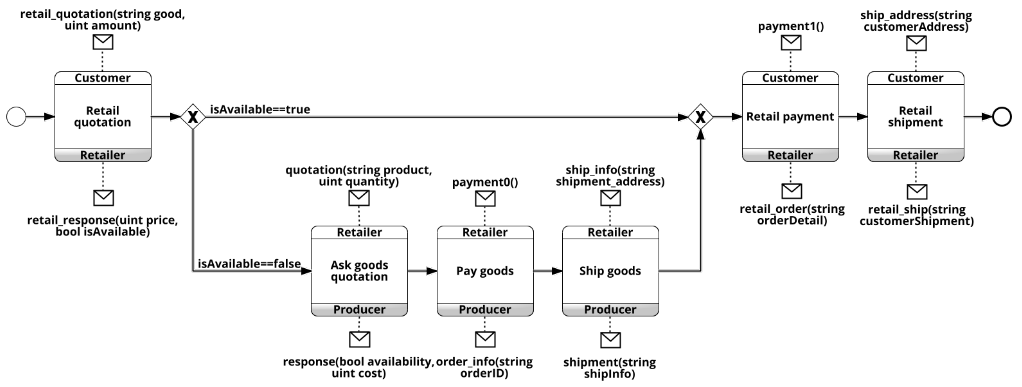

retail_quotation(string good, uint amount)
Customer
Retail quotation
Retailer
retail_response(uint price, bool isAvailable)
isAvailable==true
isAvailable==false
quotation(string product, uint quantity)
Retailer
Ask goods quotation
Producer
response(bool availability, uint cost)
payment0()
Retailer
Pay goods
Producer
order_info(string orderID)
ship_info(string shipment_address)
Retailer
Ship goods
Producer
shipment(string shipInfo)
payment1()
Customer
Retail payment
Retailer
retail_order(string orderDetail)
ship_address(string customerAddress)
Customer
Retail shipment
Retailer
retail_ship(string customerShipment)

Figure 2.3: Retail process choreography diagram.

Trust requirement In the first considered business context, the described interactions pertain to a “fair trade” business model in which participants are not known a priori and each instantiation of the model can involve different customers and organisations. In this business model, several needs emerge. As the first aspect, Customers want to make sure about fair remuneration to the Producer, requiring a mechanism for trusting this kind of market. The second aspect is instead related to the certification of goods and their effective origin. In this case, a need for assessing that a certain good comes from a valid Producer arises. For these reasons, blockchain can guarantee the sharing of certified information about the entire production chain, making it possible for the end customer to verify the source of a certain product. Furthermore, the use of blockchain does not require additional authorities or central parties to ensure the fairness of the process, avoiding the price increase due to their involvement.

Auditing requirement In the case of a fair market model, it is of interest for all the participants to keep a certain level of transparency over the transactions they perform. In particular, a Retailer operating in such a context should be interested in making publicly accessible the origin, and the price of the sold goods. In this way, current and future Customers can see exactly who the Producer is and if the history of prices is somehow fair, and related to a reasonable treatment of the producer. In particular, the public nature of the information stored in the blockchain permits the creation trustable auditing mechanism for the Retailer’s specific goods, and the prices applied over time for the products. This transparency will increment the retailer’s reliability from the customer’s perspective in the specific business model context.

Flexibility requirement Considering the flexibility aspect in the proposed scenario, a possible case is shown in Figure 2.4. Here, the Retailer proposes an internal update of the process in order to optimise the overall costs and execution steps. The new agreement consists of a different type of shipment that involves the Producer. Indeed, after the change in the choreography, this time the Producer directly ships the goods to the Customer without requiring the intermediary Retailer. This affects the entire process starting from the initial request of the Customer that, after the response of the Retailer, pays for the goods and passes the information for the shipment. After this, if the goods are available, the Retailer ships them and the process ends. In the other case instead, the Retailer concludes the business with the Producer that will ship the acquired goods to the Customer thus, ending the process.

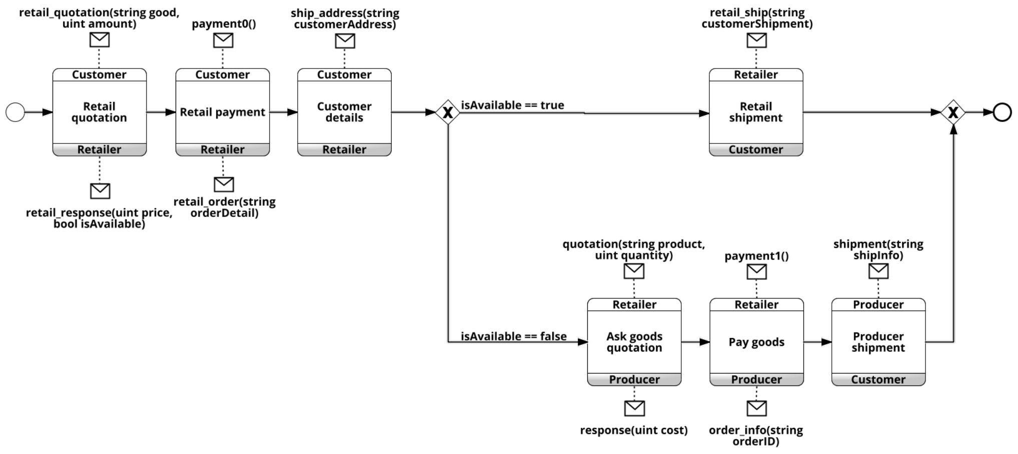


Figure 2.4: Updated Retail process (flexibility).

Multiplicity requirement In the previously mentioned process, each role defined in the choreography corresponds to one single organisation at run time. However, it is common to have a single market in which several types of goods are available, having for example a Retailer that communicates with more Producers at once asking for different quotations and selecting the most proper one. Also, in a standard setting, a Retailer can serve more Customers asking for one or more goods to purchase. This kind of multiplicity becomes so a fundamental aspect to consider when modelling and implementing a system. Indeed, when this factor is not considered, for every new product or Producer, a new instantiation of the process is required thus increasing the related costs inside the blockchain. For this reason, expressing multiplicity both in the model and in the resulting implementation is a relevant challenge, especially when relying on a blockchain. To highlight this kind of behaviour, Figure 2.5 depicts the Retail process with multi-instance tasks and participants.

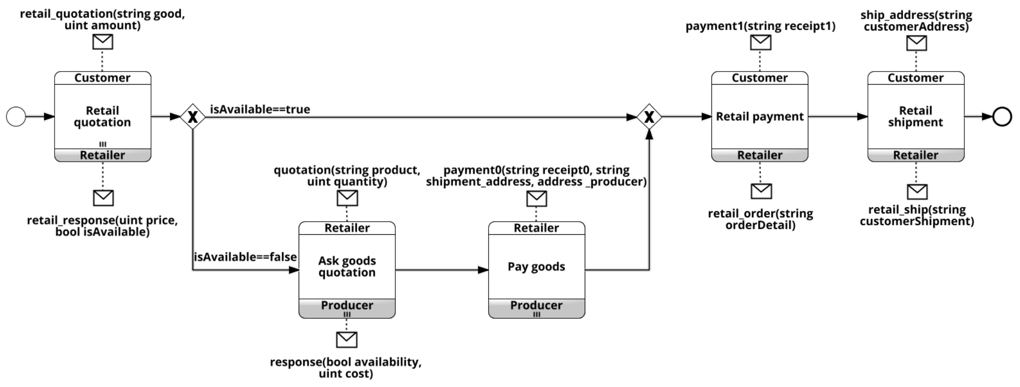

retail_quotation(string good, uint amount)
Customer
Retail quotation
Retailer
retail_response(uint price, bool isAvailable)
isAvailable==true
isAvailable==false
quotation(string product, uint quantity)
Retailer
Ask goods quotation
Producer
response(bool availability, uint cost)
payment0(string receipt0, string shipment_address, address _producer)
Retailer
Pay goods
Producer
payment1(string receipt1)
Customer
Retail payment
Retailer
retail_order(string orderDetail)
ship_address(string customerAddress)
Customer
Retail shipment
Retailer
retail_ship(string customerShipment)

Figure 2.5: Multi-instance Retail process (multiplicity).

Privacy and Confidentiality requirement The last scenario takes place when the parties operate in a close environment where, for business purposes, the participants are more interested in keeping private most of the information related to the products. In particular, considering the participants actively involved in "traditional" business operations, the retailer and the producers are generally interested in keeping confidential the quotations they agree on about a specific selling. While a producer may want to keep secret the quotation applied to a specific retailer, this may want to make a private offer to a particular customer without showing the price for the same goods. The retailing scenario provides an example of a multi-party business process that, when different operative domains are considered, does not differ much from the operative aspects, and the interactions put in place to reach specific objectives. Instead, the operative domains result in rather diverging needs when modalities of such interactions, and capabilities of successive analysis, are considered.

PART II

BLOCKCHAIN FOR INTER-ORGANISATIONAL BUSINESS PROCESSES

CHAPTER 3

CHORCHAIN: TRUSTED EXECUTION OF INTER-ORGANISATIONAL BUSINESS PROCESSES

In this chapter, the thesis introduces the ChorChain framework and tool supporting the trustable execution of inter-organisational business processes. ChorChain relies on a model-driven approach and it poses the basis of the contributions presented in this thesis. In particular, ChorChain supports various phases of the choreography, from the modelling to its instantiation and execution exploiting blockchain for a trusted environment. Thanks to the model-driven approach, the proposed solution automatically generates *smart contracts*, making the process entirely transparent to the final user. In particular, ChorChain targets all that kinds of users that intend to collaborate without having to deal with technicalities or that want to exploit the model-driven development. For instance, those users can be referred to the organisations' developers or to business parties involved in a choreography. The use of BPMN as modelling notation permits to have a visual representation of system implementation and makes it usable also by non-technical people [55]. Furthermore, the enforcement provided by blockchain and smart contracts, allows choreography participants to perform only those actions that are enabled at the current step of the choreography. It guarantees that the execution of the choreography complies with the given specification. The rest of the chapter first introduces the peculiarities of the ChorChain framework, consisting in its phases, the BPMN meta-model extension and the conceptual translation of BPMN to smart contract. Those aspects are then described in practice with a focus on the implemented tool. Then, a set of experiments on the running example and other use cases are reported. Finally, a comparison with already existing approaches is provided.

3.1 ChorChain Conceptual Framework

The ChorChain framework aims at providing a trustable execution environment addressing various trust concerns. To this purpose, ChorChain implements most of the patterns used in a collaborative environment to reduce uncertainty and increase trust [86]. In particular, the proposed framework supports data integrity provided by storing data in the smart contracts and non-repudiation of actions performed by organisations, using the blockchain logs to identify which action is performed by whom. Also, the integrity of the system is ensured by storing the entire process in the blockchain, and by enforcing the prescribed execution. Finally, the availability of activities and resources is provided to each organisation by interacting through the smart contracts and by monitoring them. Considering other technical factors, the usage of a model-driven technique and of a blockchain environment, makes ChorChain support security aspects and other attributes such as the usability and transparency of the solution [83].

3.1.1 Framework phases

The ChorChain framework supports different phases of the choreography development. Those are reported in Figure 3.1 and they consist of the initial modelling of a choreography that becomes available to be instantiated. The resulting instance is then deployed on the blockchain and the involved parties can finally interact with it.

The first phase of the proposed model-driven methodology is the **system modelling**, consisting of the creation of a choreography diagram. The main motivation for using a specification to design how the different system components should interact to reach a common goal concerns the possibility to abstract from implementation details. Indeed, in the blockchain-based solution, the use of a high-level specification permits alleviating the burden on the shoulders of the developer, who can avoid dealing with smart contract technicalities directly.

The resulting **model is published in a choreography repository** making it publicly available and enabling the **searching** phase. Open access to models can foster cooperation among unrelated parties. In the ChorChain approach, a choreography model is conceived as a blueprint, which

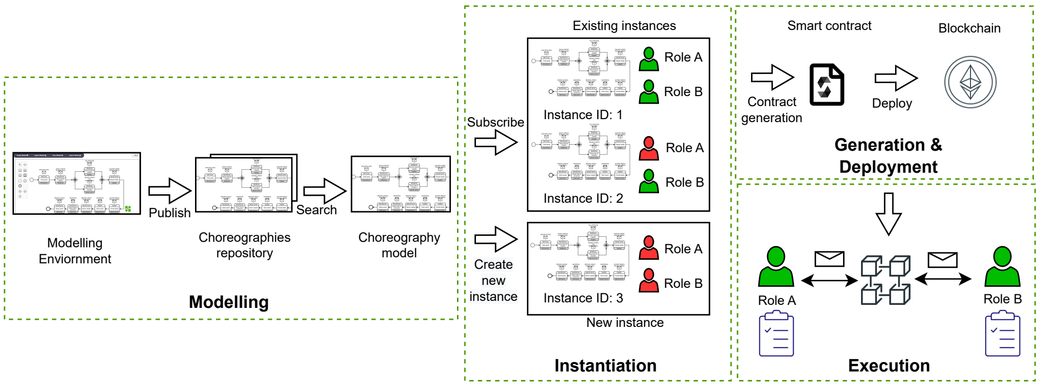


Figure 3.1: Choreography life-cycle supported by ChorChain.

can be **instantiated** to activate multiple cooperations with the same structure, but possibly involving different participants. To increase the reusability of choreography models, the instantiator can indicate which roles in the newly created instance are mandatory and which are optional. In this case, the initiator has to accurately select the mandatory/optional roles to ensure the proper completion of the instance execution. When a choreography instance is created, in order to be executed it has to be **subscribed by** the participants that aim at playing a given role in that specific cooperation. Before passing to the next phase, it is necessary that each mandatory role has been subscribed to by a participant.

Once all the required roles are filled, the smart contract corresponding to the choreography instance is **generated** and then automatically **deployed** on the blockchain. At that time, the **execution** phase starts, and the participants can cooperate following the message protocol established by the choreography specification, and implemented in the smart contract. The smart contract permits to ensure that the participant interactions are compliant with the choreography specification since only the enabled actions are allowed to the partner in charge for their execution.

3.1.2 BPMN meta-model extension

Since a choreography, due to its level of abstraction, does not include all the information needed for its execution, ChorChain asks the modeller additional data related to the specification of *(i)* messages and *(ii)* guards. Such a modelling activity is conceptually supported by a small extension of the BPMN standard meta-model, so to permit the inclusion of blockchain-specific information within the model as represented in Figure 3.2. The figure reports in grey (upper part) the concepts coming from the BPMN standard, while the bottom part in orange reports the ChorChain ones. To derive a blockchain-based infrastructure, supporting a choreography-based

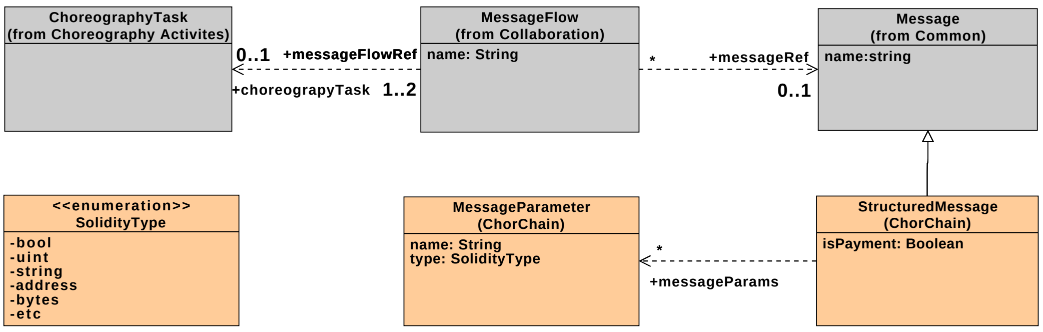


Figure 3.2: ChorChain BPMN meta-model extension.

system at run-time, it is necessary to specify a list of parameters for the exchanged messages. Therefore, a stereotyped form of the message is defined, named *StructuredMessage*, to which a list of *MessagePameter* objects is associated. Each element in this list has a name and a type taken from an enumeration including all the Solidity types. Thus, during the modelling phase, the modeller can annotate the message(s) of each choreography task with the parameters needed to perform the underlying function call in the generated smart contract. More in general, the specification of a task requires a set of information related to: the participant names, the name of the exchanged message, its parameters, and in case the message contains a payment this has to be clearly specified. The result of this procedure is the addition of a list of parameters after the message name in the form of: $msgName(paramType_1\ paramName_1, \ldots, paramType_n\ paramName_n)$.

The blockchain naively supports financial transactions among the interacting partners for exchanging amounts of cryptocurrency. Thus, ChorChain gives the possibility to include messages in a choreography producing financial transactions. To do this, the name assigned to the message is of the form *payment n ()*. The lack of any parameter is justified by the fact that the only information required by the payment function refers to the involved participants, which can be directly and automatically retrieved from the task description. The amount to be paid is indicated by the sender during the choreography execution, exploiting the dedicated page. The resulting transaction transfers the amount in Ether from the sender to the receiver wallet. The second aspect to consider when deriving models that can be translated into executable smart contract refers to the specification of the guards in the sequence flows outgoing from an exclusive gateway, used to determine the path to be triggered. In ChorChain, a guard is an expression in Solidity format, written using the message parameters combined with the standard comparison operators for boolean, numeric and string values.

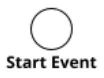
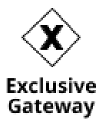
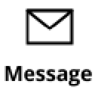
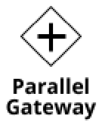
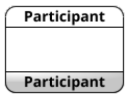
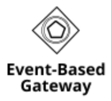
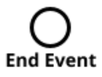
3.1.3 Translation approach: BPMN to Solidity

The smart contract generation is an automatic phase where the choreography instance is translated into Solidity code starting from a BPMN choreography. To this purpose, a novel approach to the automatic generation of smart contracts is proposed. In particular, this translation maps each BPMN element of a choreography to a Solidity construct as reported in Table 3.1 where the different translation cases are shown. In general, each generated smart contract shares a similar structure and an initial header defining general information such as state variables and roles definition. After this part, each choreography element generates a specific function that varies depending on the model. To enforce the correct execution flow, inside the smart contract, each element is associated with a state that is originally blocked and can be activated only by its predecessor depending on its behaviour. The states are as follows: `DISABLED` is used when the element has never been called and is waiting for being enabled, `ENABLED` when is waiting for being executed, and `DONE` once it has completed the execution. Indeed, inside each generated function, there is an initial check on the current state of the element that has to be active. Then, before terminating the execution, the next connected element is enabled.

The choreography elements appearing in the contract can be divided into two main categories, the first one contains *messages*, representing the interactions between participants. During the generation, the concept of choreography *task* is concealed in favour of the connected messages (case 2). In particular, in a one-way choreography task, only the single message exchanged by the two participants is translated. Similarly, in a two-way task, both exchanged messages are translated. Thus, the translation generates a public function for each message of a (one-way or two-way) task containing the inputs defined inside the message name. In case the function is marked as payment, the corresponding code is generated. It is worth noticing that the participants represented in a task (case 3), are associated inside the smart contract to Ethereum addresses used for access control inside a special Solidity function called *modifier*. Another important aspect is related to the user inputs which are parsed starting from the message name. Indeed, each parameter is translated to a state variable inside the smart contract which keeps the memory of the exchanged data.

The second category is instead related to *control flow* elements which determine and enforce the execution flow. In this case, a private function for each of them is generated, hence it cannot be directly called by the choreography participants, but only from inside the contract. This guarantees that the control flow of the model is only managed by the contract and not influenced by external users. Different from the message function, this one has no parameters since the execution semantics of the corresponding element do

Table 3.1: Translation approach from BPMN elements to Solidity.

BPMN element	**Solidity code**	**BPMN element**	**Solidity code**
1) 	• Private function • Check on its state • Activates next element	5) 	• Private function • Check on its state • Activates next element
2) 	• Public message function • Public payment function • Check on its state • Activate next element	6) 	• Private function • Check on its state • Activates next elements
3) 	• Ethereum account • Modifier access control	7) 	• Private function • Check on its state • Activates next elements
4) Sequence Flow	• Guard expression	8) 	• Private function • Check on its state

not require external inputs.

In particular, the first considered control flow element is the *start event* (case 1) which generates a private function starting the execution of the process inside the blockchain by activating the next element. Similarly, the *end event* terminates the execution (case 8) without activating any other functions. Sequence flows instead, do not directly derive Solidity code but they are used to enforce the overall execution sequence and to derive the activation mechanism between elements. Furthermore, those elements contain the expressions used as guards evaluated inside the connected exclusive gateway. The last considered control flow elements are gateways which share the same translation logic. Indeed, all of them are represented as a private function deciding the path to follow according to some conditions. In the case of an *exclusive gateway* (case 6), this decision is evaluated through an expression previously defined. *Parallel* and *event-based* gateways enable instead all the following elements (case 6 and 7). The difference is that the event-based defines a race condition in which only the first function executed by the user is valid, thus disabling the others.

3.2 ChorChain Tool

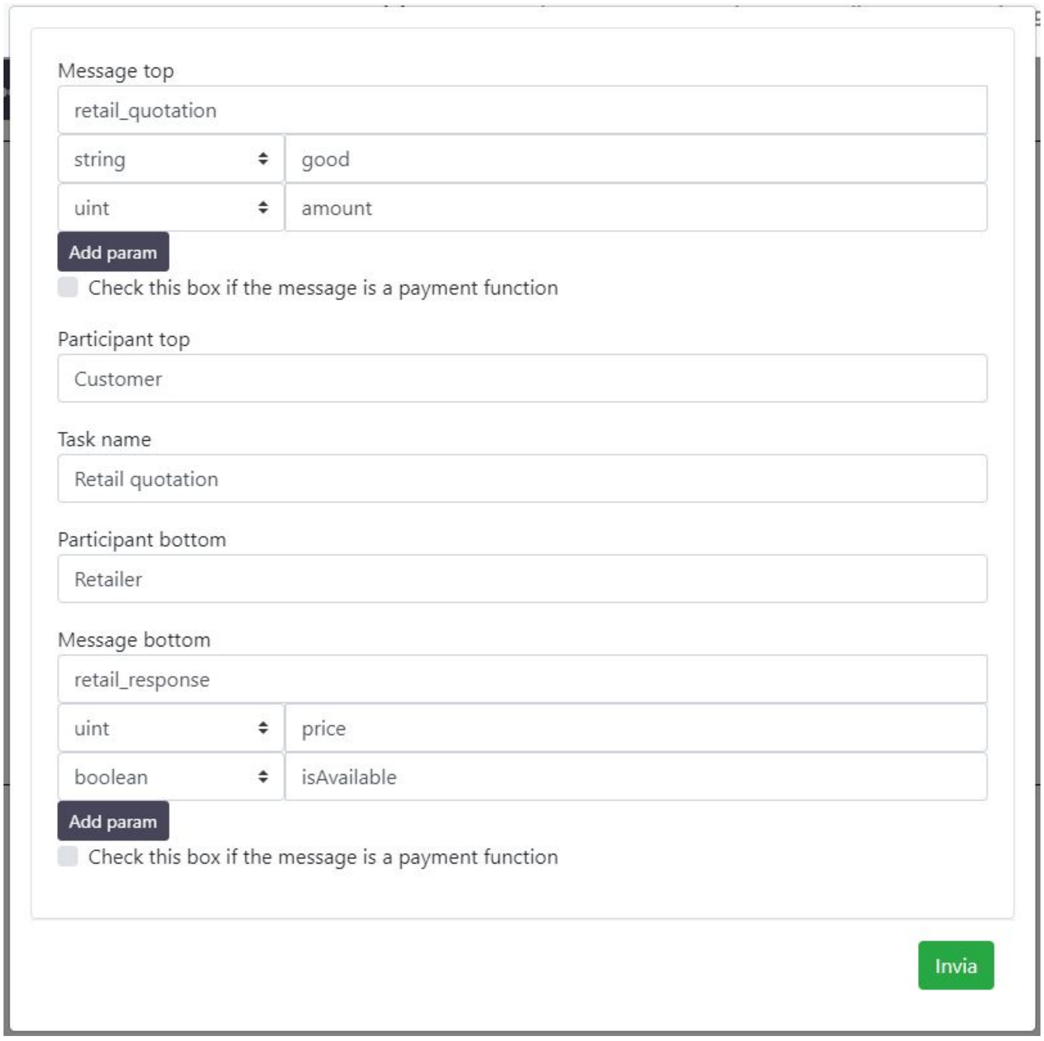
This section presents the ChorChain tool[1], by focusing on the design choices that have driven its development, and the technical solutions adopted to implement it. The presentation is organised according to all the methodology phases previously introduced, from modelling to execution.

3.2.1 Modelling

The modelling phase is the starting point of the choreography life cycle. To support it, the chor-js [65] modelling environment is integrated into the tool. More specifically, chor-js is imported in the modelling page of the ChorChain front end, which is based on Angular JS. The modelling area offers several functionalities, such as the creation, the import, the export and the saving of a model in the ChorChain repository. Some of these actions are also connected to the ChorChain back end, which is based on a JAX-RS web service. For example, when a model is saved, a REST call containing the XML of the diagram to store is directed toward the back end, which then handles it by creating and saving the file. It is worth mentioning that, from a technical point of view, in the development of the framework for the inclusion of blockchain-specific information in the models the usage of the extension mechanisms made available by the BPMN standard is avoided. The main objective, indeed, is to derive a supporting infrastructure and not to export the instrumented model out of ChorChain. Therefore, similarly to [87], the additional information is included just as text in the *name* attribute of messages, and hence it is included in the resulting XML file and graphical representation. The text is then parsed by ChorChain to generate the needed infrastructure. In the integrated modelling environment, the specification of a task is supported by an intuitive panel depicted in Figure 3.3 that requires the insertion of the related information. Furthermore, in case the *payment* checkbox is selected, the corresponding function is created and the message is automatically filled with no parameters accordingly to the payment format. In particular, for each payment function, a counter n manages the uniqueness of the names, and it is incremented at each new included function.

Retail Process Example (Modelling) Here some excerpts of the modelling phase for the retail process example are discussed. Figure 3.3 depicts the panel in the ChorChain modelling environment for the definition of the *Retail quotation* task. This is the first task in the retail process scenario that

[1]The tool can be used and tested at `http://virtualpros.unicam.it:8080/ChorChain`.



Message top

retail_quotation

string	good
uint	amount

Check this box if the message is a payment function

Participant top

Customer

Task name

Retail quotation

Participant bottom

Retailer

Message bottom

retail_response

uint	price
boolean	isAvailable

Check this box if the message is a payment function

Figure 3.3: Created panel for task and messages definition.

involves the Customer and the Retailer. The panel permits to set the name of the task, the participants, and the parameters of the exchanged messages. ChorChain provides a straightforward way to annotate the model without incurring syntactic issues. In particular, it allows for the dynamic inclusion of information in a guided manner. This functionality annotates the task in the graphical representation of the model (see Figure 2.3) with the task's name (*Retail quotation*), the participant's names (*Customer* and *Retailer*), and the messages definition ($retail\_quotation(string\ good,\ uint\ amount)$ and $retail\_response(uint\ price,\ boolean\ isAvailable)$). Figure 3.4 shows the exclusive gateway used to manage the choice coming after the first task. In particular, the progress of the model is controlled by a boolean expression that labels the outgoing sequence flow of the gateway. The guard evaluates the $isAvailable$ variable, instantiated in the precedent interaction (*Retail quo-*

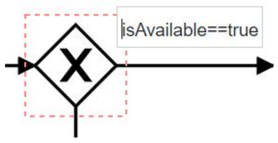
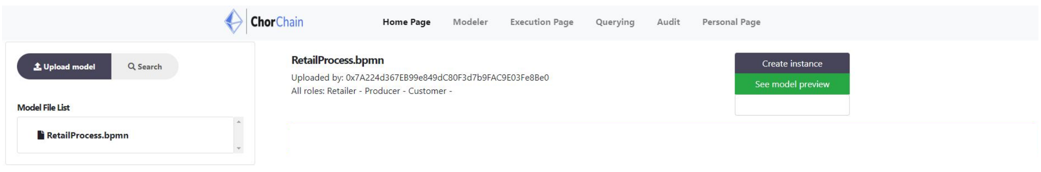


Figure 3.4: Gateway guard example.



Model File List

RetailProcess.bpmn

RetailProcess.bpmn
Uploaded by: 0x7A224d367EB99e849dC80F3d7b9FAC9E03Fe8Be0
All roles: Retailer - Producer - Customer -

Figure 3.5: ChorChain home page with uploaded model.

tation task), and it is defined using the standard boolean expression syntax for equality.

3.2.2 Publishing, Searching and Instantiation

A choreography specification can have an impact only if concrete instances are derived from it so that from time to time the participants specifically involved in the created instance interact to reach the specified choreography's goals. Therefore, ChorChain provides support for publishing, searching and instantiating a choreography specification.

To publish the choreography, ChorChain provides a repository that can be accessed via an intuitive user interface. However, in order to interact with the repository, it is necessary to register and login into the platform. To ease access to the platform, the user can exploit the Metamask browser plugin[2], which provides a web interface for managing Ethereum accounts. The account selected in Metamask then constitutes the identifier of the participant in the choreography contract. After the login, the user is redirected to the homepage depicted in Figure 3.5. On the left side of the web page, the user has the possibility to publish a new model, by uploading the corresponding file. Alternatively, it is possible to search for an existing one.

The searching phase is an important aspect of the framework since it enables reusability and facilitates the meeting between the supply and demand of services. Any registered user, once logged in, can search for a particular choreography and the framework proposes a list of all models matching the

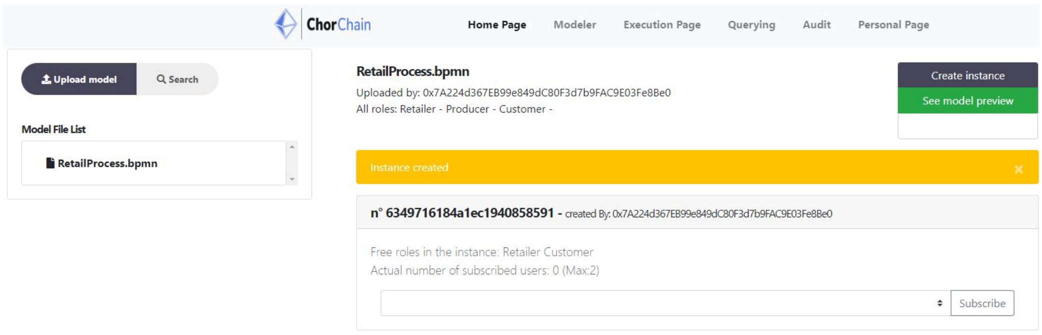
[2]`https://metamask.io/`

searched topic. These are listed below the search form (bottom left corner in Figure 3.5). At this stage, a simple discovery functionality is provided, as this is not the main focus of the thesis. More advanced mechanisms can be adopted, by resorting, for instance, to semantic annotations of choreography models. The information about the selected choreography is shown on the right side of the homepage. In particular, ChorChain shows the owner of the model, the maximum number of involved participants and the required roles. The preview of the graphical representation, and the possibility to create a new choreography contract, are also visualised.

In the instantiation phase, the user generates a new instance for a specific model. In this phase, he has to select possible optional roles, otherwise, they are considered mandatory. Consequently, the choreography instance is kept in a "suspended" state while waiting that all the mandatory roles to be subscribed. ChorChain supports both public choreographies, where participants are free to enter, and private ones where only pre-selected parties have the opportunity to join. Figure 3.1 shows three choreography instances deriving from the same choreography composed of two mandatory roles (Role A and Role B). Instance 1 is the one deployable on the blockchain since both roles are subscribed (green colour). Instances 2 and 3 are suspended since just one role is subscribed (Role B) and one is vacant (Role A, in red colour). The latter instance is instead newly created and all its roles are vacant.

When a choreography instance has no more vacant mandatory roles, ChorChain considers the partnership complete and starts the generation of the Solidity smart contract, deploying it on the blockchain. If the contract has some optional roles, the subscription form remains enabled on the homepage with only the optional roles, also after its deployment. In case a user selects an optional role, the correlated subscription function is triggered directly on the already deployed smart contract. This operation generates a standard transaction, that needs to be accepted via the metamask plugin.

Retail Process Example (Instantiation) Considering the Retail process example, Figure 3.6 depicts the home page with a created instance in which the *Customer* and *Retailer* were set as mandatory roles and are currently available to be subscribed. The *Producer*, instead, was set as optional, since its participation is not always required. This means that after the deployment of the contract, a participant can join the running instance by covering the Producer optional role.



Model File List

RetailProcess.bpmn

RetailProcess.bpmn
Uploaded by: 0x7A224d367EB99e849dC80F3d7b9FAC9E03Fe8Be0
All roles: Retailer - Producer - Customer -

Instance created

n° 6349716184a1ec1940858591 - created By: 0x7A224d367EB99e849dC80F3d7b9FAC9E03Fe8Be0

Free roles in the instance: Retailer Customer
Actual number of subscribed users: 0 (Max:2)

Figure 3.6: Home page with a created instance.

3.2.3 Smart Contract Generation

The generation of the smart contract code starts after the parsing of the choreography model, performed by means of the Camunda Java library[3], to which some functionalities were added to deal with choreography diagrams syntax as defined in the standard. More in detail, once a participant subscribes to a choreography instance if the covered role is the last one left free, the .xml file of the instance is retrieved and parsed using the Camunda functionalities. These allow the translator to iterate every BPMN element inside the instance file, creating the respective Solidity code. As mentioned above, the Camunda library was extended with some controls for the recognition of choreography tasks and messages, and some utility functions for the extraction of specific information, such as elements IDs and names. Here below are some hints of the automatic translation performed by ChorChain for each choreography element admitted by the BPMN standard following the translation logic described in Section 3.1.3.

Listing 3.1 reports the header of the contract. This part is quite standard and similar for each newly generated contract.

```
contract Choreography{

  enum State {DISABLED, ENABLED, DONE} State s;
  struct Element{string ID; State status;}
  struct StateMemory{
    string var1;
    bool var2;
    uint var3;
    ...}

  event functionDone(string eventID);

  Element[7] choreographyElements;
  StateMemory currentMemory;

  mapping(string=>address payable) roles;
  mapping(string=>address payable) optionalRoles;

```

[3]https://docs.camunda.org/javadoc/camunda-bpm-platform/7.11/

```
19   string[2] roleList = ["Role A", "Role B"];
20   string[1] optionalList = ["Role C"];
```

Listing 3.1: Contract header with state variables.

A contract keeps track of the choreography instance state by means of the list of elements `choreographyElements` (Line 13) and the structure of variables `currentMemory` (Line 14). In particular, this last one contains all the information influencing the state of the contract and they are derived from all the inputs defined in the messages. Each element of the choreography is represented as a structure of type `Element` (Line 4) containing the information related to that model element (i.e., its identifier and current status), while the current memory has type `StateMemory` (Line 5) containing instead all the global variables appearing in the model. The states of an element are defined by the enumeration `State` (Line 3). The event `functionDone` (Line 11) is emitted for each completed element, and it permits the direct retrieval of the contract transactions. In addition, the function is also used to notify the partners about a possible contract state change. The header also includes the list `roleList` and `optionalList` of the mandatory and the optional roles involved in the choreography (Lines 19-20) and generated by participants elements.

After the declaration of the global variables, the contract includes the definition of the modifier (Listing 3.2).

```
21 modifier checkRole(string memory role) {
22   require(msg.sender == roles[role] || msg.sender ==
         optionalRoles[role]);
23   _;
24 }
```

Listing 3.2: Modifier for participant and role access control.

A modifier has the same structure as a method but can be called only inside the definition of a function. In this case, the contract invokes the modifier inside the functions representing messages. Specifically, the modifier `checkRole` (Lines 21-24) checks if the mandatory/optional role of the sender in that particular function corresponds to the role for which the same account was subscribed. This construct is used to enforce, from the contract side, the right identity of the sender according to what is expressly defined in the choreography instance.

The contract constructor (Listing 3.3) is the principal method executed at contract deployment time. It performs all the operations concerning the start and participants initialisation, necessary for the subsequent execution of the contract.

```
25 constructor() public{
26   elements[0] = Element("StartEvent_0gb8jks", State.ENABLED);
27   //roles initialisation with participants addresses
28   roles["Role A"] = 0x9bAAf595...;
29    roles["Role B"] = 0x2BBc833C...;
30    optionalRoles["Role C"] = 0x0000000...;
31    emit functionDone("Contract creation");
32    StartEvent_0gb8jks();
33  }
34
35  //function for the optional subscribe
36  function sub_as_part(string memory _role) public {
37    if(optionalRoles[_role]==0x00000000...){
38      optionalRoles[_role]=msg.sender;
39    }
40  }
```

Listing 3.3: Contract constructor with participant initialisation and start event invocation.

The first operation performed by the constructor is the initialisation of the `start event` element (Line 26) with its status set to `ENABLED`. Lines 28-30 report the initialisation of the roles with the addresses of the participants. The mandatory roles are hard-coded in the constructor since they are roles without which the execution of the choreography model can take place. For the optional roles, instead, the participants can subscribe at run-time using the `sub_as_part` function in (Lines 36-39). At this point the event related to the contract creation is emitted (Line 31); this informs the external users that the contract is deployed and ready to be executed. Finally, in Line 32, the function related to the enabled start event is invoked, allowing the next elements to be executed.

After the generation of this first part, which is similar to each choreography contract, the generation continues by appending the functions corresponding to the translation of the elements included in the choreography model. Listing 3.4 shows the public function depicting a message exchanged between two participants in a choreography task. The function name in the contract is represented by the message identifier inherited from the model, while the parameters are from the annotated name.

```
41  function Message_ID(string memory _var1) public
42                checkRole(roleList[1]) {
43    //checking the status of the current element that is the invoked message
44    require(elements[0].status == State.ENABLED);
45    currentMemory.var1 = _var1;
46    done(0);
47    //it enables the next element
48    enable("Next_Element_ID", 1);
49  }
```

Listing 3.4: A message function.

First of all, the modifier `checkRole` is called with the assigned role (Line 42). Once the right identity of the caller inside the function is ascertained, a second check on the status of the task is performed: as expected it should be equal to `ENABLED` (Lines 44). After that, it is performed the registration of the `_var1` parameter in the memory of the contract (Line 45). The last step before enabling the execution of the next element is to change the status of the current element to `DONE` (Line 46) and set to `ENABLED` the status of the next element (Line 48).

After messages, which translation produces code with the same structure, the other considered elements are control flow ones. More specifically, these are gateways directly translated as private functions. In Listing 3.5 there is the representation of the *parallel gateway*.

```
50 function parallelGateway_ID() private {
51   require(elements[1].status == State.ENABLED);
52   done(1);
53   //enable the next elements
54   enable("Next_Element_ID1", 2);
55   enable("Next_Element_ID2", 3);
56   ...
57 }
```

Listing 3.5: Function for a parallel gateway.

In addition to the private visibility and the absence of inputs (see Section 3.1.3), the other characteristic of the control flow functions is the absence of the modifier for the role checking since there is no interaction coming from outside the blockchain. The body of the function contains just the code to complete the current element (Line 52) and to enable the next ones directly connected (Lines 54-55). A similar code can be expected for an *event-based gateway*, as in principle all the tasks connected to that gateway have to be enabled, and only when the message has received the tasks waiting for the antagonistic messages have to be disabled. A different implementation is expected for an *exclusive gateway*, where the logic of the function is depicted in Listing 3.6. In this case, the next element is enabled only after the evaluation of a condition that discriminates which element to enable. This condition is inherited from the boolean expressions annotated in the outgoing sequence flows of the exclusive gateway. In the smart contract, the choice is managed by using an if-else control (Lines 60-67); this gives more guarantee on the mutual exclusion but limits the execution to the first satisfied condition.

```
58 function exclusiveGateway_ID() private {
59   require(elements[2].status == State.ENABLED);
60   if(currentMemory.var2==false){
61   //If the next is an internal element it is invoked
62     enable("Next_Element_ID", 4);
63     Next_Element_ID();
64   }else if(currentMemory.var2==true){
65   //if the next is a message it is only enabled
66     enable("Next_Element_2", 5);
67   }
68   done(2);
69 }
```

Listing 3.6: Function for an exclusive gateway.

The state changes for the elements are simplified by the usage of three auxiliary functions defined in Listing 3.7. The `enable` function takes as input a string representing the element identifier and a numeric position. This information is used to create and push a new element in the list with the `ENABLED` status (Line 71). The `disable` and `done` functions are similar; they change the status of the indicated element position to `DISABLED` and `DONE`, respectively. The only difference is in the `done` function, which notifies the participants about the current completed element by emitting the `functionDone event` (Line 79).

```
70 function enable(string memory _taskID, uint position) internal {
71   elements[position] = Element(_taskID, State.ENABLED);
72 }
73 function disable(uint position) internal {
74     elements[position].status=State.DISABLED;
75 }
76
77 function done(uint position) internal {
78     elements[position].status=State.DONE;
79     emit functionDone(elements[position].ID);
80 }
```

Listing 3.7: State change functions.

Finally, a specific function, reported in Listing 3.8, is used for dealing with payments.

```
81 function payment() public payable checkRole(roleList[0]){
82     require(elements[3].status==State.ENABLED);
83     roles["Role B"].transfer(msg.value);
84     done(3);
85     enable("Next_Element_ID", 6);
86 }
```

Listing 3.8: Function for a payment.

This function is used to send money to the receiving participant indicated in the model. The function is marked as `payable` and allows the account of the participant executing it to transfer Ether. The `transfer` function (Line 83) sends to the addressee (the role between squared brackets) an amount of Ether specified by `msg.value`, which is defined by the sender user.

Retail Process Example (Contract Generation) This example presents a sketch of the smart contract generated after the subscriptions of the users for the retail process example. Listing 3.9 depicts the representation of the *retail_ quotation* message into a Solidity function, while Listing 3.10 shows the code generated for the exclusive gateway after the *retail quotation* task. The first function is now identified in the smart contract using the type and the "ID" of the element in the BPMN model (see Line 87). The second one, instead, does not require any parameter and it only checks the expressions (Lines 96, 98) derived from the guards `isAvailable==false` and `isAvailable==true` annotating the sequence flows outgoing from the exclusive gateway in the BPMN model. Depending on this check, it enables the next element. For a more complete account of the translation of the retail process example into the corresponding Solidity contract, the contract code is available on-Line[4].

```
function Message_0b917rc(string memory good, uint amount) public
    checkRole(roleList[1]){
  require(elements[2].status==State.ENABLED);
  enable("Message_1xxdwx2",3);
    currentMemory.good=good;
    currentMemory.amount=amount;
    done(2);
}
```

Listing 3.9: Code corresponding to the *retail_ quotation* message.

```
function ExclusiveGateway_042aut8() private {
    require(elements[4].status==State.ENABLED);
    if(currentMemory.isAvailable==false){
        enable("Message_1h3ew61", 5);
    }else if(currentMemory.isAvailable==true){
        enable("ExclusiveGateway_1johog7", 6);
        ExclusiveGateway_1johog7();
    }
    done(4);
}
```

Listing 3.10: Code corresponding to the exclusive gateway after the *retail quotation* task.

3.2.4 Deployment

Once the contract has been generated, the framework automatically deploys it into the Ethereum blockchain.

[4]The Solidity contract of the retail process example is available at the following link https://bit.ly/39GxxJU

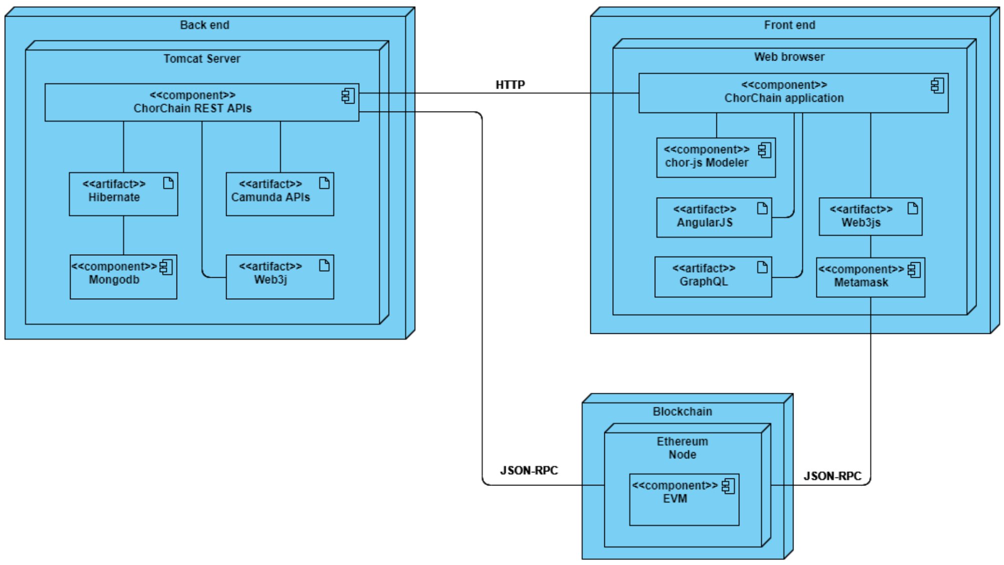


Figure 3.7: ChorChain architecture with software components.

To get a better understanding of this process, a first clarification is made on the architectural organisation of ChorChain depicted in Figure 3.7. The framework follows the typical *DApp* architecture, and it can be divided into the following components: the back end, the front end and the blockchain.

The Back end contains a web server, running on a Tomcat server, that provides the REST APIs for communication. It also includes the database for storing information related to the choreography models/instances and their subscription state exploiting the Hibernate framework. The Web3j library is used for interacting with the blockchain, and the Camunda APIs are for parsing the choreography models.

The Front end communicates through the ChorChain REST APIs with the back end. In particular, the front end contains the ChorChain web application providing the interface for the interaction with the tool. It is created using the AngularJS framework and it includes the chor-js modeller for the design of the BPMN choreographies, the GraphQL API for supporting custom blockchain queries, and the Web3js library (connected to the Metamask service) for enabling the user-blockchain interactions.

The Blockchain is connected to the other components by means of JSON-RPC communication using the Web3 APIs targeting an Ethereum node of the network. When the smart contracts are triggered, the Ethereum Virtual Machine (EVM) inside the node is used for executing the Solidity code.

The interactions between a participant, the ChorChain components and the Ethereum blockchain, for the subscription and the deployment of the choreography contracts, can be now summarised as depicted in Figure 3.8. Given a suspended contract, the participant can subscribe to a role by sending a request to the back end, through the front end. At this point, two alternatives are admitted. In case all the mandatory roles are covered, the ChorChain back end generates the smart contract, by sending a transaction to the blockchain using the *web3j* library[5]. The Ethereum blockchain uniquely identifies the contract with a hexadecimal number, which is generated only after the corresponding transaction is mined. The contract creation event is then caught by the back end, which updates the front end, and shares the contract identifier among all participants. The choreography execution can now be started. The other alternative in Figure 3.8 describes instead the case where the instance is still incomplete. In this case, the back end just updates the subscribed role without involving the blockchain.

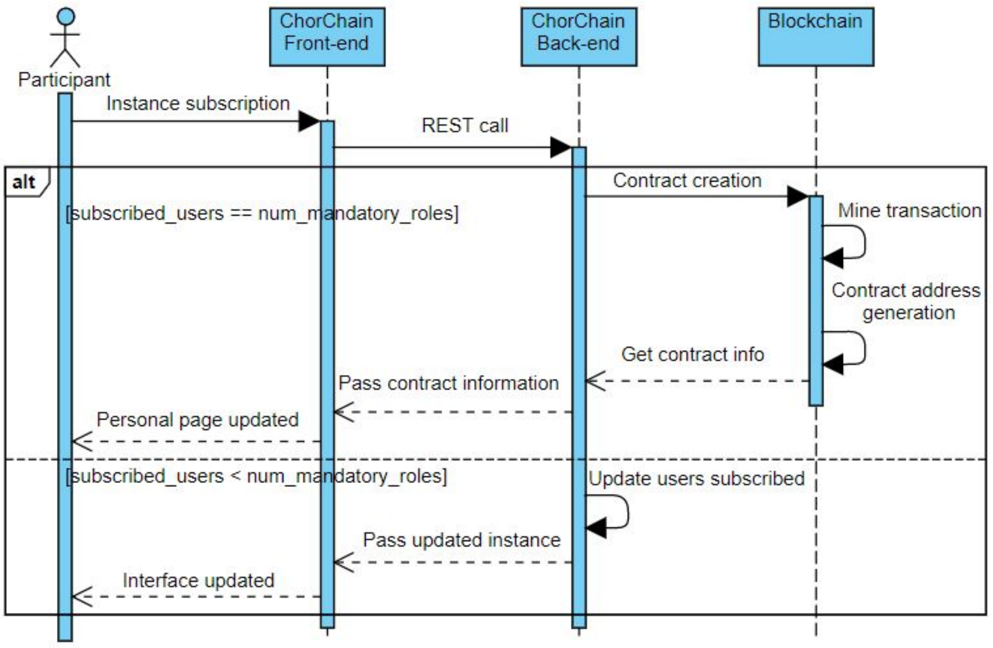


Figure 3.8: ChorChain deployment phase.

3.2.5 Execution

Once a new contract is deployed into the blockchain, the execution phase takes place and the participants can collaborate by means of the functions exposed by the contract. In order to facilitate these interactions, ChorChain provides an execution page accessible by each (human) participant shown in Figure 3.9.

[5]`https://web3j.io`

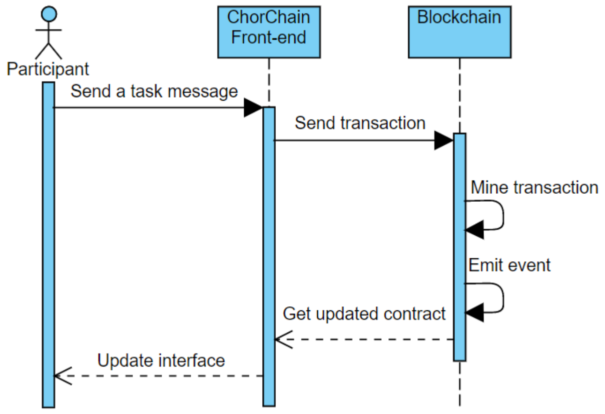
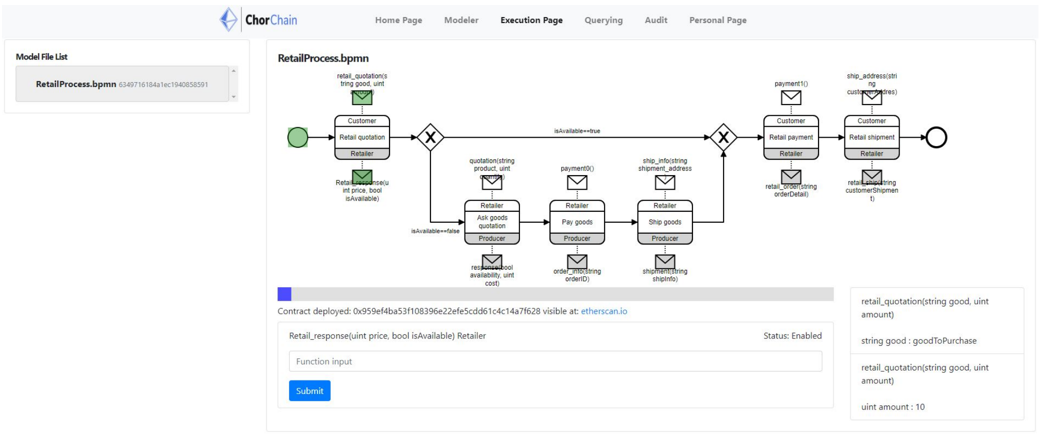


Figure 3.10: ChorChain execution phase.


ChorChain
Home Page
Modeler
Execution Page
Querying
Audit
Personal Page
Model File List
RetailProcess.bpmn 6349716184a1ec1940858591
RetailProcess.bpmn
retail_quotation(string good, uint amount)
Customer
Retail quotation
Retailer
Retail_response(uint price, bool isAvailable)
isAvailable==true
isAvailable==false
quotation(string product, uint quantity)
Retailer
Ask goods quotation
Producer
response(bool availability, uint cost)
payment0()
Retailer
Pay goods
Producer
order_info(string orderID)
ship_info(string shipment_address
Retailer
Ship goods
Producer
shipment(string shipInfo)
payment1()
Customer
Retail payment
Retailer
retail_order(string orderDetail)
ship_address(string customerAddres)
Customer
Retail shipment
Retailer
retail_ship(string customerShipment)
Function input
Submit

Contract deployed: 0x959ef4ba53f108396e22efe5cdd61c4c14a7f628 visible at: etherscan.io

Retail_response(uint price, bool isAvailable) Retailer

Status: Enabled

retail_quotation(string good, uint amount)

string good : goodToPurchase

retail_quotation(string good, uint amount)

uint amount : 10

Figure 3.9: ChorChain participant execution page.

The left-hand side of this interface reports a list of all contracts to which the participant is subscribed. On the right-hand side, it is shown a preview of the model: in green, there are the messages done, and the ones actually active below the respective forms that are dynamically constructed by ChorChain for executing them. Each form contains information, like the name of the message, the role of the participant, the space for inserting all the required parameters, and the submit button. Notably, the submission form is visible only to the participant in charge of sending the enabled message. In addition, by double-clicking on a completed message, a little panel with the exchanged values is shown.

The sequence diagram reported in Figure 3.10 summarises the steps for the execution of a single message. To send it, the participant has to fill up the corresponding form and then click on the submit button. The generated transaction has to be confirmed using the Metamask pop-up. It contains the gas price plus the total amount of Ether to spend for the transaction. As soon as the transaction is included in a block (i.e., it is mined), the related event is emitted. This event is used by the front end to update the interfaces of all participants involved in the choreography with the new contract status, thus enabling the next admitted message(s). It is worth noticing that the choreography is executed in a distributed manner since the participants interact via the front end directly with the blockchain, without referring anymore to the back end component.

Retail Process Example (Execution) The example in Figure 3.9 depicts the execution page for the *Retailer*. The *request_ quotation* message in green on the model represents the completed activity performed by the customer that requested a good to purchase with an amount. The *Retailer* at this point answers using the dedicated form, inserting the availability as a boolean value and its cost. The transaction can be then sent to the blockchain using the submit button.

3.3 Experiments and Validation

In this section, the thesis illustrates the results of the experiments carried out to assess both the cost (in terms of transaction fees) of the execution via the ChorChain framework of choreography-based systems and the performance and scalability of the facilities provided by the framework. First, the experiments on the cost of the proposed running example are presented[6], and then the validation is extended by analysing the costs and performances of a collection of synthetically generated choreographies. Overall, the results of our experimentation show that ChorChain has the potential to be effectively used in practice.

Notice, all the considered case studies assume a common interest among participants in keeping transparent the overall execution and identities. For this reason, user anonymity is not considered a core property of ChorChain but it is demanded in a permissioned approach as presented in Chapter 7.

[6]It is worth noticing that, beyond the Retail process case study, the approach was tested also on scenarios from other application domains (e-commerce, hotel booking, procedures for student internship and exam registration, review of scientific papers, smart heating system management, etc.), to check how the ChorChain approach works in different application contexts. The implementations of these scenarios are available at the following link: `https://bitbucket.org/proslabteam/chorchain/src/master/Examples/`.

Table 3.2: Cost Analysis for the Retail scenario.

Transaction Name	**Gas Used Shortest path**	**Gas Used Longest path**
contract creation	3,482,898	3,482,898
retail_quotation	126,765	126,765
retail_response	221,246	153,657
optional subscribe	none	45,686
quotation	none	126,741
response	none	106,011
payment0	none	92,603
order_info	none	107,013
ship_info	none	104,508
shipment	none	154,621
payment1	92,648	92,648
retail_order	104,473	104,473
ship_address	104,520	104,484
retail_ship	107,142	107,082

3.3.1 Running Example Cost Analysis

Following the proposed methodology, the choreography model of the Retail scenario depicted in Figure 2.3 has been published into the ChorChain framework. Then, two new choreography instances were generated and, after the subscription of participants, the contracts were deployed in the blockchain. The first contract was subscribed by the two mandatory participants and they followed the shortest path. In the second case, the optional *producer* participant subscribed at run-time, participating in the execution. The path followed this time is the longest one and it comprises the messages exchanged by the retailer and the producer, including also a payment. The two choreography instances were then completely executed from the start to the end event. Then, from the transaction logs, the *gas used* parameter was extracted, as it corresponds to the pure cost for executing smart contract code. Indeed, external factors such as the gas price, depending on the network status, can significantly impact the resulting fees. On the contrary, the gas used exclusively depends on the proposed approach so it is the most interesting unit of measurement. The experiments had been performed on September 18Th, 2022 on the Sepolia Testnet blockchain. The reader can refer to Sect. 3.2 for insights into the Solidity code, while the executed transactions are publicly accessible on the Etherscan website[7].

[7]Transactions for the shortest path are available at the link `https://sepolia.etherscan.io/address/0xb927340c2a5bace6ae0c3144a1f9d7bf8e291a6b`, while those for the longest one are at the link `https://sepolia.etherscan.io/address/0xd88e798efc5a87bd20daf5dba5ffaa9e8f783035`.

Table 3.2 reports the cost analysis for the retail contracts according to the two different paths. For each transaction, its name and the used gas are reported. Looking at the transactions, the first concerns contract creation, while the remaining are related to the control flow and the exchange of messages between participants. The contract creation transactions are the most expensive: they consumed around 3.4 millions unit of Gas. All the other transactions, instead, are rather cheap, since the used amount of gas ranges from a minimum of 92,648 to a maximum of 221,246.

Except for the contract creation, the fees paid during the contract execution are charged to the user accounts. The amount requested can be simply calculated as follows:

$$\textit{gas used} \times \textit{gas price} \;=\; \textit{total fee}$$

Notice, the gas price is a parameter chosen by the user according to its necessity to include the transactions in a faster way. The execution of the shortest path in Table 3.2 without including the contract creation requires 756,794 gas units. As expected, the total gas used is mainly influenced by the contract creation, while the number of exchanged messages has a minor impact. Setting the gas price to 0.000000001 Eth (1 Gwei), the execution of the whole path corresponds to 0.0007568 Eth, which at the date of the experiment corresponds to USD 0.99 given the Eth-USD exchange rate of 1,312.84.[8].

The gas price could be increased by the user for improving the registration of the transactions. However, the obtained benefits are limited by technology constraints given by the public network. To have a better transaction inclusion throughput the ChorChain framework could be deployed in a private Ethereum blockchain. Nevertheless, the cost of using the public net is still reasonable in return for the trust.

To provide a more detailed evaluation of the gas consumption for the retail process, Figure 3.11 shows the cumulative graph resulting from the two execution paths represented in Table 3.2. The graph shows that the consumed gas increases linearly with the number of messages. This trend is encouraging since it can be kept under control. Also, the current (October 2022) Ethereum block gas limit is 30 million, which hence permits without big restrictions to deploy and execute contracts that can be created for each choreography. In case, the gas limitation could be simply overcome by splitting the contract into separate sub-contracts.

[8]Notably, the resulting fee in US dollars is volatile due to the high fluctuations of the Ethereum exchange rate. The reader can easily calculate the cost at a given date in fiat money by using an online converter (e.g., `https://coinmarketcap.com/converter/eth/usd/`).

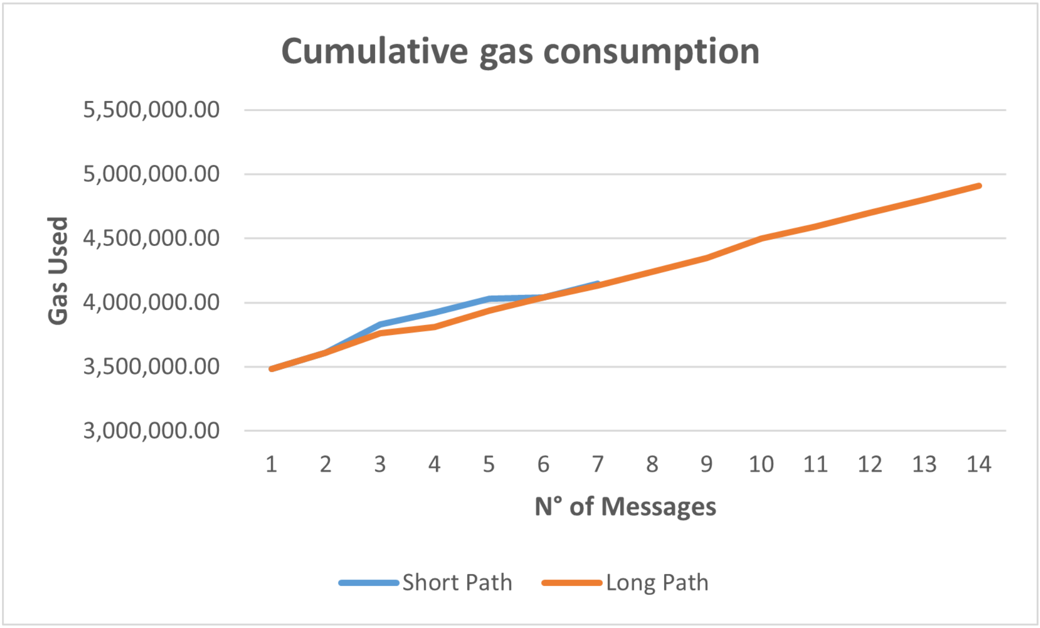


Figure 3.11: Cumulative gas consumption in the Retail scenario.

3.3.2 Choreography Elements Analysis

Here are reported experiments performed on 30 different choreographies aimed at measuring the effectiveness of the approach by isolating and executing all the BPMN choreography elements supported by ChorChain. Specifically, for each element, a family of simple models were created, with a growing number of elements in order to capture the incremental trend of costs and times, both for the deployment and for the execution. Below the experiments carried out for each choreography element are discussed and they are related to choreography task, exclusive gateway, parallel gateway, and event-based gateway.

Choreography task Figure 3.12 shows the last model in which 5 choreographies are created containing from 1 to 5 tasks. Notice, as explained in Section 3.2.3, during the translation of tasks, ChorChain considers only the messages they exchange.

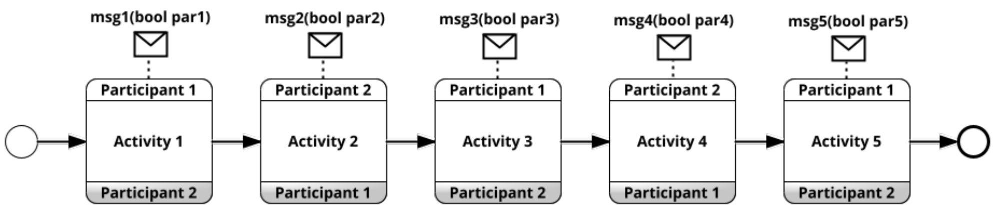


Figure 3.12: Choreography model with five tasks.

The results regarding the deployment of the 5 contracts are reported in Table 3.3. Here the transactions were included on average every 13 seconds with a gas cost growing from a minimum of 1 million, for the contract with only one message, to a maximum of 1.5 million for the contract with 5 messages (hence, 5 functions in the contract). From this data, it is possible to observe that each new message added to the model costs on average around 123,000 units of gas in the deployment.

Table 3.3: Time and cost analysis for the deployment of the task element.

	1 Message	**2 Message**	**3 Message**	**4 Message**	**5 Message**
Deploy - Time (s)	16.133	6.024	11.537	16.200	16.150
Deploy - Gas used	1,081,646	1,204,181	1,326,819	1,449,434	1,575,483

Table 3.4, instead, reports the execution times and costs for the five contracts. In particular, each column contains the execution times/costs for each message exchanged in the corresponding contract; thus, the first column contains only one measure of execution time and one measure for the consumed gas, while the last column contains the measures for five messages. The execution times are very variable and cannot be associated directly with a behaviour. A different consideration can be done for the gas usage; it is indeed possible to observe that the average gas usage for each message is 92,904.73.

Table 3.4: Time and cost analysis for the task element.

	1 task	**2 task**	**3 task**	**4 task**	**5 task**
Time	24.005	20.369	9.387	19.019	21.197
		10.969	11.966	13.811	19.695
			22.474	26.971	22.780
				25.380	15.426
					29.744
Gas used	106,011	103,397	103,397	103,419	103,397
		88,961	86,325	86,347	86,325
			89,005	86,391	86,369
				88,939	86,348
					88,940

Exclusive gateway The performance of the exclusive gateway element by means of 5 choreography models has been analysed, with an incremental number of split-join pairs of gateways, the last of which is reported in Figure 3.13.

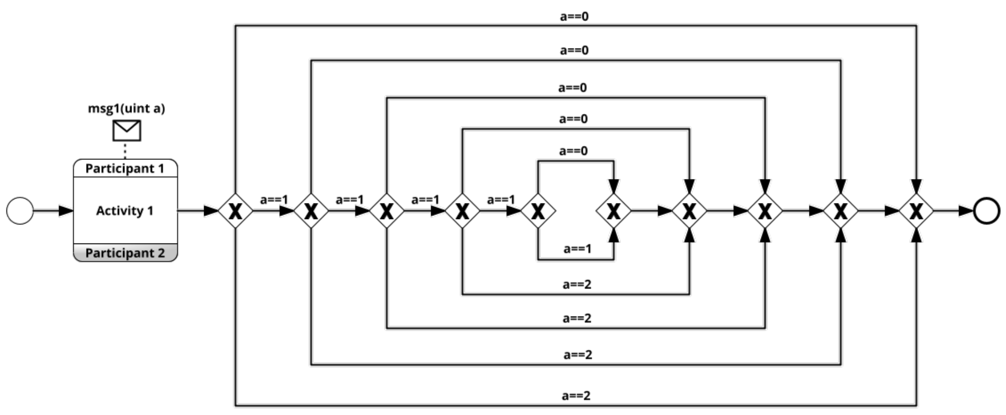


Figure 3.13: Choreography model with five split and five join exclusive gateways.

These models are composed of an initial dummy task, which has the role of initialising the variable used for executing the exclusive paths after the gateways. According to the definition of the translation of the exclusive gateway (see Section 3.2.3), all gateways are represented by internal functions that are executed automatically by the execution of a message (in our models, this is the dummy message at the beginning), hence their executions are included in the transaction of the message. For this reason, in all reported experiments the cost of the gateway is included in the cost of a message function. Table 3.5 reports the costs in terms of gas and time required for the deployment and execution of the models. The average time for the deployment is quite standard, around 9 seconds, while the gas used is higher and varies from a minimum of 1.1 million to a maximum of 1.5 million. Each pair of gateways, consisting of a split and a join, impacts around 108,000 units of gas. For the execution, differently from the case of the task element, the measure for every single element is not reported because, as mentioned above, the gateway transactions are executed internally in the contract and are included in the transaction of the message. Thus, once the first message is executed, all the split and join elements are automatically triggered until the end event. In this case, there is a very different behaviour, especially when analysing the execution time that is much higher and around 23 seconds. The transaction size ranges from a minimum of 201,000 to a maximum of 582,000. It is also possible to notice that each split-join pair increases the transaction by around 95,000 units of gas.

Table 3.5: Deployment and execution time and cost for split and join exclusive gateways.

	1 Split 1 Join	**2 Split 2 Join**	**3 Split 3 Join**	**4 Split 4 Join**	**5 Split 5 Join**
Deploy - Time (s)	1.700	5.550	9.240	17.972	13.862
Deploy - Gas used	1,161,950	1,270,822	1,379,516	1,488,198	1,596,887
Execution - Time (s)	17.114	22.090	19.352	40.100	17.500
Execution - Gas used	201,231	296,644	392,057	487,348	582,640

Another experiment was conducted on exclusive gateways, with the aim of isolating the behaviour of the split elements, using 5 models with a growing number of paths, the last of which is reported in Figure 3.14. Here it is important to measure the impact of the splitting element since this is one of the most recurrent elements in the choreography model.

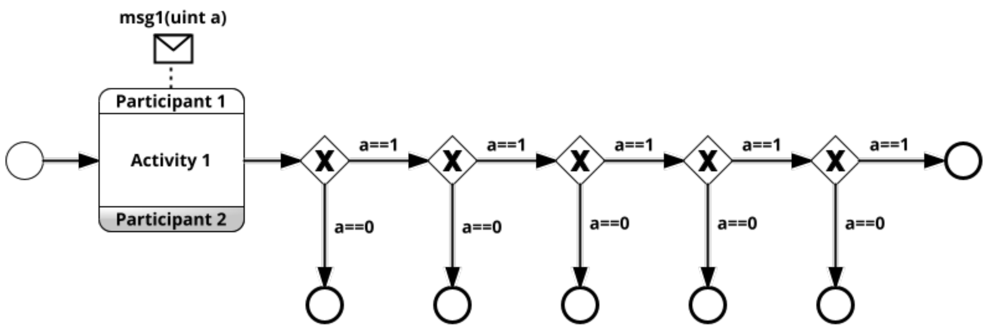


Figure 3.14: Choreography model with five split gateways.

The results reported in Table 3.6 show an average deployment time of 10 seconds, with a gas cost lower with respect to the previous case. In fact, this time each new contract has an increment of 73,000 units of gas with respect to the 108,000 of the previous experiment. Thus, it is possible to derive that each join element costs around 35,000 units of gas. For what concerns the execution, there is an average time of 20 seconds, which could be impacted by the cost of the transactions that range from 153,680 to 344,648 units of gas. This time the average cost without the join element is lower and the cost of each split is around 47,000 units of gas. An interesting result comes from the comparison of the execution costs of the models with and without the join element. Indeed, it is possible to extract that the two gateways cost around 47,000 each.

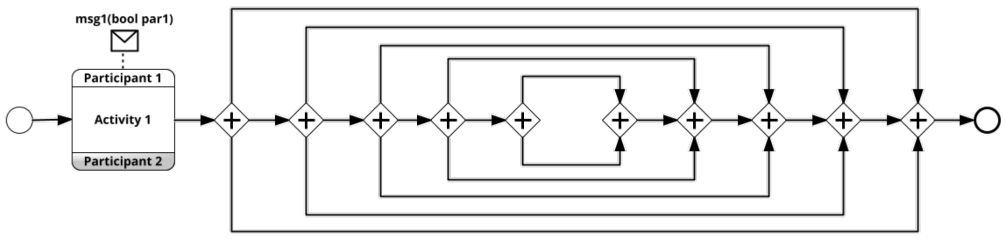


Figure 3.15: Choreography model with five split and join parallel gateways.

Parallel gateway Similar models to those used for the exclusive gateway were analysed to measure the impact of parallel gateways, with and without join elements. In particular, the first 5 models have a growing number of both split and join, the last one reported in Figure 3.15, from which the results reported in Table 3.7 are reported.

Table 3.6: Deployment and execution time and cost for the split exclusive gateway.

	1 Split	**2 Split**	**3 Split**	**4 Split**	**5 Split**
Deploy - Time (s)	12.970	6.849	14.154	15.119	9.792
Deploy - Gas used	1.146.617	1,220,152	1,293,490	1,366,823	1,440,156
Execution -Time (s)	25.314	17.160	8.325	30.001	19.935
Execution - Gas used	153,680	201,452	249,225	296,997	344,648

It results to be the most expensive element since the deployment of the model with 10 elements (5 splits and 5 joins) reaches the cost of 1,750,864 units of gas, and its execution is 748,032 units of gas. This behaviour comes from how the parallel gateway operates: differently from the exclusive gateway, it activates and triggers all the outgoing paths, significantly increasing the amount of code to execute and, as consequence, the overall cost. It is also interesting to notice that, this time, the growing number of elements leads to a linear increment of the used gas.

Table 3.7: Deployment and execution time and cost for split and join parallel gateways.

	1 Parallel	**2 Parallel**	**3 Parallel**	**4 Parallel**	**5 Parallel**
Deploy - Time (s)	13.804	18.574	18.049	11.712	2.480
Deploy - Gas used	1,191,686	1,331,617	1,471,358	1,611,113	1,750,864
Execution - Time (s)	43.564	20.699	16.956	17.312	30.871
Execution - Gas used	213,992	333,555	462,413	600,571	748,032

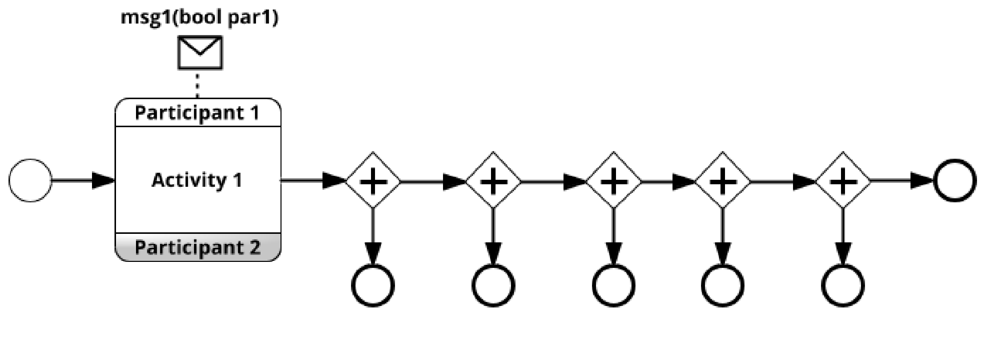


Figure 3.16: Choreography model with five split parallel gateways.

The second analysis performed on the parallel gateway aims at isolating the behaviour of the split element reported in Figure 3.16. Table 3.8 reports the related times and costs. Here there is no particular behaviour; in general, every single split element increases the deployment cost of around 65,000 units of gas and the execution cost of 95,000 units of gas.

Table 3.8: Deployment and execution time and cost for the split parallel gateway.

	1 Parallel	**2 Parallel**	**3 Parallel**	**4 Parallel**	**5 Parallel**
Deploy - Time (s)	9.227	1.800	15.278	7.375	8.823
Deploy - Gas used	1,147,637	1,213,618	1,279,437	1,345,215	1,411,010
Execution - Time (s)	16.949	36.552	16.915	6.472	22.285
Execution - Gas used	201,003	296,018	391,033	486,048	581,064

Event-based gateway For the next experiment, due to the nature of the event-based gateway, in these experiments, five choreographies were used with a single event-based gateway having an increasing number of messages, and a dummy task at the start. The resulting model is reported in Figure 3.17. Table 3.9 reports the experiment data related to the deployment and execution costs and times. The deployment is the most expensive one, with a maximum of 1,918,233 units of gas, while the execution has a quite standard range. This derives from the fact that many messages have to be included, considerably increasing the deployment cost. For the execution, instead, the reported transactions do not include the executions of the messages connected to the gateway, so it is possible to observe only the cost of executing the dummy message and the activation of the event-based outgoing paths.

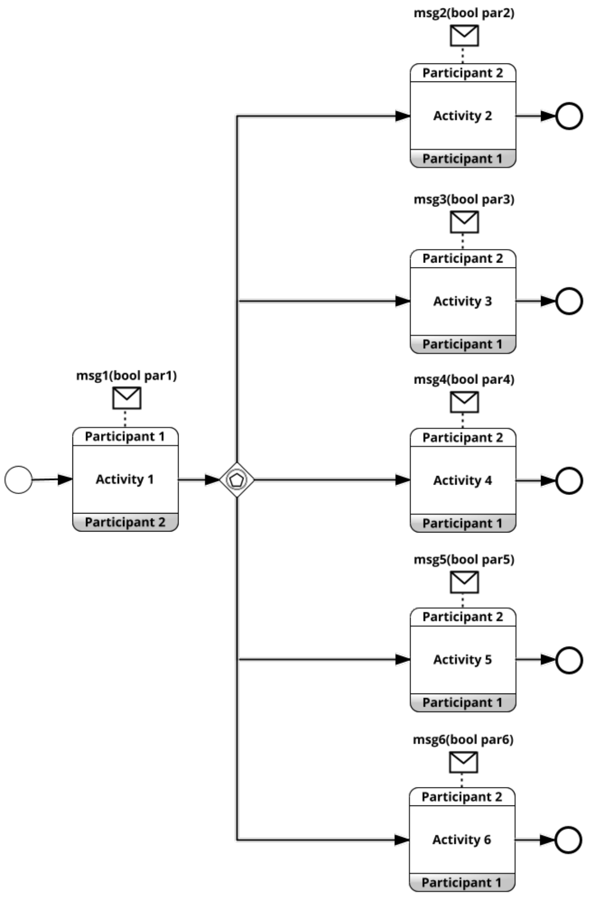


Figure 3.17: Choreography model including an event-based gateway with five messages.

Table 3.9: Deployment and execution time and cost for the event-based gateway.

	1 Event	**2 Event**	**3 Event**	**4 Event**	**5 Event**
Deploy - Time (s)	12.488	8.752	13.584	13.382	16.321
Deploy - Gas used	1,237,189	1,409,482	1,573,563	1,745,489	1,918,233
Execution - Time (s)	15.072	10.773	16.131	16.314	25.864
Execution - Gas used	150,910	195,790	240,692	285,596	330,476

The results presented in this section show a variety of elements tested in different situations, for assessing how the single elements impact the approach. The costs discussed above could seem high but is important to consider that they are all encapsulated in a single transaction representing the initial message sending, which is added to the model for enabling the experiments. In a realistic scenario, like the proposed running example, it never happens to have a large number of gateways concatenated without messages, so the costs for a single transaction are significantly reduced. Moreover, when talking about the ChorChain costs, it is necessary to consider that the execution of the model is split between the involved parties, as each participant pays only the transactions corresponding to the sent messages.

Finally, it is worth mentioning that an experimental comparison of execution costs with other similar tools (like Caterpillar and Lorikeet) could not be done. In fact, it turned out that it is not possible to run the same models in the different tools, because either they rely on different kinds of diagrams (choreography vs. collaboration), or there are anyway some issues in using our choreography models.

3.3.3 Real-world Use Cases

To further show the feasibility and the applicability of the ChorChain approach, two additional experiments on use cases taken from real-world situations are reported. The first scenario was originally presented in [48], but the one under consideration is the revised and analysed version of [120]. To deal with the ChorChain architecture, to each task of the original choreography model, messages were added and formatted as described in Section 3.2.

The resulting model is reported in Figure 3.18; the example involves six business partners and describes the manufacturing and delivering process of product orders. Specifically, the Bulk buyer starts by placing an order with the Manufacturer, which orders the calculated materials to a Middleman. In its own turn, the Middleman forwards the request to a Supplier and also arranges transportation by means of a Special carrier. At this point, the Special carrier picks the materials and delivers them to the Manufacturer, which interacts with the Bulk buyer to finalise the order.

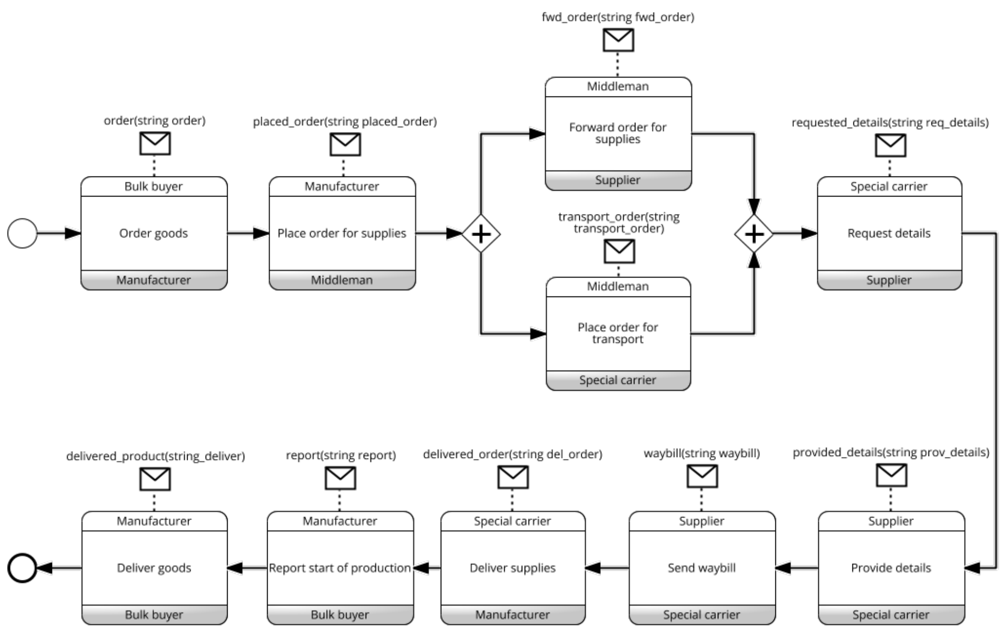


Figure 3.18: Adapted choreography from the supply chain scenario.

The supply chain choreography was instantiated and the corresponding smart contract was executed using ChorChain. The results of this experiment are reported in Table 3.10. As for the previous examples, the most expensive transaction is contract creation, which requires 2,802,543 units of gas. For the total execution, instead, the supply chain scenario requires 1,156,734 units of gas[9]. From these results, linear behaviour can be noticed since each transaction has a similar cost. The only exceptions concern the messages before the split and join parallel gateways since they need to activate them in the smart contract. In particular, the *placed_ order* transaction consumes 196,838 units of gas required to activate all the next elements.

[9]The entire execution is available at `https://bit.ly/Sup_Chain`.

Table 3.10: Cost Analysis for the supply chain scenario.

Transaction Name	**Gas Used**
contract creation	2,802,543
order	104,416
placed_order	196,838
fwd_order	109,538
transport_order	116,939
requested_details	104,342
provided_details	104,446
waybill	104,337
delivered_order	104,477
report	104,364
delivered_product	107,037

The second use case concerns an Incident Management choreography taken from [88]. The model was adapted to meet the ChorChain requirements and the result is shown in Figure 3.19. The choreography represents an incident management process of a software manufacturer and is started by a customer asking support due to a problem in the purchased product. In the first place, the account manager is responsible for providing a solution to the customer. In the negative case, request is forwarded to the 1st level support agent and, in case the issue is still not resolved, to the 2nd level support. As final solution, the 2nd level support can also contact the software developer asking for an opinion. Assuming a solution is finally found, this is forwarded back to the account manager that explain it to the customer.

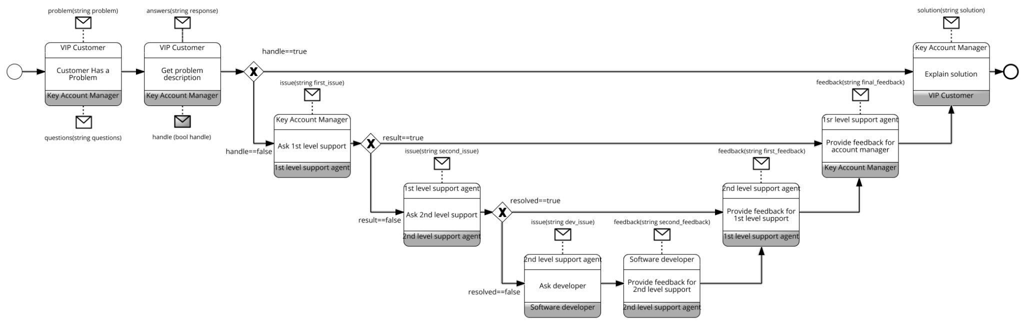


Figure 3.19: Adapted choreography from the incident management scenario.

Table 3.11 shows the execution costs of the longest path in which all the messages are sent. Compared with the performance of the previous scenario, here the contract creation has a higher cost (3,278,656 units of gas), since the choreography, hence the smart contract contains more elements. The execution cost, instead, remains lower and requires a total of 1,440,071 units of gas.

Table 3.11: Cost Analysis for the incident management scenario.

Transaction Name	**Gas Used**
contract creation	3,278,656
problem	104,394
questions	104,406
answer	104,316
handle	131,207
issue1	104,395
result1	131,161
issue2	104,365
resolved	131,227
issue3	104,351
feedback1	104,422
feedback2	104,409
feedback3	104,410
solution	107,008

As in the previous case, the transactions corresponding to the sending of a message have a quite similar cost, while the ones activating a gateway consume more gas units. This time the highest transaction costs are around 131,000 units of gas; they activate the internal choice of the exclusive gateways[10].

3.4 Comparison with Existing Approaches

The usage of choreography-based specifications, to drive the development of multi-party distributed systems, has been extensively studied and investigated. In the last years, the EU commission financed various projects specifically devoted to the topic (see, for instance, CHOReOS[11][13] and CHOREVOLUTION[12]). The relevance of choreographies in relation to the description of complex inter-organisational systems, and the need for suitable infrastructures for the management of the choreography life-cycle, is discussed in [7]. In particular, the authors propose ServicePot, a complex choreography registry elevating choreography specifications to first-class citizens, in order to facilitate the dynamic integration and interoperability of services managed and made available by different organisations. A correlated work is the one in [14], where the authors propose an automatic approach for enforcing the realizability of choreographies providing adapters. Starting from a choreography model and a set of services, the proposed adaptation and coordination

[10]The entire execution is available at `https://bit.ly/IncidentMan`.

[11]https://cordis.europa.eu/project/rcn/96288/factsheet/en

[12]http://www.chorevolution.eu/bin/view/Main/

solution allows services to collaborate according to the choreography specification. In [20] the authors propose a correct-by-construction method to build realisable choreographies described using conversation protocols, while in [38] the authors propose an approach for checking the conformance of possible system implementations, with respect to choreographies.

Similarly, the use of blockchain technology in the development of decentralised applications is discussed thoroughly in the literature [122] and different works move in this direction, reinforcing the motivations that led the effort in this context. In particular, in [54] the authors propose an application for automatic gasoline purchase to highlight the concrete possibilities and limitations related to the adoption of blockchain technologies. Among the various possibilities, the authors identify three main important aspects that can be synthesised in: transparency, longevity and trust. The latter is a key aspect also in this thesis. Other concrete applications can be found in environments where the innate characteristics of this new technology can replace old system components. For example, in [125] the authors propose a design approach for building a blockchain-based product traceability system, replacing the central database with the blockchain. In the same direction, in [77] the authors presented a blockchain agent-based simulator for cities in which the agents communicate via smart contracts. This solution takes advantage of blockchain decentralisation and of its encryption mechanism that stores information in a secure way.

In [100] the authors explain the importance of developing blockchain-oriented software using smart contracts in a model-driven approach. In this scenario, BPMN is recognised as an enabler for the process-driven development of contracts. Also, the authors in [15] present a conceptual model-driven approach based on BPMN choreographies, whose target platform relies on Hyperledger technologies. Other works investigate instead how blockchain technologies can be used in enterprise modelling to provide a trusted mechanism for attesting the existence of information in a transparent and publicly verifiable way. In particular, in [49] the authors present the concept of Knowledge Blockchain, which is a solution for storing in an immutable and tamper-resistant way the knowledge in enterprise models exploiting the blockchain. In a similar direction, in [55] the authors propose the same approach applied to ontologies. In [56], instead, the authors exploit the blockchain for the decentralised attestation of information and knowledge represented in conceptual models. These works have in common the scope to guarantee trust by exploiting blockchain technology; however, the ChorChain framework has an additional focus on the enforcement of the execution phase. In [81] the blockchain is recognised as a key enabler for the use of BPMN choreography diagrams as appropriate abstractions for the top-down design of cross-organisational business processes.

Choreographies are used also in [66], where the authors propose an extension to the BPMN 2.0 standard in order to give more expressiveness to blockchain concepts. In particular, the proposed elements are related to data objects, sub and call choreographies, condition expressions and script tasks. Differently, the main goal of ChorChain is to use already existing notation elements to support the full life-cycle in the blockchain, without adding extension elements to the language.

Finally, in [51] the authors explain the advantages of a decentralised blockchain solution for cross-organisational workflow management, highlighting the possibility to use the blockchain as an instrument for auditing manual operations performed in relation to process execution.

Among the works combining process management and blockchains, it is certainly worth mentioning [120]. Here the authors propose the usage of smart contracts for monitoring and coordinating multi-organisational collaborative business processes. Similarly to ChorChain, choreography diagrams are the input model and the lack of trust is the main driver for the work. On the other hand, this work differs in its objectives. In fact, in [120] the choreography model is used to generate a monitor and a mediator. The former checks the conformance of the messages exchanged by the participants with respect to the protocol defined by the choreography; in case of a non-conforming message, an alert is broadcast. The latter plays an active role, by sending and receiving messages and by performing computation and data transformation. ChorChain objective instead, is the enforcement of the interactions among the choreography participants preventing the execution of non-conforming interactions. Only the conforming actions are proposed to the participants via the interfaces automatically generated and, anyway, also in case the interface would be bypassed, the logic inside the smart contract enables only the conforming interactions. Other differences between the two approaches concern some technicalities. In [120], a factory smart contract is generated, while ChorChain directly generates a smart contract for each specific choreography instance so as to deal with the subscription of participants in a previous phase of the choreography life-cycle. Moreover, the considered fragment of the BPMN choreography element is richer; in particular, ChorChain supports the event-based gateway that plays an important role in modelling collaborative processes.

In [107] collaboration diagrams provide a framework permitting the execution of decentralised processes exploiting blockchain-related technologies. Apart from a different kind of model used to represent the processes cooperation (collaborations, not choreographies), this approach differs from ChorChain as it introduces a scaffolding smart contract that must be instantiated for each model instance. This implies that, for each instance, a sequence of transactions is required to instrument the contract with the logic of the model, specifying tasks and organisation addresses. This architecture reduces the deployment cost (the used gas is always 1,265,261) since the smart contract has an 'empty' logic, but it requires additional transactions to upload the logic of the model (around 127,000 gas units for each included element), which introduce a long delay. Another weakness is related to the need for a supervisor that has to fill the smart contract, who has to be a trusted third party in contrast with the decentralised nature of the blockchain. Differently, ChorChain generates a new contract already filled with all the information necessary for the execution. This may lead to a higher deployment cost, but the execution has no additional set-up costs or delays.

The work in [106] extends the architecture proposed in [107], considering the lack of trust issue in an inter-organisational context. The authors use collaboration diagrams, translated into process models, to provide a framework permitting the execution of decentralised processes, exploiting blockchain-related technologies. Differently, in [61] collaborations are translated directly to smart contracts, without requiring a reduction to a single pool process. Nevertheless, the logic of some elements has to be added manually, while in ChorChain once the model is created no additions are needed. The cited work presents features like the division of the deployment costs, which is out of the scope of this work, and the pre-creation of a consortium between participants, which can lead to a possible centralisation point. In addition, their mechanism of events subscription is optional, while in ChorChain it is included by design, allowing a built-in direct and decentralised communication between the user client and the blockchain.

In [74] and [75] the Caterpillar tool is proposed. This is one of the first attempts to support the combination of business process management with blockchain infrastructure. The tool takes as an input a process model and transforms it into Solidity code. Again, the use of a different kind of diagram distinguishes this proposal from the one illustrated in this thesis, and the same considerations reported above apply here. Extensions of this tool are presented in [71], where the authors propose a dynamic role-binding model and a binding policy language for supporting collaborative business processes. A similar extension of Caterpillar is proposed by the same authors in [72], where a list of components is provided for the update of models and their smart contracts at run-time, in order to react to unexpected situations during the execution. Lorikeet [111] is a similar tool, which however focuses more on the asset management and business process interactions on the blockchain.

Simple process models are also used in [90] where the authors extended the LabChain tool to support the execution of business processes on DLTs. However, the lack of model expressiveness and the tool purpose are the main differences with ChorChain. Indeed, via choreographies, it is possible to support complicated interactions. In addition, the proposed tool was built from scratch to provide a complete set of functionalities to work with BPMN processes. A similar direction is followed in [29, 28], where the authors translate BPEL processes into smart contracts. However, in this case, the authors are mainly interested in investigating how to ensure data confidentiality in the presence of an untrusted oracle. In [52], the authors provide an optimised execution method for contracts generated from business process models via a 2-step translation. Firstly, a model is translated into a Petri net, which is optimised by removing 'tau' transactions, and then the resulting Petri net is translated into a smart contract. ChorChain instead avoids the use of an intermediate language, like Petri nets, as it provides a direct translation from choreography models to the Solidity language. Indeed, ChorChain aims at achieving a one-to-one correspondence between the choreography elements of the model and the Solidity functions of the generated smart contract, which allows a compositional construction of the code and results in a readable and understandable smart contract. In this way, it is not required any additional transformation phase, as the created BPMN choreography (following the ChorChain principles) contains all the information necessary to be directly deployed and executed. Furthermore, the natural translation of choreographies without intermediate models does not need to be optimised and it permits to achieve a lighter translation process.

It is worth mentioning that the works reported above mainly, or only, focus on aspects related to the generation of the smart contract, while they generally overlook integration aspects related to the need for an infrastructure to support the whole life-cycle of choreographies and processes. ChorChain, instead, permits the derivation of a concrete implementation of choreography models, by relying on the underlying blockchain technology. The whole approach is encapsulated in a user-friendly framework that allows the developer to deal with all the phases of the choreography life-cycle, from the modelling to the deployment and execution. All these phases are supported by a web-based interface, easily accessible also to users not familiar with blockchain-related technologies. Furthermore, ChorChain is the first tool implementing choreographies into a smart contract following a model-driven approach and permitting their execution in a trustable way, also supporting naively the exchange of Ether represented as payments at the choreography level.

CHAPTER 4

ChorChain: AUDITING INTER-ORGANISATIONAL BUSINESS PROCESS EXECUTION

In this chapter, the thesis presents the challenge of auditing inter-organisational business processes implemented and executed on the blockchain. Indeed, the immutability of the data stored in the blockchain, and the access of data in a transparent, secure, and consistent way, open the possibility to apply auditing techniques for detecting possible deviations and anomalies that happened during the execution of the smart contracts [30] in a completely independent manner [12]. This is reflected in the ChorChain approach, where the guarantees on stored data enable the auditing of exchanged messages during the choreography-based system execution. To support such capabilities, the ChorChain framework was extended including an additional phase dedicated to the auditing of the executed process. This phase was concretely implemented by providing **auditing strategies** which take their basis on a **conceptual model** and that can be exploited by the involved parties. In this way, the time and cost of auditing contracts are reduced, without involving manual operations on a set of selected transactions. This makes also it easier to investigate deviations highlighting anomalies during choreography execution since auditors are enabled to monitor the process in a continuous way and are kept updated simultaneously on information disclosure [99]. The next section introduces the ChorChain framework extension for supporting auditing capabilities regarding the con-

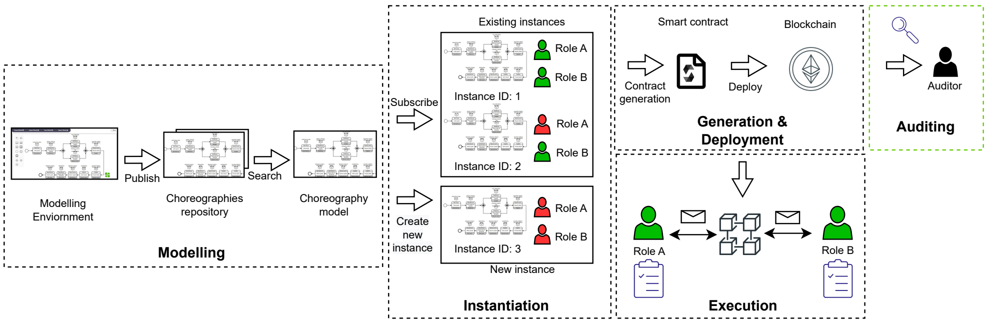


Figure 4.1: ChorChain framework: supporting auditing

ceptual model and auditing strategies. The implemented capabilities are then presented in the ChorChain tool with a final evaluation involving practitioners. Finally, a comparison with related works is provided.

4.1 ChorChain Extended Framework for Auditing

The provided auditing mechanisms are a direct outcome of the proposed auditing strategy that can be considered choreography-centric. This strategy relies on the retrieval of information related to a choreography model and to the execution of possible instances. Such mechanisms complement those related to the enforcement of the interactions among the choreography participants. The objective is to enable the assessment of constraints related to data exchanged by the participants in a choreography instance, as well as time and gas-related aspects with respect to a choreography instance execution. The combined usage of enforcement and auditing mechanisms can highly increase accountability and trust in a multi-organisational interaction context. Figure 4.1 depicts the ChorChain framework extended with the auditing phase. The ChorChain auditing mechanisms are based on two main elements. The first one is a **conceptual model** that has been defined to structure the retrieval of relevant information from the blockchain, for auditing purposes. The second one complements the first one, defining the **auditing strategies** that have been implemented in ChorChain and can be accessed via a graphical user interface.

4.1.1 Conceptual Model

ChorChain auditing mechanisms are based on a carefully elaborated conceptual model that has driven the development of the querying infrastructure made available by ChorChain. The definition of this conceptual model

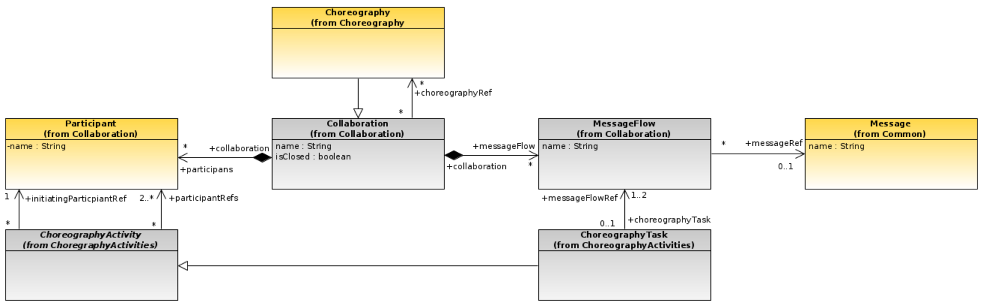


Figure 4.2: BPMN Choreography meta-model excerpt.

started with the identification of relevant concepts within the BPMN choreography meta-model. The idea was to identify that information considered relevant for the auditing purpose that is related to a choreography model, and that could also be associated with an execution trace. Figure 4.2 reports an excerpt of the BPMN choreography meta-model highlighting (in light orange) the concepts that, as a result of such activity, have been included in the conceptual model. Successively, concrete manifestations of such concepts were added within a trace execution. The resulting conceptual model is represented in Figure 4.3, where six different entities and related relations are included. The model constituted the base for the definition of the data model adopted by the ChorChain auditing mechanisms.

The entities included in the conceptual model are as follows:

- *Choreography* relates to the BPMN model diagram stored in the ChorChain repository;
- *Participant* related to a participant that plays a specific role in the choreography;
- *Message* relates to the single interaction between two participants in a choreography;
- *User* relates to and identifies the blockchain account and can assume a specific participant role in the choreography;
- *Choreography contract* relates to the instance generated from a choreography model in the form of a smart contract;
- *Transaction* relates to a blockchain transaction containing either the message execution with the corresponding payload or the contract deployment.

In addition, relations among the presented entities included in the conceptual model, and that are relevant to define the retrieving mechanisms that relate to the information stored in the blockchain, are following presented:

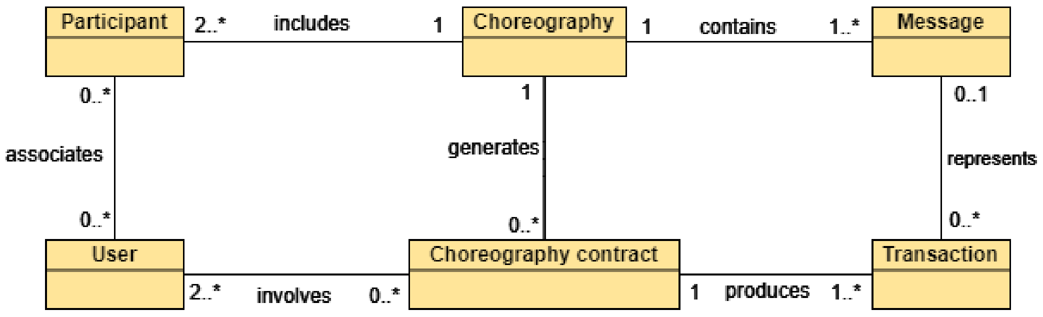


Figure 4.3: ChorChain conceptual model.

- The *includes* relationship between a Choreography and a Participant underlines that a choreography includes from two participants (this is certainly the case where the choreography includes just a single task) till N participants. A participant can belong to a single choreography model since it is not possible to assume that the connected role has the same behaviour in every model.

- The *contains* relationship between Choreography and Message confirms that the choreography has at least one message. Each message belongs to a single choreography.

- The *generates* relationship between Choreography and Choreography contract shows that a choreography relates to any number of choreography contracts, even zero if nobody instantiated it. Each contract, of course, refers to a given choreography model.

- The *produces* relationship between Choreography contract and Transaction underlines the fact that the contract must have at least one transaction, corresponding to the contract creation. A transaction corresponds to the occurrence of a specific interaction in a given contract.

- The *involves* relationship between a Choreography contract and a User confirms that the contract has at least two users. A user could be bound to zero or more contracts according to the instances in which he/she subscribes.

- The *associates* relationship between Participant and User shows that a participant role could be not subscribed by any user, this is the case of an optional role that has not been subscribed.

- The *represents* relationship between Transaction and Message represents the fact that a message could have zero or more transactions associated. A transaction represents at most one message; it does not refer to a message when it corresponds to the contract deployment.

4.1.2 Auditing Strategies

In ChorChain it is possible to audit information relative to choreographies with three different strategies, each of them corresponding to a specific view in the ChorChain tool. The first two strategies are completely automatic and transparent for the auditor since the information can be extracted from the blockchain and navigated through the ChorChain user interface without the need for any technical knowledge. The third one, instead, consists of directly using the 'raw' query engine, which can be used by expert auditors that own technical skills. The high-level views of the first two strategies allow the auditors to concentrate on their main objectives, as they can easily browse the choreography-related data set looking for relevant information. The third strategy is left for those users that know how to build queries and extrapolate information within the blocks.

Table 4.1: Choreography-related parameters.

Choreography
• list of generated choreography contracts • number of generated choreography contracts • number and percentage of completed choreography contracts • all the users for each participant • cardinality of users for each participant • all the transactions for each message • cardinality of the transactions for each message • all the users involved for each message • min, max and avg execution time, calculated on the completed choreography contracts • min, max and avg gas used, calculated on the completed choreography contracts

Table 4.2: Choreography contract related parameters.

Choreography Contract
• blockchain contract address • total cost, both in fees and gas • cumulative cost/percentage for each user both in fees and gas • emitted transactions • completion status • involved users, and their role • transaction(s) generated for each message

Table 4.3: Transaction related parameters.

Transaction
• timestamp • user sender • gas consumed • fee paid • message payload

Table 4.4: User-related parameters.

User
• list of choreography models in which the user was involved • all covered participant roles • participant roles covered in each choreography contract • number and percentage of completed choreography contracts • min, max and avg execution time for all completed choreography contracts • min, max and avg gas used for all completed choreography contracts

The first strategy is the **Choreography-based** one and it allows for the retrieval of information starting from the selection of a choreography entity. In particular, this strategy is based on the information reported in Tables 4.1–4.3 and related to a choreography, its contract implementation and the resulting transactions. From here, the system will start the extraction of data from the blockchain and will organise it for making the information easily accessible to the user. This view is accessible by both internal and external auditors, who in this way have a global and complete vision of all occurred interactions.

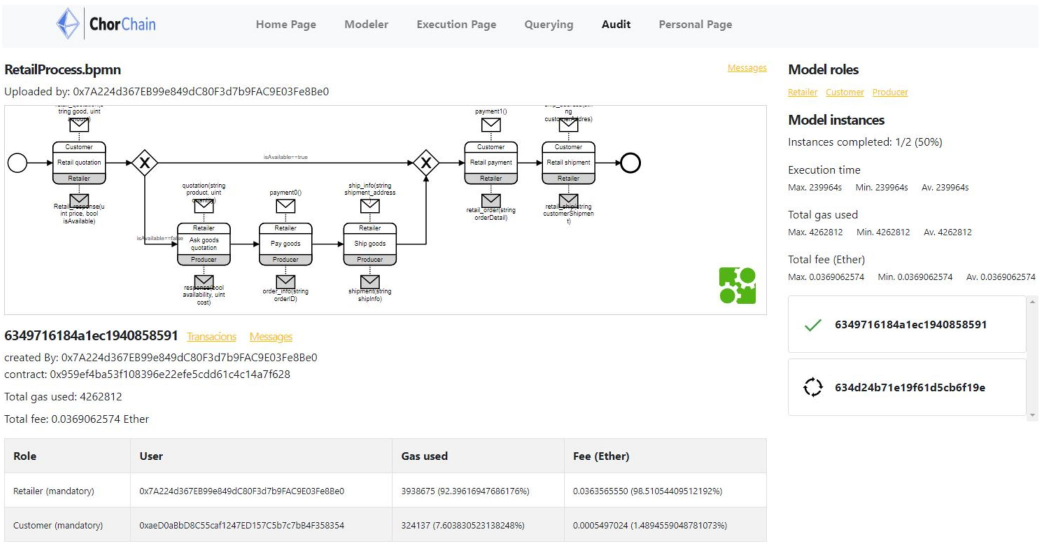
The second strategy is **User-based** and it is similar to the previous one, but with a different starting point. Indeed, its aim is to present only the choreography models in which the user was directly involved, showing the information described in Table 4.4. Notably, this information is retrieved from the blockchain. Access to such a perspective is also dependent on the auditor's role. In particular, only internal auditors can use this perspective to access details on the choreographies in which a role has been played.

The third strategy provides a **Query engine** with low-level interactions with the blockchain data, hence it is targeted to expert users that intend to query directly the blockchain. This last case was included to enable the exploration of the blockchain without any limitation given by the boundaries of the ChorChain approach.

4.2 ChorChain Extended Tool for Auditing

This section describes how the ChorChain tool was extended to concretely support the presented choreography-based, user-based and query engine auditing strategies. Those were implemented inside new dedicated pages that are accessible to all interested users.

The first implemented strategy is the **Choreography-based** reported in Figure 4.4. The graphical user interface allows the auditors to retrieve information in relation to specific choreography models. The page is organised into three main parts and allows the user to navigate the information listed in Tables 4.1–4.3.



RetailProcess.bpmn

Uploaded by: 0x7A224d367EB99e849dC80F3d7b9FAC9E03Fe8Be0

6349716184a1ec1940858591

created By: 0x7A224d367EB99e849dC80F3d7b9FAC9E03Fe8Be0
contract: 0x959ef4ba53f108396e22efe5cdd61c4c14a7f628

Total gas used: 4262812

Total fee: 0.0369062574 Ether

Role	User	Gas used	Fee (Ether)
Retailer (mandatory)	0x7A224d367EB99e849dC80F3d7b9FAC9E03Fe8Be0	3938675 (92.39616947686176%)	0.0363565550 (98.51054409512192%)
Customer (mandatory)	0xaeD0aBbD8C55caf1247ED157C5b7c7bB4F358354	324137 (7.603830523138248%)	0.0005497024 (1.489455904878107 3%)

Model roles

Model instances

Instances completed: 1/2 (50%)

Execution time
Max. 239964s Min. 239964s Av. 239964s

Total gas used
Max. 4262812 Min. 4262812 Av. 4262812

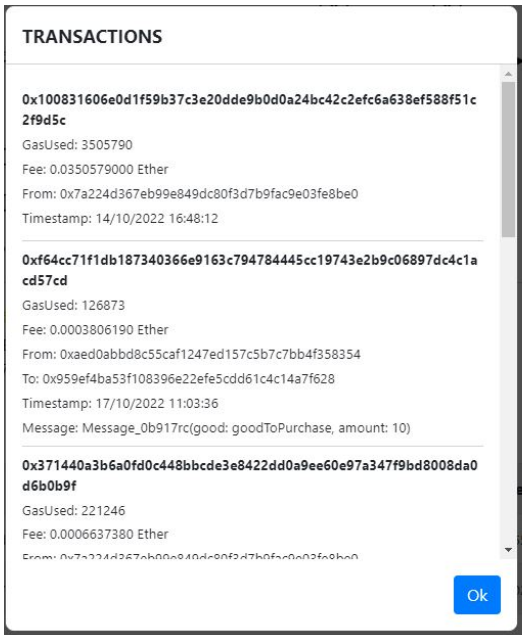
Total fee (Ether)
Max. 0.0369062574 Min. 0.0369062574 Av. 0.0369062574

✓ **6349716184a1ec1940858591**

634d24b71e19f61d5cb6f19e

Figure 4.4: Choreography-based auditing page.

On the left-hand side, it is shown the list of all choreography models uploaded in ChorChain. Clicking on one of them, the corresponding BPMN choreography diagram appears in the centre, together with all the deployed contracts on its right. Clicking on a specific contract, the ChorChain system shows under the model the information concerning the respective transactions. Here, two additional views are accessible via the *Transactions* and *Messages* links. As shown in Figures 4.5 and 4.6, the two panels contain the information related to all the exchanged messages of the choreography contract (i.e., the process instance), and their associated transactions. In this way, an interested auditor can see the execution path written in the blockchain and automatically retrieved by ChorChain. The sequence diagram in Figure 4.7 graphically describes the steps performed in the auditing phase using the ChorChain interface. The first operation consists of the



TRANSACTIONS

0x100831606e0d1f59b37c3e20dde9b0d0a24bc42c2efc6a638ef588f51c2f9d5c

GasUsed: 3505790

Fee: 0.0350579000 Ether

From: 0x7a224d367eb99e849dc80f3d7b9fac9e03fe8be0

Timestamp: 14/10/2022 16:48:12

0xf64cc71f1db187340366e9163c794784445cc19743e2b9c06897dc4c1acd57cd

GasUsed: 126873

Fee: 0.0003806190 Ether

From: 0xaed0abbd8c55caf1247ed157c5b7c7bb4f358354

To: 0x959ef4ba53f108396e22efe5cdd61c4c14a7f628

Timestamp: 17/10/2022 11:03:36

Message: Message_0b917rc(good: goodToPurchase, amount: 10)

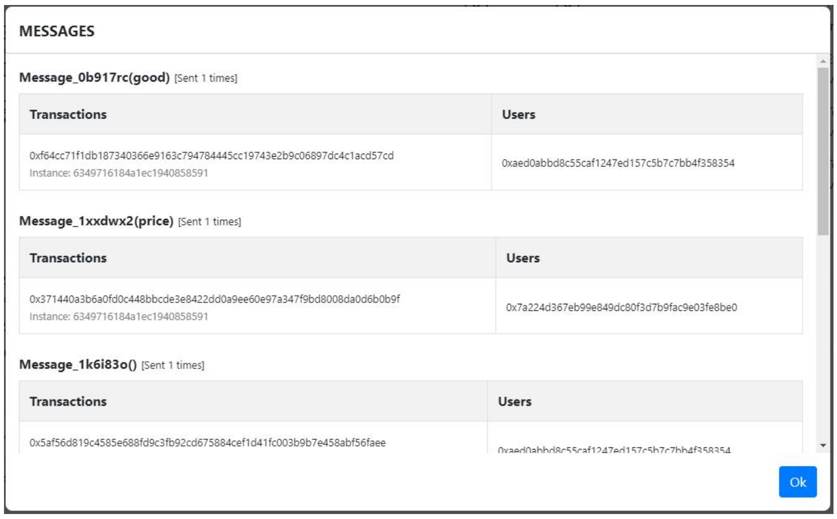
0x371440a3b6a0fd0c448bbcde3e8422dd0a9ee60e97a347f9bd8008da0d6b0b9f

GasUsed: 221246

Fee: 0.0006637380 Ether

From: 0x7a224d367eb99e849dc80f3d7b9fac9e03fe8be0

Figure 4.5: Transactions view for a single instance.



MESSAGES

Message_0b917rc(good) [Sent 1 times]

Transactions	Users
0xf64cc71f1db187340366e9163c794784445cc19743e2b9c06897dc4c1acd57cd Instance: 6349716184a1ec1940858591	0xaed0abbd8c55caf1247ed157c5b7c7bb4f358354

Message_1xxdwx2(price) [Sent 1 times]

Transactions	Users
0x371440a3b6a0fd0c448bbcde3e8422dd0a9ee60e97a347f9bd8008da0d6b0b9f Instance: 6349716184a1ec1940858591	0x7a224d367eb99e849dc80f3d7b9fac9e03fe8be0

Message_1k6i83o() [Sent 1 times]

Transactions	Users
0x5af56d819c4585e688fd9c3fb92cd675884cef1d41fc003b9b7e458abf56faee	0xaed0abbd8c55caf1247ed157c5b7c7bb4f358354

Figure 4.6: Messages view for a single instance.

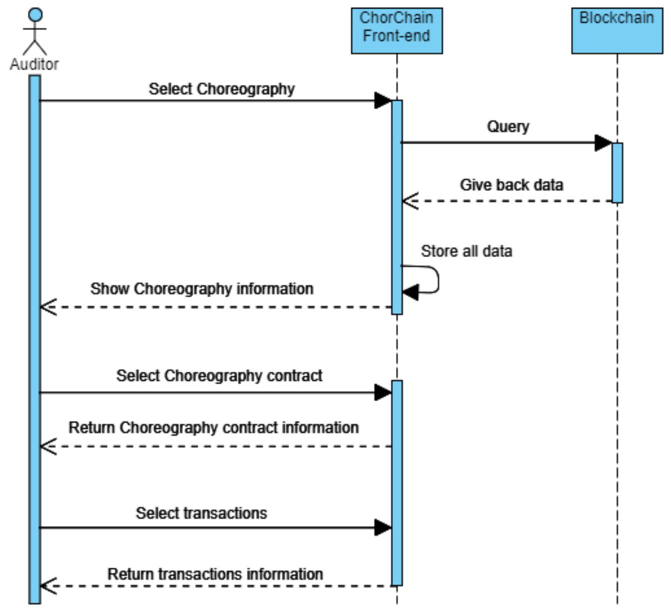
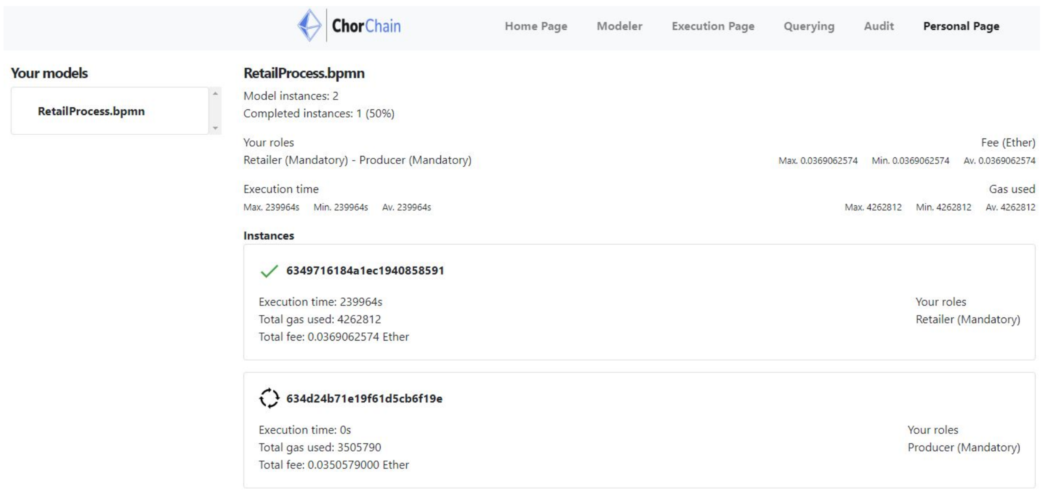

Auditor
ChorChain Front-end
Blockchain
Select Choreography
Query
Give back data
Store all data
Show Choreography information
Select Choreography contract
Return Choreography contract information
Select transactions
Return transactions information

Figure 4.7: Audit sequence diagram.

selection of a choreography, which triggers a set of preconfigured queries to retrieve from the blockchain the information regarding the model and all the connected contracts and transactions. To limit the interactions and provide a better user experience, all the retrieved data is kept in memory. At this point, the auditor can select a specific choreography contract and the related transactions, without interacting anymore with the blockchain.

The second implemented strategy is the **User-based** one, reported in Figure 4.8. This time the ChorChain interface is used to audit those choreographies in which a user played a role according to the data reported in Table 4.4. In this case, the user has the opportunity to select one model at a time, and all the contracts for which the user was one of the participants are displayed. In this view, an analysis of costs and completion statistics is provided.

The last strategy is the **Query engine** available on a dedicated page in Figure 4.9 showing a GraphQL query. The syntax used is composed of the hash of the transaction and the list of parameters to extract. This auditing strategy is built on top of GraphQL, a modern query language introduced in Geth (the standalone Ethereum client) since the 1.9.0 version, allowing an Ethereum node to open a dedicated endpoint. This makes it possible to create a flexible query mechanism for extracting extra information not shown in the previous strategies and permits to avoid waste of bandwidth and faster querying response. Obviously, in this case, the user needs to know the inputs to be used in the query (e.g., transaction hash or block number) that can be retrieved on the execution page or in the other auditing strategies.



Your models

RetailProcess.bpmn

RetailProcess.bpmn

Model instances: 2

Completed instances: 1 (50%)

Your roles

Retailer (Mandatory) - Producer (Mandatory)

Fee (Ether)

Max. 0.0369062574 Min. 0.0369062574 Av. 0.0369062574

Execution time

Max. 239964s Min. 239964s Av. 239964s

Gas used

Max. 4262812 Min. 4262812 Av. 4262812

Instances

✓ 6349716184a1ec1940858591

Execution time: 239964s

Total gas used: 4262812

Total fee: 0.0369062574 Ether

Your roles

Retailer (Mandatory)

634d24b71e19f61d5cb6f19e

Execution time: 0s

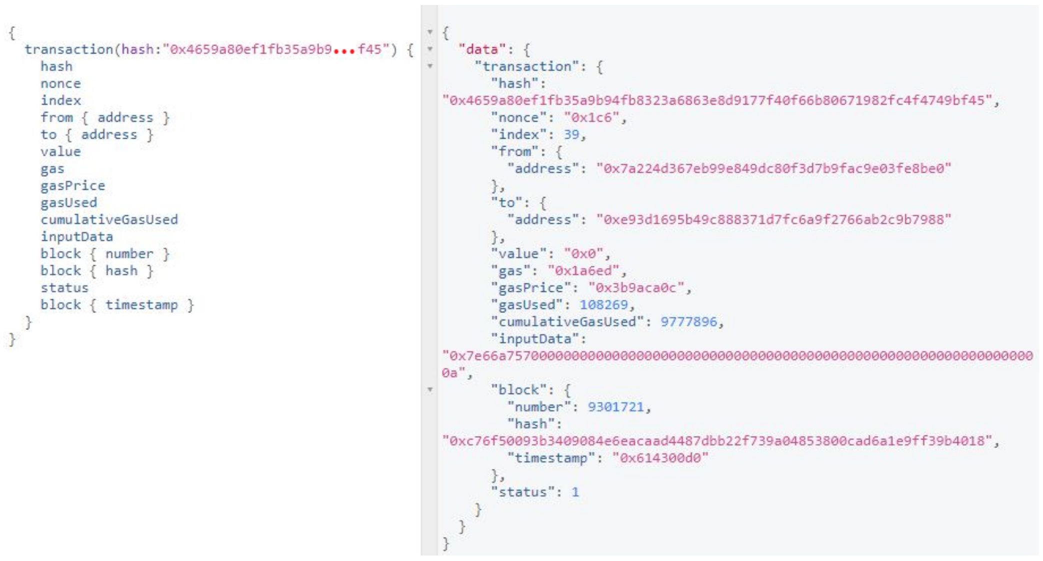
Total gas used: 3505790

Total fee: 0.0350579000 Ether

Your roles

Producer (Mandatory)

Figure 4.8: User-based auditing page.



```
{
  transaction(hash:"0x4659a80ef1fb35a9b9...f45") {
    hash
    nonce
    index
    from { address }
    to { address }
    value
    gas
    gasPrice
    gasUsed
    cumulativeGasUsed
    inputData
    block { number }
    block { hash }
    status
    block { timestamp }
  }
}
```

```
{
  "data": {
    "transaction": {
      "hash":
"0x4659a80ef1fb35a9b94fb8323a6863e8d9177f40f66b80671982fc4f4749bf45",
      "nonce": "0x1c6",
      "index": 39,
      "from": {
        "address": "0x7a224d367eb99e849dc80f3d7b9fac9e03fe8be0"
      },
      "to": {
        "address": "0xe93d1695b49c888371d7fc6a9f2766ab2c9b7988"
      },
      "value": "0x0",
      "gas": "0x1a6ed",
      "gasPrice": "0x3b9aca0c",
      "gasUsed": 108269,
      "cumulativeGasUsed": 9777896,
      "inputData":
"0x7e66a75700000000000000000000000000000000000000000000000000000000000000
0a",
      "block": {
        "number": 9301721,
        "hash":
"0xc76f50093b3409084e6eacaad4487dbb22f739a04853800cad6a1e9ff39b4018",
        "timestamp": "0x614300d0"
      },
      "status": 1
    }
  }
}
```

Figure 4.9: GraphQL query example on a single transaction.

It is worth noticing that GraphQL queries are also used in the implementation of choreography-based and user-based auditing strategies. Indeed, to retrieve the necessary information, ChorChain executes a set of automatic queries and combines the corresponding results in a browsable structure. In particular, the queries target a blockchain node to get the selected parameters in a certain transaction or block. The resulting output contains then all the data usually visible in other online tools such as Remix IDE[1] or a block explorer. However, using such instruments requires the end user to have a certain level of expertise and to perform a lot of manual operations to get every single piece of data. With the ChorChain auditing instead, it is possible to drastically reduce the manual effort and the technical skill required thanks to the automatic query mechanism that also extracts the entire set of blockchain data, increasing the overall performance.

Considering the provided auditing mechanism, once a model is selected, the automatic queries executed by ChorChain can be synthesised as follows:

1. firstly, all the choreography contracts generated from the model are retrieved;

2. for each transaction in the contract, a query is executed for retrieving information.

Finally, the results are parsed and shown to the user via the ChorChain front-end. Notably, all the inputs used in the described process, like the smart contract addresses and the transaction hashes, are automatically injected in the queries by ChorChain. This provides high-level and user-friendly functionalities.

Retail process example (Auditing) To clarify how the ChorChain audit mechanisms can be used, and which information can be retrieved from the blockchain, the retail process scenario is presented.

The first case concerns the information related to the contracts generated from the retail choreography model and they are reported in Table 4.5. In particular, the information available in this table refers to: the number of instances, with the relative hashes; the percentage of completed instances; the list of users, and the number of times they participated. Similar information is available for each message. Finally, it reports the minimum, maximum, and average measures of completed contracts with respect to the required time and the used gas.

In case a single choreography contract is selected, a set of information is visible as listed in Table 4.6. The reported example is related to the retail process choreography, in which the contract contains: the contract address;

[1] `https://remix.ethereum.org/`

Table 4.5: Rental process details for the Choreography entity.

Choreography	**Values**
list of generated choreography contracts	ID: 6349716184a1ec1940858591 ID: 6349716184a1ec1940858591 ...
number of generated choreography contracts	5
number and percentage of completed choreography contracts	3 (60%)
all users for each participant	Retailer: 0x7A224d367EB99e... Customer: 0xaeD0aBbD8C55ca...
cardinality of the users for each participant	Retailer: 0x7A224d367EB99e... (x4) Customer: 0xaeD0aBbD8C55ca... (x1) ...
all transactions for each message	0x100831606e0d1f59b37c... Instance: 6349716184a1ec1940858591 ...
cardinality of the transactions for each message	request_quotation(string good, unit amount) (Sent 4 times) ...
all users involved for each message	Message: request_quotation(string good, unit amount) Transaction: 0x100831606e0d1f59b37c... User: 0xaed0abbd8c55ca...
min, max and avg execution time calculated on all completed choreography contracts	Min. 2.028s Max. 239.964s Avg. 82.952s
min, max and avg used gas calculated on all the completed choreography contracts	Min. 4,262,488 Max. 4,439,142 Avg. 4,321,480

the costs in terms of Ether afforded by each user involved in the collaboration; the list of emitted transactions with the respective messages; and the users involved in each existing role.

Once the choreography contract has been selected, the auditor can investigate the information related to a specific transaction. In Table 4.7, the available information concerns the timestamp of the transaction, with the respective sender and payload, and all data related to the consumed gas and paid fee.

Finally, when the information retrieving starts from the User entity, Table 4.8 provides the example of the available view in the deployed retail process. In such a case, the retrieved information concerns the list of models in which the user was involved with the respective role. Furthermore, additional information refers to the percentage of completed instances by the user, and the statistics related to the minimum, maximum and average execution time and used gas.

4.3 Experiments and Validation

This section reports the experiments conducted on the extended ChorChain tool and on the auditing strategies. Initially, an evaluation of auditing performances is made, while in the last part of the section, an overall assessment

Table 4.6: Retail process details for the Choreography contract entity.

Choreography Contract	**Values**
blockchain contract address	0x959ef4ba53f108396e22efe5cdd61c4c14a7f628
total cost, in fees and gas	0.0369062574 Ether - 4.262.812
cumulative cost/percentage in fees and gas for each user	0.0005497024 Ether (1.4%) - 324137 (7,6%)
emitted transactions	Transaction: 0xf64cc71f1db1... ...
completion status	Complete
involved users and their role	Retailer (mandatory) Customer (mandatory) Producer(optional)
transaction(s) generated for each message	request_quotation(string good, unit amount) (Sent 1 time) Transaction: 0xf64cc71f1db18734036...

Table 4.7: Retail process details for the Transaction entity.

Transaction	**Values**
timestamp	17/10/2022 11:03:36
user sender	0xaed0abbd8c55caf1247ed157c5b7c7bb4f358354
gas consumed	126,873
fee paid	0.0003806190 Ether
message payload	Message_0b917rc(good: goodToPurchase, amount: 10)

of the complete ChorChain framework is done.

4.3.1 Running Example Performance Analysis

Here the results of some experiments are discussed aiming at assessing the performance and scalability of the ChorChain auditing facilities. In particular, here is measured the time requested for retrieving information from the blockchain by means of the query operations delineated in Figure 4.7. For this scope, the Sepolia Testnet blockchain was used along with the GraphQL endpoint hosted in the node within our lab. It is worth noticing that the experiment results are influenced by the network conditions, the node congestion, and the number of transactions contained in each choreography contract to query.

To measure the scalability of auditing with respect to the increasing number of contracts, five different contracts were created from the Retail choreography and executed following the shortest path. In particular, at the end of each execution, the audit page was launched ten times and the final result corresponds to the average time for loading the page. Figure 4.10 displays the resulting graph showing an evident increment of the response time with respect to the number of existing choreography contracts. Indeed, 1.039 seconds are requested with just one contract, while 4,399.1 seconds are requested for the same choreography with five contracts. The graph depicts

Table 4.8: Retail process details for the User entity.

User	**Values**
list of choreography models in which the user was involved	RetailProcess
all the covered participant roles	Retailer
the participant role covered in each choreography contract	6349716184a1ec1940858591 : Retailer
number and percentage of completed choreography contracts	Instances: 5 - completed: 3 (60%)
min, max and avg execution time for all completed choreography contracts	Min. 2.028s - Av. 82.952s - Max. 239.964s
min, max and avg gas used for all completed choreography contracts	Min. 4,262,488 Max. 4,439,142 Avg. 4,321,480

a linear growth, representing a constant time requested for accessing each single choreography instance. Notably, for each newly completed contract, the auditing test is re-executed and it is comprehensive also of the contracts already created and executed.

4.3.2 Assessment with the Involvement of Practitioners

Here are reported the results obtained from those experiments devised to assess the effectiveness of the approach and the tool. The objectives of the experiment, indeed, are to establish if the proposed approach can actually help the engineering of trustable and auditable systems and to verify whether the auditing tools provide easy access to the necessary information.

Experiment Set-Up The experiment involved 12 students enrolled at the University of Camerino at the 1st year of the MSc in Computer Science (Information Systems Engineering curriculum). MSc students cannot be certainly considered experts in choreography modelling and blockchain-based systems; however, they can be neither considered novices in the discipline: all of them have got a BSc degree in Computer Science and have taken two semestral courses at master level about business process modelling as well as one semestral course about enterprise software infrastructure. Thus, the students selected for the experiments can be considered knowledgeable about modelling practices and process automation platforms, even though not really experts. In addition, as suggested by other studies [53, 80], the involvement of students for the kind of experiments is considered rather effective, as students are not biased by prior practical knowledge and experience, which could influence the final results.

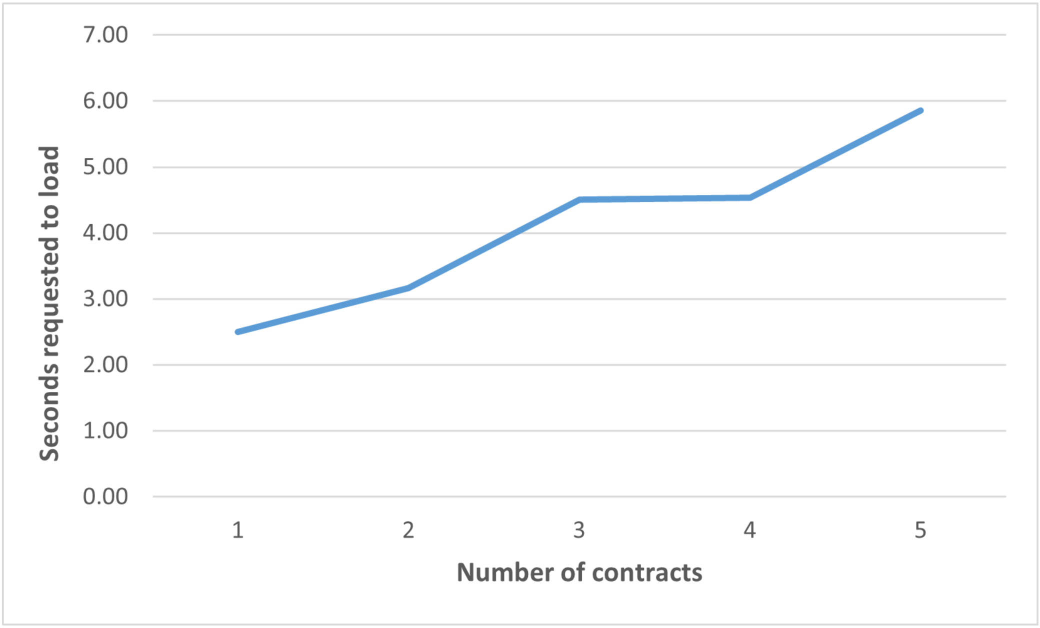


Figure 4.10: Scalability of ChorChain auditing.

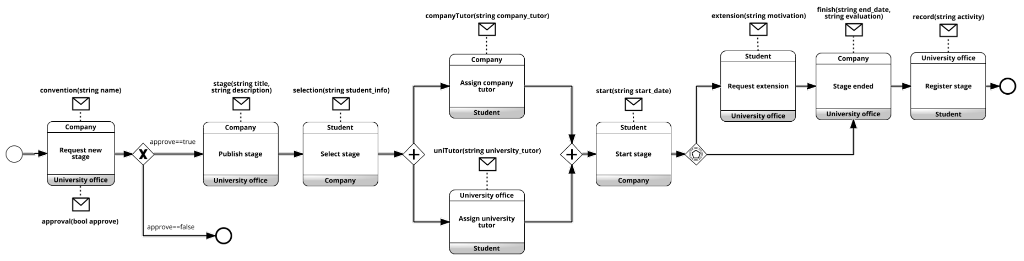


Figure 4.11: Internship choreography .

In the experiments, the students had to use the ChorChain framework focusing on two different parts of the methodology, i.e., (i) subscription, instantiation, execution and (ii) auditing. To this aim, a common scenario referring to the internship process flow depicted in Figure 4.11 was provided. It shows a company, a university office and a student interacting in order to publish, select, assign and complete a given internship. According to the selected scenario, the students were divided into groups of three forming a total of 4 groups. Each member of a group was asked to subscribe to a participant role in the choreography and start to interact with the ChorChain interface in order to interact with the choreography execution. In the end, the members of each group were asked to use the auditing functionalities to check the transactions performed by the other members of the group, as well as the executions of the other groups. The activity took one hour and a half and involved the members of the groups executing the same scenario and playing different roles.

At the end of the activity, the students filled out the questionnaires reported in the Appendix, in order to judge the ChorChain tool in relation to its usability and perceived usefulness. The questionnaire has been conceived according to the guidelines provided in [24]. For each of the considered questions, the student could mark just one box, that would best describe the experience with the usage of ChorChain. The students separately marked the auditing feature to have a more precise view of its maturity with respect to the subscription, instantiation and execution features. To evaluate the results, the System Usability Scale (SUS) was used, since it "provides a quick and dirty reliable approach for measuring usability" [18].

Experiment Results After the students ended the experiment, 11 completed instances of the internship choreography were obtained. To assess the effectiveness of the approach and the tool, the results of the questionnaires filled out by the students were considered. Tables 4.9 and 4.10 report the answers provided by the students with reference to the subscription, instantiation and execution phases and auditing phases, respectively. The strategy used in the questionnaires alternates questions for which the "positive" answer is somehow inverted. In particular, for questions 1, 3, 5, 7 and 9 the greater the number the better, while for questions 2, 4, 6, 8 and 10 the smaller the number the better. This technique somehow tries to avoid answers provided in a superficial way. Once all administered questionnaires have been filled, the following formula has been computed:

$$\mathcal{S} = \frac{\sum\limits_{i=1}^{N}(\sum\limits_{j=1}^{5}(5 - res_{i,2j}) + \sum\limits_{j=0}^{4}(res_{i,2j+1} - 1))}{N} \times 2.5$$

It provides a way to get an overall score ($\mathcal{S}$) that can be used to globally assess the experiment results. In the formula, N represents the number of returned questionnaires, while $res_{i,j}$ is the response provided to the questionnaire by each participant (i) to every single question (j). The formula at first conducts all the answers to the scale $0-4$, where positive answers now always correspond to higher values. Then, for each questionnaire the total sum is computed, getting a number between 0 and 40, and the average over all the questionnaires is derived. Finally, the number is multiplied by 2.5 to get a final score $\mathcal{S}$ in the range $0-100$. For the experiment, the calculation of the SUS score gives the value of 68.54 to the subscription, instantiation and execution phases and 66.25 to the auditing phase. According to the proponents of the SUS approach, this is somehow a good result in particular for what concerns subscription, instantiation, execution. Indeed, in their experience, values for $\mathcal{S}$ greater than 68 relate to perceived usability somehow better with respect to other used software. Being the involved users somehow experienced with process automation, this suggests that overall they got a relatively positive experience in using ChorChain. It is worth noticing that the usability test related to the auditing phase performed worst than the subscription, instantiation, execution phases; this suggests the need of investing in this part of our solution, to make it even more usable to the users.

Going more in detail, Tables 4.11 and 4.12 report the distribution of the evaluation over the range 0–4 for each question. Considering the subscription, instantiation and execution phases, it is possible to observe that the best evaluation was obtained by question 5 (" *I found the various functions in* ChorChain *were well integrated*") that got an average score of 3.0, while the worst evaluation was reported by question 4 ("*I think that I would need assistance to be able to use* ChorChain.") and question 10 ("*I needed to learn many things before I could get going with* ChorChain") with an average score of 2.15. Considering the auditing phase, instead, the best evaluation was obtained by question 5 (" *I found the various functions in* ChorChain *were well integrated*") that got an average score of 2.77, while the worst evaluation was reported by question 4 ("*I think that I would need assistance to be able to use* ChorChain") with an average score of 2.08. The two questionnaires show a similar trend. On the positive side, the tool seems well integrated; instead, on the negative side, it seems to require some background knowledge to be used in practice.

	Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**
Student 1	3	1	5	1	4	1	5	1	5	1
Student 2	5	3	4	3	5	2	5	3	4	5
Student 3	4	3	5	2	4	2	4	2	4	2
Student 4	3	1	3	2	4	2	5	3	4	2
Student 5	4	1	4	3	5	2	5	1	5	2
Student 6	5	2	4	3	4	2	4	3	3	2
Student 7	4	3	4	4	4	3	3	3	3	4
Student 8	4	3	2	3	4	3	2	4	3	2
Student 9	4	2	4	2	4	2	4	2	4	4
Student 10	4	1	4	2	4	2	3	3	4	4
Student 11	5	3	2	4	5	2	2	3	4	2
Student 12	4	3	3	3	4	3	4	3	3	2

Table 4.9: Results from the questionnaires on usability related to the subscription, instantiation and execution phases.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Student 1	3	2	4	3	4	2	4	3	4	2
Student 2	4	2	2	4	4	2	4	3	3	3
Student 3	4	2	4	2	4	2	4	3	3	3
Student 4	4	1	3	2	4	2	5	3	4	2
Student 5	5	1	5	2	4	1	4	2	4	1
Student 6	4	3	3	3	4	2	3	2	3	2
Student 7	4	3	4	3	5	3	3	3	3	4
Student 8	4	3	2	3	3	2	3	2	3	2
Student 9	4	2	4	2	4	2	4	2	4	3
Student 10	4	2	4	3	4	3	3	3	3	3
Student 11	5	2	2	3	4	2	3	3	4	1
Student 12	4	2	3	3	4	3	4	3	3	2

Table 4.10: Results from the questionnaires on usability on the auditing functionality.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	0	0	2	1	0	3
2	2	6	2	5	0	3	2	7	4	0
3	7	2	6	4	9	8	4	2	6	7
4	3	4	2	1	3	1	4	2	2	1
Avg	2.85	2.62	2.46	2.15	3.00	2.62	2.62	2.23	2.62	2.15

Table 4.11: Distribution of scores for each question related to the subscription, instantiation and execution phases (over the range 0–4).

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
0	0	0	0	0	0	0	0	0	0	0
1	0	0	3	1	0	0	0	0	0	1
2	1	3	3	7	1	3	5	8	7	4
3	9	7	5	4	10	8	6	4	5	5
4	2	2	1	0	1	1	1	0	0	2
Avg	2.85	2.69	2.15	2.08	2.77	2.62	2.46	2.15	2.23	2.46

Table 4.12: Distribution of scores for each question on the auditing functionality (over the range 0–4).

4.4 Comparison with Existing Approaches

Blockchain conveniently enables auditing activities, since all operations registered on the blockchain are verifiable. Blockchain technology represents a new opportunity for Computer-Assisted Audit Tools and techniques, referring to software supporting auditors in the audit process. In particular, blockchain is recognised as useful to reduce the workload of the auditors and to reduce frauds [1]. Indeed, it guarantees secure and immutable information storage thanks to the possible verification of the executed transactions. The literature also discusses how this new technology could be affected by GDPR; in particular, in [92] it is highlighted the impact of this European regulation in data sharing environments.

Many works focus on enabling auditing by exploiting process mining algorithms applied directly to transactions stored in the blockchain [37, 45, 63, 84]. They show how this kind of procedure is not trivial, and some retrieved information could be incomplete or very difficult to aggregate. However, the main objective is not the application of process mining techniques on data stored in the blockchain, but the definition of an integrated approach able to support the entire life-cycle of choreographies, considering the possibility to examine and evaluate statements usually happened in the collaboration, providing also information relative to quantitative indicators like costs and times. In the auditing part, ChorChain is able to retrieve all the detailed information related to the exchange of messages between organisations through a scan of the events generated by the smart contracts. The parsed information will be shown to the auditors in an aggregated way, permitting the auditors to navigate between the details. Moreover, the choreography-centric approach in the auditing phase permits to have a one-to-one mapping of information with the related choreography instances, guaranteeing always a good level of details in an efficient way.

In [64], the blockchain is recognised as an improvement of the auditing system, referring to continuous auditing and continuous monitoring technologies. In particular, continuous process auditing is the constant monitoring and analysis of systems, that improves the focus and the scope of the auditors [115]. In this context, the authors focus on accounting and assurance functions, these could indeed benefit from a new blockchain approach.

In [11], the auditors are seen as beneficiaries of the blockchain in order to verify records in accounting systems. This is relevant both for internal and external auditors, interacting with accounting records. The authors expose also issues of actual audit systems that are addressed by blockchain: data reliability, data security and transaction transparency. Data must be reliable in order to be trusted; blockchain could guarantee reliable evidence thanks to its structure. Data security refers to the assurance that the information has not been corrupted; thanks to cryptographic algorithms blockchain makes data secure and immutable. Finally, transaction transparency is required in order to have observable and verifiable data; again, blockchain has native properties of transparency suitable for this requirement. ChorChain leverages these properties of blockchain in order to provide its audit functionalities.

In addition to verification, tamper-proof and security properties, pseudo-real-time is highlighted in [96] and [42]. In particular, this last work reconsiders accounting environments, suggesting a permissioned blockchain. This could indeed ensure an authorisation layer that prevents companies from external attackers while the auditing process remains visible to auditors. Another opportunity is seen in smart contracts that can contain policies and assertions, enabling a continuous and automatic execution of smart controls. Both works expose also possible issues in the use of blockchain. On the one hand, the permanent loss of private key, the impossibility of reverting a transaction, and the lack of a central administration that can be contacted [96]. On the other hand, the scalability of blockchain when a lot of data is produced, the investments required to create a new infrastructure and the instruction needed by the internal department for acquiring blockchain knowledge [42].

An approach for auditing based on the client-server model is proposed in [109]. Here the blockchain is used to audit the exchanged messages without relying on a client and/or server for storing data. Thanks to its properties, blockchain is used as a message-storing system allowing secure and distributed auditing to auditors that can access the network. Every time a message is exchanged, a new transaction is generated and then stored in the blockchain. This makes it possible for auditors to see and verify data. The authors propose also a scheme with blockchain and message requirements, evaluating also that scheme with experiments relying on the Bitcoin technology. However, the authors show how the time for a message exchange is higher than a normal interaction between a client and a server. Another problem is the cost of fees in the Bitcoin blockchain. In this thesis, a similar use of blockchain is made to store the messages exchanged by the participants of a choreography. However, the use of a different blockchain in ChorChain allows better performance in terms of time and cost. Indeed, in Ethereum, the time for a block inclusion is much lower (around 15 seconds) and fees are not so expensive.

Bitcoin is also used in [108] where the authors propose a solution for tamper-proof privacy audit logs. In a collaborative environment, it is indeed important to have compliance with privacy policies when personal information is shared. In actual systems, where a central authority is present, auditors may not rely on data that, in addition, can also be corrupted. In this context, blockchain technology is used to guarantee distributed and immutable storage. As stated by the authors, Bitcoin and the proposed solution is limited considering costs, speed, and scalability. The authors indicate Ethereum as possible evolution in terms of fee, scalability, and smart contracts. This analysis reinforces the choice of using Ethereum as blockchain technology underlying ChorChain.

In [3] the authors consider a client-server model to build a secure auditing system based on the Hyperledger technology. This model is analysed under security aspects, showing potential attacks and issues with the actual way of maintaining audit logs. The presented BlockAudit application is implemented over the permissioned Hyperledger Fabric blockchain, which guarantees a secure logging activity. This reduces the risks of potential attacks due to security vulnerabilities of current database systems. Differently from this work, in ChorChain the transactions generated from participants' activities are performed directly in the blockchain, without passing from the database. This ensures a high level of security in all the information and operations of the process, eliminating risks of attacks on the database layer.

A permissioned structure is used also in [4], where a multi-chain solution, called BlockTrail, is implemented starting from Hyperledger. The blockchain structure guarantees secure storage of audit logs while the multi-layering allows a throughput increment, reducing costs and efficiency of the audit activity. This is obtained with replicas and partitioning of peers and networks, creating a hierarchical infrastructure. Similarly, the CertChain certificate management system, presented in [31], uses the blockchain, in this case, Ethereum, to audit and store certificates for TLS connections that have still security issues. Anyway, even if there are similarities between the exploited blockchain technology and the possibility of making auditing on the stored data, this latter solution has a quite different goal with respect to ChorChain: CertChain aims at managing TLS certificates while ChorChain offers full support to the choreography life-cycle, including the auditing phase.

What mainly differentiates ChorChain's auditing phase from the above approaches is how the users of the framework are supported. Indeed, the ChorChain framework provides the audit functionalities without requiring complicated actions by the users. As usual, all information exchanged by participants is stored in the blockchain, but the novelty is in the audit interface. Indeed, it is designed to retrieve automatically all the exchanged messages from the blockchain, they are then elaborated, showing all the key points useful to audit the process. This allows taking advantage of the framework without the need for additional actions, exploiting all the built-in functionalities in order to obtain evidence of the interactions between participants and to analyse them.

CHAPTER 5

FLEXCHAIN: SUPPORTING RUN-TIME FLEXIBILITY

In this chapter, the thesis focuses on the flexibility challenge derived from the immutability of the blockchain. Indeed, while the blockchain permits achieving trustworthiness enforcing the interactions and providing transparent and secure proof of past events, at the same time it hinders the flexibility of the business process execution [105]. In this thesis, the term flexibility indicates the need for business processes that are not static, and that instead embed mechanisms permitting to react to changes occurring at run-time. In particular, this thesis refers to *flexibility by change*, which is the ability to modify a business process definition at run-time and migrate the current execution to the newly uploaded definition [33]. Flexibility is a relevant property for aligning business processes [114, 34, 97]. Many solutions supporting business process flexibility have been proposed for traditional (not based on blockchain) implementations like those surveyed in [79]. However, flexibility is still an open challenge when blockchain technology supports the process execution [81].

To face this challenge, the thesis proposes FLEXCHAIN, a framework for supporting run-time flexibility in blockchain-based business processes. To overcome the immutability of blockchain code, the framework decouples the business logic of the process from its execution state. In this way, it is possible to guarantee run-time changes to the process execution without losing the fundamental properties of trust provided by the blockchain.

In the following sections, the thesis introduces the FLEXCHAIN framework, describing also the theoretical translation approach producing rules starting from choreography diagrams. Then the FLEXCHAIN architecture is presented together with the implemented tool and its validation. The

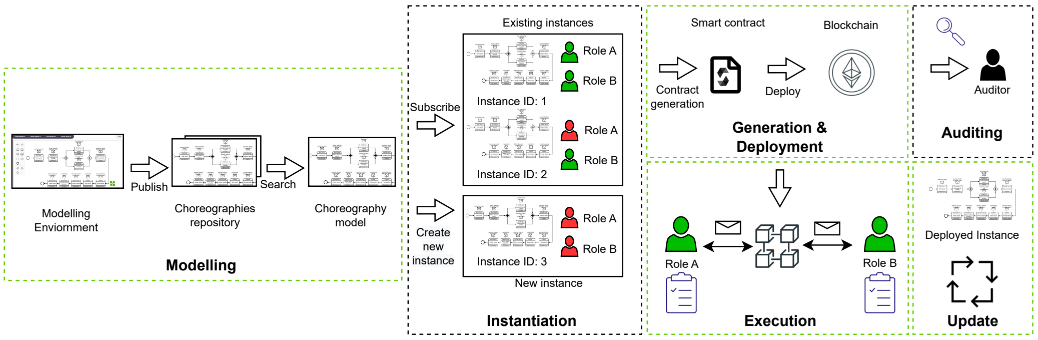


Figure 5.1: FlexChain framework: supporting run-time flexibility.

concluding section provides a comparison with already existing approaches.

5.1 FlexChain Conceptual Framework

The FlexChain framework aims at guaranteeing the flexible execution of smart contracts following a model-driven technique. The flexibility aspect is indeed an attribute influencing in a direct way the perception of trust in a collaborative scenario [83] as it permits a service to behave in an acceptable way in anomalous or unexpected situations or when the context changes.

5.1.1 Framework phases

The framework is reported in Figure 5.1 and it revises some aspects of ChorChain for generating an infrastructure based on a public blockchain, starting from BPMN Choreography diagrams, representing the business interactions between parties. In particular, FlexChain focuses on the modelling, generation, deployment and execution of a choreography, introducing also a new update phase.

The **instantiation** is the first phase and it starts with the user uploading a choreography model using a dedicated interface. This model is then automatically translated into an on-chain smart contract and into a set of rules. The contract is deployed through the use of the factory contract and represents the choreography instance state. The rules instead are stored in the InterPlanetary File System (IPFS) and they represent the execution logic of the process. In particular, they specify when a message can be sent, under which conditions and which result the execution produces. The IPFS[1]

[1]The IPFS [19, 43, 112, 131] is a distributed system for storing different types of data. It is based on a peer-to-peer network where each peer stores some information accessible from anywhere. Through the hashing of the information, IPFS binds data with

is used in FlexChain to store voluminous information avoiding reporting them in the blockchain. In such a way it is possible to reduce the cost to store information [131], especially those related to the messages exchanged by the participants in a choreography.

Rules are automatically derived by a translation performed on the messages of the model and they are executed by an off-chain processor. To notice, the decoupling of the state from its logic is an important aspect of the proposed approach, it allows the use of blockchain immutability for storing business information, while the logic and its constraints can be maintained off-chain. Thanks to this solution, it is possible to mix the flexibility and the immutability of the code in blockchain technology.

The **update** phase is similar to the previous one and it consists of the willingness of a participant to perform changes to a running process instance. In particular, in FlexChain, changes are isolated and affect only the running instance to which they are applied. Indeed, the motivation behind a run-time change can not be assumed to be valid for all the current active instances, as they may involve different participants with different needs and contexts. It is worth noting that the update of the rules does not affect the current instance state stored on-chain, permitting the new rules to continue operating in the current state. The willingness for a change in the process instance is expressed with the proposal of an updated model generating only a new set of translated rules derived from it.Before being part of the effective execution, these rules are stored in a new IPFS location and the hash is stored on-chain in the running smart contract instance. At this point, the participants can use the instance contract for voting on the update. In case the quorum[2] is reached, the smart contract automatically replaces the old hash with the one just voted. In this way, the smart contract code contains only the current state of the process avoiding the need to modify directly it. The use of an on-chain quorum strategy comes from the need of preventing malicious behaviours that could lead to an uncontrolled update of the model. Indeed, all the participants need to vote and approve the proposal, giving explicit feedback about it. In case a malicious update is proposed, the other participants can protect themselves by voting against the update and the smart contract will automatically stop the procedure. In this way, for an attacker would not be possible to propagate faulty behaviours in the system.

a unique Content Identifier (CID) formed by the hash of the content itself. It uses the Interplanetary Linked Data ecosystem for formatting and storing data across the network. The data structure used corresponds to Merkle Directed Acyclic Graphs where each piece of information stored through the IPFS is split inside blocks. This makes the data versioning more efficient since it allows to update only part of the content, by uploading the new data and receiving back the new CID.

[2]The quorum refers to the minimum number of choreography participants that need to accept and commit an updated proposal.

The last phase regards the **execution** of a message and the consequent state update. In this case, to start the execution, the user needs to select through the FlexChain interface the message to send to the counterpart inserting also the required parameters. The interface invokes then the smart contract instance passing this information. The contract will successively emit an event that the off-chain processor will catch about the operation to perform. This retrieves the corresponding rule from the IPFS and after executing it will return the result to the smart contract.

5.1.2 Translation approach: BPMN to Drools

This Section described the conceptual approach to derive Drools rules starting from a BPMN choreography diagram. Indeed, while the FlexChain framework exploits the blockchain for regulating the interactions, the used smart contracts rely on a fixed template that does not directly depends on the initial model. Indeed, only rules are translated accordingly to the various choreography elements compliant with the BPMN standard as reported in Figure 2.2, which correspond to the most used elements in choreography diagrams [35]. Hence, no further additions to the input language are required for dealing with common application scenarios. If a new element is needed, the algorithm should be modified to include it.

In general, a rule is divided into two main constructs. In the first part, the conditions to be evaluated are expressed. Conditions are internally verified and they are used to regulate the activation and execution of message rules. The second part of the rule manages instead the update of the state variables inside the smart contract. As in ChorChain, state variables are derived by message inputs parsed from the message name.

The FlexChain translator generates rules from a BPMN choreography diagram according to the following translation algorithm and to Table 5.1. The algorithm iterates over the list of all choreography messages and, for each of them, a new rule is created. For rule generation, the algorithm considers the paths leading to the corresponding message. Indeed, for each element in these paths, a condition is added inside the rule. If the element is an exclusive gateway, the related conditions are extracted from the sequence flow of the element, and added to the rule. If instead, the element is a message, a check on the completion of this is added inside the rule and the exploration of that path stops. Finally, after all the incoming paths connected to the considered message are explored, the code for pushing state variables to the smart contract state is added inside the second part of the rule.

The algorithm translates messages (case 1) as they require direct enforcement and regulation since during the execution they actively modify the current state due to the participants' actions. Notice, as in ChorChain, the two-way task can be represented as two one-way tasks in sequence and each connected message is treated independently. Therefore, in FlexChain the two-way task is translated as two separate rules, one for each message. Similar to ChorChain, an execution state is attached to each message and its related rule so as to enforce the right execution sequence. A message element also influences the condition to evaluate before execution. This happens with a double check both on each previously connected message (which state has to be completed) and on the current translated one (which state has to be enabled).

Control flow elements such as gateways and intermediate events are instead translated only as conditions to be verified internally to message rules. This makes it possible to check whenever a decision derived by a gateway must be taken. Notice, the start and end events are not considered as they only define the starting and ending points of the choreography without actively influencing the execution. For this reason, these behaviours are transferred to the initial and ending messages, without explicitly considering the start and the end. In the case of the exclusive gateway, this produces a condition based on a boolean expression (case 3, split) retrieved from the outgoing sequence flow (case 2) or an exclusive check on the previously connected elements (case 3, join). A parallel gateway (case 4) only produces a check on the previously connected message while an event-based gateway (case 5) only defines which message is the first to be executed, blocking the others. The translation patterns and the exact translation for each element are presented with the relative examples in Section 5.2.2.

To sum up, the translation of the choreography model encodes in the generated rules the model's control flow, thus providing the choreography execution with an enforcing mechanism. In particular, to prevent unexpected execution, each rule checks the status of the involved elements according to the state variables stored in the smart contract. Moreover, the data exchanged during the execution is stored in the blockchain and can be audited by all the participants.

5.2 FlexChain tool

In this section, the thesis describes the translation approach and the FlexChain tool by providing different examples of the generated rules and the different phases of the tool[3].

[3]The tool is available at `virtualpros.unicam.it:3000` while its code, as well as the smart contracts, is accessible at `https://bitbucket.org/proslabteam/flexchain_v2/src/master/`.

Table 5.1: Translation approach from BPMN elements to a Drools rule.

BPMN element	Drools code	BPMN element	Drools code
1) Message	• Rule • Previous message condition • Actual message condition	4) Parallel Gateway	• Inclusive message state check
2) Sequence Flow	• Guard expression	5) Event-Based Gateway	• Race condition message state check
3) Exclusive Gateway	• Expression condition • Exclusive message state check		

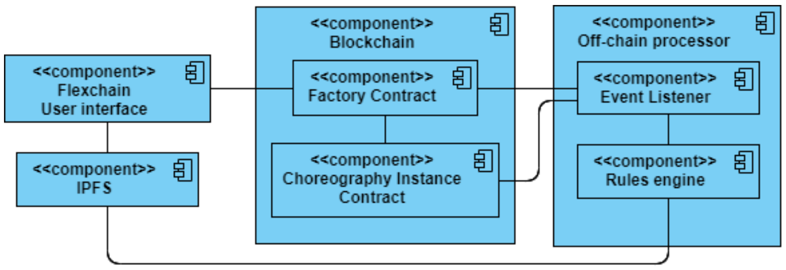

<<component>> Flexchain User interface
<<component>> IPFS
<<component>> Blockchain
<<component>> Factory Contract
<<component>> Choreography Instance Contract
<<component>> Off-chain processor
<<component>> Event Listener
<<component>> Rules engine

Figure 5.2: Component diagram of the FLEXCHAIN architecture.

5.2.1 Architecture

The FLEXCHAIN architecture is reported in Figure 5.2 and is divided into four main components based on the patterns proposed in [126]: i) User interface, ii) Blockchain, iii) Off-chain processor, and iv) IPFS.

The **user interface** provides all the functionalities used for creating a choreography diagram and interacting with the blockchain. It exposes the main operations like the instantiation, the update, and the execution. Thanks to this component and the model-driven engineering methodology adopted, the end user does not need any technical knowledge about blockchain and its implementation.

The **blockchain** component is structured by two main smart contracts used by FLEXCHAIN to regulate the on-chain behaviour. The first is the *Factory contract* that is invoked by the FLEXCHAIN User Interface every time a choreography model is uploaded and a new choreography instance must be created. This acts also as a register since it maintains in memory the name of the choreography instance and its related address. The second is the *Instance contract* and it is used as a template for instantiating a model. The code is composed of a set of state variables, that are used to manage the process and some utility functions for the different operations. In particular, FlexChain relies on a single Factory contract that deploys an Instance contract for each model instance generated by the user. This architecture provides a general and reusable infrastructure dealing with flexibility requirements. The full code of the factory and instance smart contracts is available here[4].

In such architecture, flexibility is achieved by implementing the execution logic of the choreography, using Drools rules. The rules are generated automatically by the FlexChain framework and they are then deployed on the **IPFS** repository. The resulting hash is successively stored inside the on-chain choreography instance smart contract [5] to have a certified reference of the rules during the execution. The benefit of using a rule-based technique comes from the possibility of having a direct representation of a choreography message as a rule. This allows having the execution logic of the choreography implemented as a set of atomic rules that are executed independently from the others. This, combined with the off-chain storage, allows overcoming the restrictions of the blockchain since in this way it is possible to update at runtime only the targeted rules, without dealing with the immutability of smart contracts. The other advantage of storing rules inside the IPFS regards the costs. Indeed, with this approach, for each update in the choreography, the smart contract has to change only the reference to the final IPFS hash. This reduces costs with respect to storing the whole set of rules inside the smart contract on-chain.

The **off-chain processor** is used instead to execute rules. Its behaviour is triggered directly by an event emitted by the choreography instance smart contract and caught by the *event listener* component. This component exploits the Ethereum API and waits for every event emitted by the smart contracts. In particular, its main role is to listen for message execution events, invoking then the Rules engine component by forwarding the necessary data. The event contains the necessary information regarding the rule to execute on the *rule engine* and the relative parameters. Notice, the off-chain processor at each execution always checks the latest version of the rule present in the IPFS according to the registered id stored in the choreography smart contract. The use of an off-chain practice is nowadays widely common [58, 85], in particular when combined with the blockchain to address storage and performance limitations. In these situations, the trust the blockchain provides derives from the capability to run auditing sessions on the resulting execution traces possibly. Indeed, in our case the smart contracts trigger the execution of the off-chain rules, receiving back the final status. All this

[4]`https://bitbucket.org/proslabteam/flexchain_v2/src/master/Smart%20Contracts/`

[5]For this reason, in Figure 5.2 there is no direct connection between IPFS and Blockchain.

information is stored on the blockchain, thus making it visible and verifiable by every participant providing a balanced approach in terms of performance and trustworthiness. To facilitate user interactions, the FlexChain tool also provides a graphical user interface that takes in input information from the end user and forwards them to the smart contract. From there, the smart contract and the off-chain processor automatically carry out all subsequent steps connected to the message reception. This kind of interaction is, indeed, the one considered in the case study in Section 2.2.2, where the participants of the choreography are humans. However, the framework can also support a fully-automatic form of interaction (via, e.g., REST services). In this case, the FlexChain user interface can be bypassed, and messages can be automatically executed by connecting an external application to the smart contracts.

5.2.2 Translation

This part reports the details for the generation of the rules and the smart contracts starting from a choreography as described in Section 5.1.2. Similarly to ChorChain, the state of a message can be *enabled*, *disabled*, or *completed* and depending on it, only a certain action can be performed. For example, if a message is completed it cannot be executed again, and its corresponding rule will fail if invoked. Notice that, in case of a loop in the model, the same message can be re-activated many times, in order to re-execute it.

Examples of Translation Results To clarify how the translation algorithm operates, the following lines describe some examples resulting from applying the algorithm to typical excerpts of BPMN choreography models. For each element, the corresponding pseudo-code of the rule is shown and discussed highlighting the main peculiarities.

The **one-way task** is shown in Figure 5.3 and its translation generates the rule reported in Listing 5.1. The general formatting is composed of the rule name (Line 104), corresponding to a unique message-id, the *When* part (Line 105), containing the conditions to evaluate, and the *Then* part (Line 109), defining the actions to perform. Inside the *When* clause, different types of conditions are verified. Firstly, the states of the previous connected messages are checked (Line 106). Then, the rule checks the current element state, which must be enabled (Line 107). Line 108 reports the constraints derived from the guards of the exclusive gateways. Finally, in the *Then* clause, the state variable *var1* is pushed to the smart contract (Line 110) updating the contract state. This variable corresponds to the argument of the message, whose value is set by the user when the message is sent.

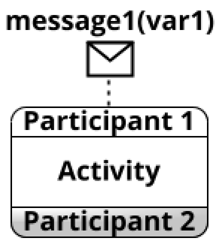


Figure 5.3: One-way Task.

```
104 Rule "message1"
105 When
106     <conditions on previous messages> &&
107     message1 == enabled &&
108     <conditions from exclusive gateways>
109 Then
110      push var1
111 End
```

Listing 5.1: Rule of a one-way task.

It is worth noticing that, after the rule action (in the *Then* clause) is executed, the message state is automatically set to completed in the smart contract.

Sequence flow connectors are reported in Figure 5.4 and their translation is shown in Listings 5.1 and 5.2. Sequence flows are used to specify the execution order of activities in choreography and the reported case considers two one-way tasks connected having a rule for message *message1* and message *message2*. In the *When* clause, the rule contains two conditions (Lines 114-115), expressing that the state of the previous message *message1* must be *completed* and the current state of *message2* must be *enabled*. This check is really important to enforce the execution of messages in the right sequence. If both conditions are satisfied, the *var2* variable is written in the blockchain (Line 117).

```
112 Rule "message2"
113 When
114      message1 == completed &&
115      message2 == enabled
116 Then
117      push var2
118 End
```

Listing 5.2: Rule of a task following another one.

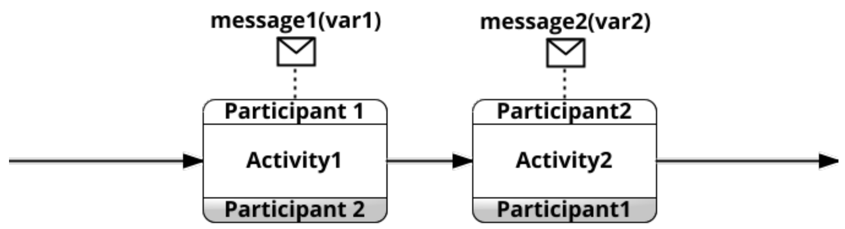


Figure 5.4: Sequence Flow.

Gateways affect the conditions of a rule when a message is directly connected to them and the first case is reported in Figure 5.5a and translated in Listing 5.3. The example refers to a split exclusive gateway in which outgoing sequence flows contain a boolean expression on the *var1* variable and are connected to *message2*. The rule contains an additional boolean expression referring to the condition of the exclusive gateway (Line 123). Depending on the type, the compare operator inside the expression can be on integer, string or boolean values.

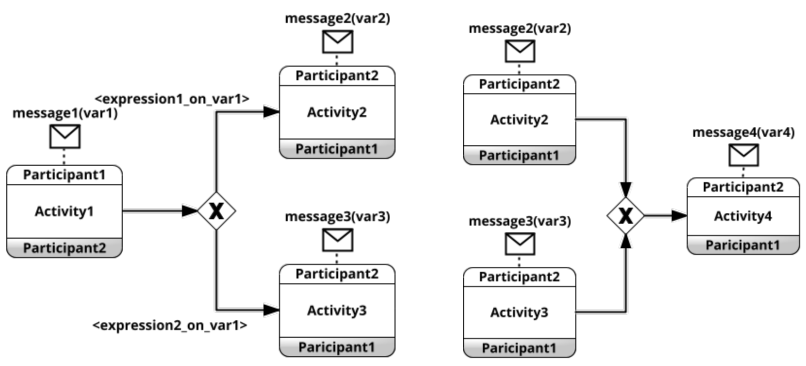
```
119 Rule "message2"
120 When
121     message1 == completed &&
122     message2 == enabled &&
123     expression_on_var1 == true
124 Then
125     push var2
126 End
```

Listing 5.3: Rule of a message after a split exclusive gateway.

Another case considering the exclusive gateway is the join one in Figure 5.5b. This time there is no expression to verify so the rule of *message4*, reported in Listing 5.4, contains only the execution state to control before the execution. Indeed *message4* requires that only one between *message2* and *message3* is completed (Lines 129-130).

```
127 Rule "message4"
128 When
129      (message2 == completed ||
130     message3 == completed) &&
131     message4 == enabled
132 
133 Then
134     push var4
135 End
```

Listing 5.4: Rule of a message after a join exclusive gateway.



(a) Split case (b) Join case

Figure 5.5: Exclusive gateway elements.

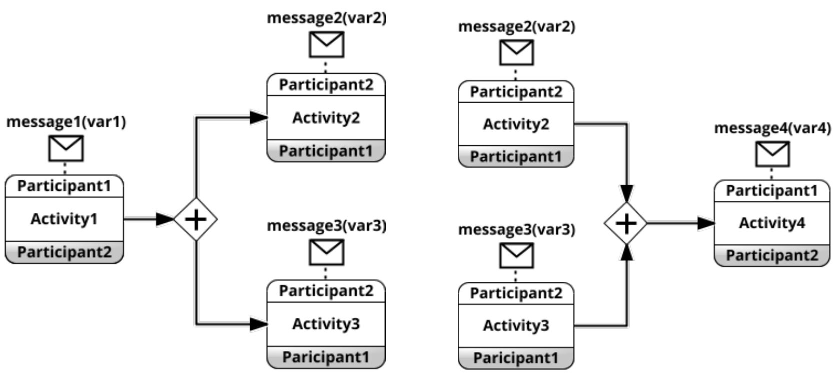
The next case regards the parallel gateway element that is divided again into a split and a join. The split gateway in Figure 5.6a, enables the execution of all successive messages without any restriction. This behaviour does not affect directly any rule neither of the messages before the gateway nor the ones after. All the messages on the right of the gateway are enabled after the completion of the previous one as usual for the others. For this reason, in Listing 5.5 the rule of *message2* does not contain any particular condition but it works as two directly connected simple messages.

```
136 Rule "message2"
137 When
138     message1 == completed && message2 == enabled
139 Then
140     push var2
141 End
```

Listing 5.5: Rule of the message after the split parallel gateway.

Different is the case for the parallel join reported in Figure 5.6b. In this case, the *message4* requires that all the previous elements are completed before its execution. This semantics affects the initial check of the rule associated with *message4*. Specifically, Listing 5.6 shows (Line 145) the parallel check in the condition for the execution state of the previous messages *message2* and *message3*, that need to be both completed.

```
142 Rule "message4"
143 When
144     message4 == enabled &&
145     message2 == completed && message3 == completed
146 Then
147     push var4
148 End
```



(a) Split case (b) Join case

Figure 5.6: Parallel gateway elements.

Listing 5.6: Rule of a message after the join parallel gateway.

The last case reported in Figure 5.7 shows an event-based gateway. The

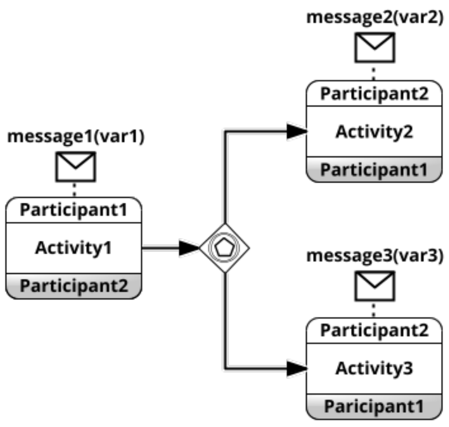


Figure 5.7: Event-based case.

gateway as shown in Listing 5.7 requires that both *message2* and *message3* are enabled but then, only one can be executed according to the first chosen. This is reflected in the inclusion of an extra condition requiring that the state of the other message connected to the gateway is enabled. Line 151 reports the example for *message2* in which the check is done also on the state of *message3*.

```
Rule "message2"
When
    (message1 == completed && message3 == disabled) &&
     message2 == enabled
Then
     push var2
End
```

Listing 5.7: Rule of a message after event-based gateway.

Retail Process Example (Translation) In Listing 5.8 the translation for the *Quotation* message is reported. The rule is generated and evaluated inside the FlexChain tool so it contains some utility code to communicate with it. In lines 158 and 159, there is a check on the state of the current and previous message while in Line 160 the *isAvailable* variable has to be false. If those conditions are successfully evaluated, an array is created with the names of the input variables (Lines 162-164). The last part of the rule creates an array with the input values sent by the user (Lines 165-167) and finally pushes it together with the names one to the contract (Line 168).

```
156 rule "Message_1h3ew61"
157 when
158      b : BlockchainUtils(b.getState("Message_1h3ew61")==0,
159      b.getState("Message_1xxdwx2")==2,
160      b.getVariableFromContract("isAvailable")==false)
161 then
162      List names = new ArrayList();
163      names.add("product");
164      names.add("quantity");
165      List values = new ArrayList();
166      values.add(b.getSingleInput(0));
167      values.add(b.getSingleInput(1));
168      b.setVariablesToContract(names, values,
            "Message_1h3ew61");
169 end
```

Listing 5.8: Rule of quotation message.

Smart contract In this part, the choreography instance and factory smart contracts are described. The former is responsible for storing the current execution state of the choreography and has a generic structure and is deployed automatically by the factory contract, which instead facilitates the managing of instances and acts as a monitor of each generated contract. Notably, the translation examples previously shown do not produce smart contracts code, since the Drools rules generated from the BPMN elements are stored inside the IPFS and just referred to in the Instance contract.

The **Factory contract** is reported in Listing 5.9 and it contains the main functionalities for instantiating new contract instances. Line 171 contains the mapping used to associate the instance name (in bytes) to its address. In this way, it is possible to track the entire history of generated choreography contract instances. The *instantiateProcess* function (Lines 172-178) generates a new contract starting from the template one. The new contract requires two inputs, (i) the creator, automatically taken from the sender of the invocation, and (ii) the quorum, required for an update. After this step, the newly generated address is stored in the state mapping.

```
170 contract ProcessMonitor{
171  mapping(bytes32 => address) processes;
172  function instantiateProcess(
173     bytes32 processName,
174     uint _quorum) public{
175         processes[processName] =
176         address(new ProcessTemplate(
177             msg.sender, _quorum));
178  }
179  ...
180 }
```

Listing 5.9: Factory Smart Contract.

Listing 5.10 reports the code for the **Instance contract** representing the choreography instance. In particular, in Line 183 the *enum* variable for managing the execution state of the rules is defined. Then, the hash location of the rules inside the IPFS is stored in a separate bytes32 variable (Line 183). The next mappings are used instead to associate the messages to an execution state (Line 184) and to store the input variables of the messages with their name (Line 185). Finally, two structures are used to manage the voting system for updating the process. The first (Line 186) represents the participants that vote for an update, while the second (Lines 187-191) handles the proposal. This contains indeed the new hash of the updated rules, the required quorum, and the actual votes received for that proposal. The next part of the contract defines the execution of the process instance. In Line 192 the *executeMessage* function is used to emit the event corresponding to the *messageToExecute* and its *inputs*. In the FLEXCHAIN approach, the above event is used to trigger the execution of the message rule; after this step, the result of the off-chain computation is pushed to the contract using the *setVariables* function (Line 197). Here the *names* and the *values* of the output variables are updated in the contract state and the message is set to the *COMPLETED* state. The function defined in Line 203 manages the creation of an update proposal, requiring the hash of the location of the new rules and the IDs of the corresponding messages. The last function in Line 208 permits to vote for the update proposal and it requires only the *vote* that will be associated automatically with the sender of the transaction (i.e., the choreography participant). The remaining part of the contract defines all the getter functions for reading the contract state and other information.

```
contract ProcessTemplate{
 enum State { ENABLED, DISABLED, COMPLETED }
 bytes32 rules_ipfs;
 mapping(bytes32 => State)  elements;
 mapping(bytes32 => bytes32) allValues;
 struct Voter{bool voted; bool vote;}
 struct Proposal{
    bytes32 proposedHash;
    uint quorum;
    uint actualVotes;
 }
 function executeMessage(
    string memory messageToExecute,
    string[] memory inputs
 ) public{...}

 function setVariables(
    bytes32[] memory names,
    bytes32[] memory values,
    string memory messageID
 ) public{...}

 function createProposal(
    string memory _proposalHashRules,
    string memory _proposalHashIds
 ) public{...}

 function voteProposal(
    bool _vote
 ) public{...}
 ...
}
```

Listing 5.10: Choreography Instance Smart Contract.

5.2.3 Instantiation

The first operation in the proposed framework is the instantiation of a model. This functionality can be triggered directly from the FlexChain interface after the design of the model using the provided modeller based on the Chor-js implementation [65]. The modeller indeed permits to create, upload, download and deploy a choreography model. Using the *deploy* button on the interface, the instantiation process starts as reported in Figure 5.8. The first step consists of the automatic generation of the corresponding rules and their storage inside the IPFS. This generates the hash of the rules that is sent back

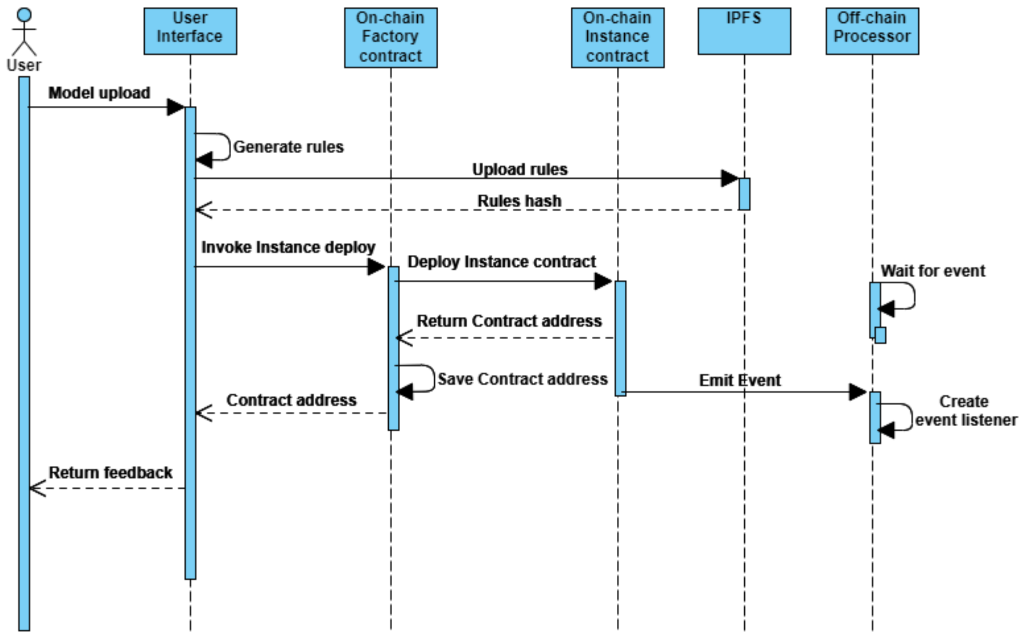


Figure 5.8: Sequence diagram of the instantiation phase.

to FlexChain. At this point, the factory smart contract is invoked passing the generated IPFS hash and the number relative to the *quorum* that must be reached in an eventual update phase. The factory contract will use a fixed template to generate a new instance contract containing all the information relative to the uploaded model such as the list of the choreography elements. The address of the newly generated contract is then stored in the factory that emits an event to notify the external process about the existence of a new contract instance.

Retail Process Example (Instantiation) To show in practice this first phase, Figure 5.9 reports the FlexChain modelling and deployment page with the retail process. In order to allow a translation suitable to the rules and contract generation, FlexChain requires the messages to be formatted as presented in the previous frameworks. However, since in this case the messages are not translated directly into Solidity functions, the variables inserted do not require a type. Indeed, the first message of the example *retail_quotation(good, amount)* only specifies the names of the variables that will be used later in the execution. To additionally support the modelling phase, FlexChain integrates also the standard property panel for facilitating the insertion of the required information.

Once the model is created, it is possible to export it in .XML so as to be available for the next interactions. At this point, before proceeding with the deployment of the contract, the quorum parameter must be indicated in

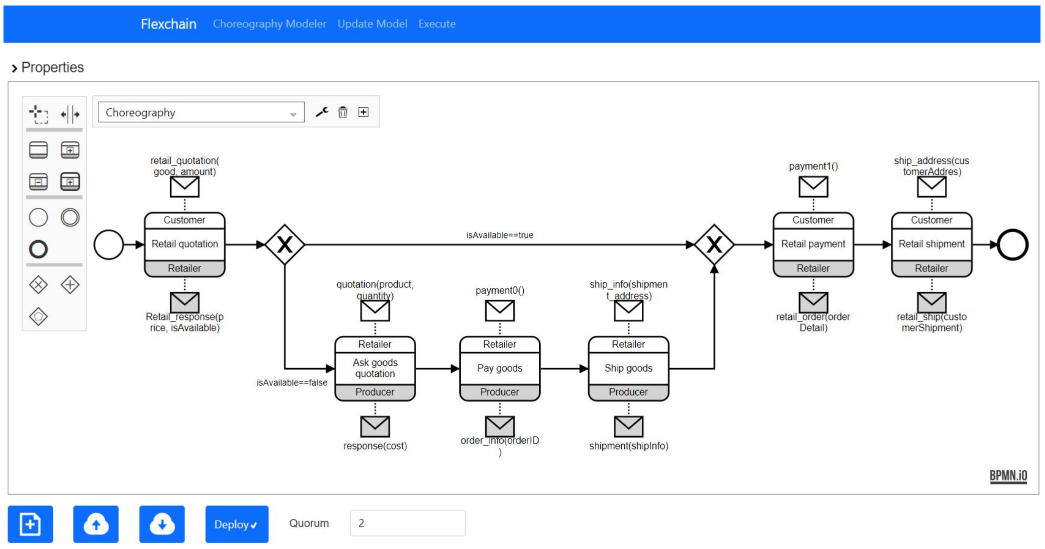


Figure 5.9: Modelling page.

the dedicated form. This value will be used during the updating phase, in which the involved participants will have to vote for changes. Finally, the *Deploy* button allows the user to launch the procedure (described above and in Figure 5.8) for the translation of the model into Drools rules, their upload into the IPFS, and into the newly generated contract instance.

5.2.4 Update

In the update phase, it is possible to make run-time changes in order to react to unexpected situations or to optimise the running process. The FLEXCHAIN tool provides a dedicated page visible in Figure 5.11 where the user can design the updates. The first step consists to upload the .bpmn file corresponding to the original choreography to change, the model will be then loaded inside the two panels of the page. The upper one is a viewer and it only contains the imported choreography, the lower one instead is a modeller and it can be used by the user to change the bpmn elements creating the updated version of the model. After this, it is possible to click the *View Changes* button to show the differences between the two models that will be coloured in green (added elements) or in red (removed elements). To effectively upload the changes in the blockchain, first, the user has to select the related smart contract using the drop-down list. This is possible thanks to the factory contract that during the instantiation associates the process names to their addresses (acting as a monitor) so as to be easily retrieved. After selecting the contract, a first control comparing the imported choreography elements (not the updated ones) with those stored inside the instance

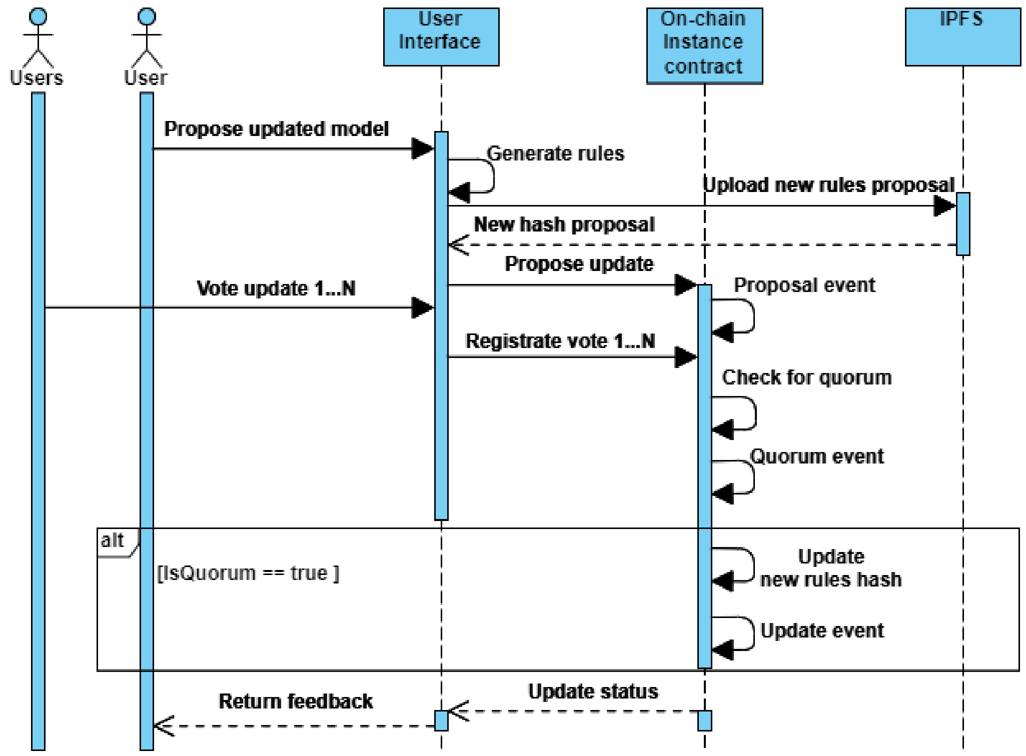


Figure 5.10: Sequence flow of the updating phase.

contract is made. In case they do not match, an error is shown to the user, otherwise, the *Update rules* button is enabled and the update process as described in Figure 5.10 starts. At this point, the FlexChain tool generates the new rules for the updated elements and stores them inside the IPFS. After that, the voting procedure starts for approving the new rules and adopting them. Indeed, whenever a new update is made in the blockchain, it does not immediately change the state of the contract but it comes as an *update proposal* that must be approved by the other participants. When a proposal is created, an event is emitted and it is captured by the other participants that can vote using the interface for approving or not the proposal. When the quorum is reached, the previously created IPFS hash is permanently written in the blockchain and the participants are informed.

It is worth mentioning that the aforementioned update mechanism introduces some security that should be discussed. Exploiting the FlexChain functionalities, it is indeed possible for a participant to propose a malicious update of the choreography without the consensus of the counterparts. Here, thanks to the use of an on-chain voting mechanism, all the participants need to vote on the proposal previously stored in the smart contract. Then, only in case the quorum is reached the smart contract adopts the new rules. Otherwise, the proposal can be rejected, thus interrupting the update. However,

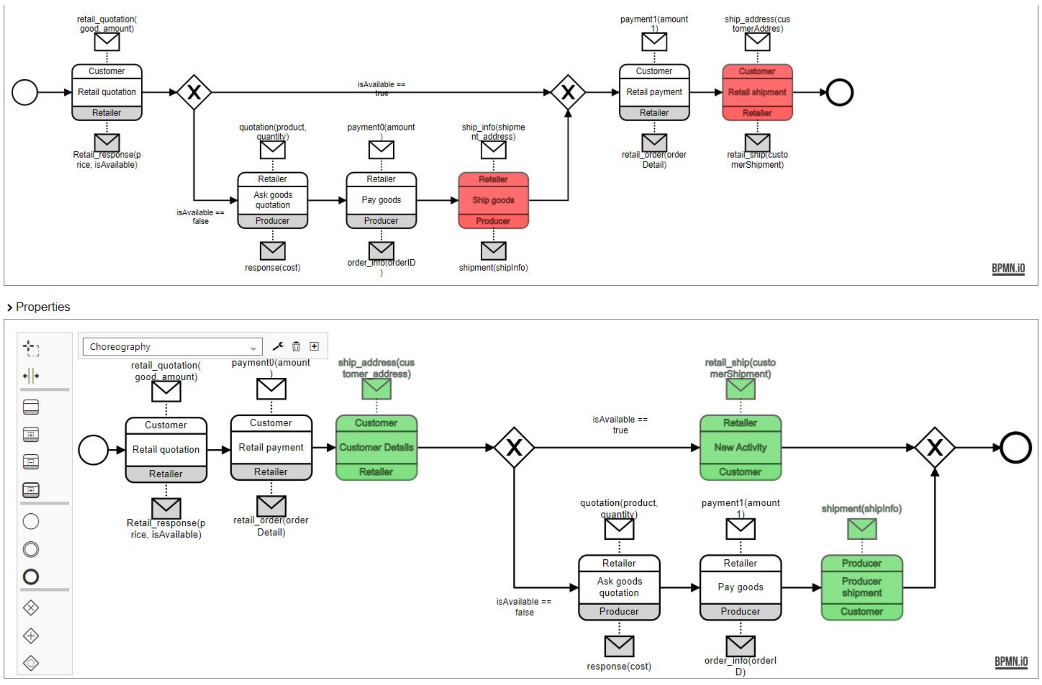


Figure 5.11: Updating page.

the voting procedure can be exploited to introduce latency and noise in the collaboration, forcing the other participants to vote and consequently pay execution fees. To solve this, it is possible to set a limit to the proposal that each participant can perform and that depends also on its correctness. For instance, if a participant proposal is rejected two times consequently, a further proposal is automatically blocked.

Retail Process Example (Update) Figure 5.11 shows the updating page in which the retail process is updated according to the changes described and motivated in Section 2.2.2. As described above, the page presented two main views. In the upper one, the original version of the retail process is shown while in the lower view, the updated version is available. In particular, the updated model has three new tasks (*Customer details*, *Retail shipment* and *Producer shipment*) with three related messages and two tasks removed (*Ship goods* and *Retail shipment*). Also, the *Retail payment* task is moved. Once the changes are modelled, with the *Show Changes* button it is possible to have a graphical representation of the current changes. In particular, the removed tasks are coloured in red and the new ones are coloured in green. Differently, the moved task is not marked since this simple change has no real impact on the underlying infrastructure.

At this point the user can trigger the propagation of changes through the *Update Rules* button, notifying the other participants that the voting is available.

5.2.5 Execution

In the execution phase shown in Figure 5.12), participants can interact by sending messages according to the flow prescribed by the model. Figure 5.13 shows the execution page, where the user can select the choreography to execute from the list of available ones. At this point, the user can select one message to execute from the list of available ones reported in the drop-down menu (automatically retrieved from the blockchain). The required parameters can be inserted in the dedicated form provided by the FlexChain user interface. Finally, the button *Execute message* sends the request to the smart contract that emits an event containing the message to execute, the IPFS hash of the rules and the inserted inputs. The off-chain processor captures this event, reads the rules from the IPFS and executes the one related to the message. If the conditions are successfully evaluated, the inputs are written in the smart contract producing an instance state update. This returns also feedback to the user informing them about the status of the execution. To notice, in the proposed scenario the off-chain processor is implemented as a Spring Boot server while the user interface is a React application. However, this is only one of the possible implementations since our approach fits also other technological choices.

Retail Process Example (Execution) The last phase of the FlexChain approach permits the participants of an instance to collaborate at run-time exchanging messages. In Figure 5.13 the execution page is reported with the updated version of the retail process. On this page, after selecting the contract from the list of deployed contract instances, the BPMN model is shown. In addition, two forms allowing the execution are available. The first one (*Message ID*) selects the message to execute from the list of ones available in the model. On the right instead, the *message Parameters* form requires the input to be passed to the function that will be evaluated in the smart contract. Finally, the *Execute Message* button triggers the concrete execution of the message function in the smart contract. From this, the procedure invokes the off-chain processor and the IPFS, following the steps described in Figure 5.12.

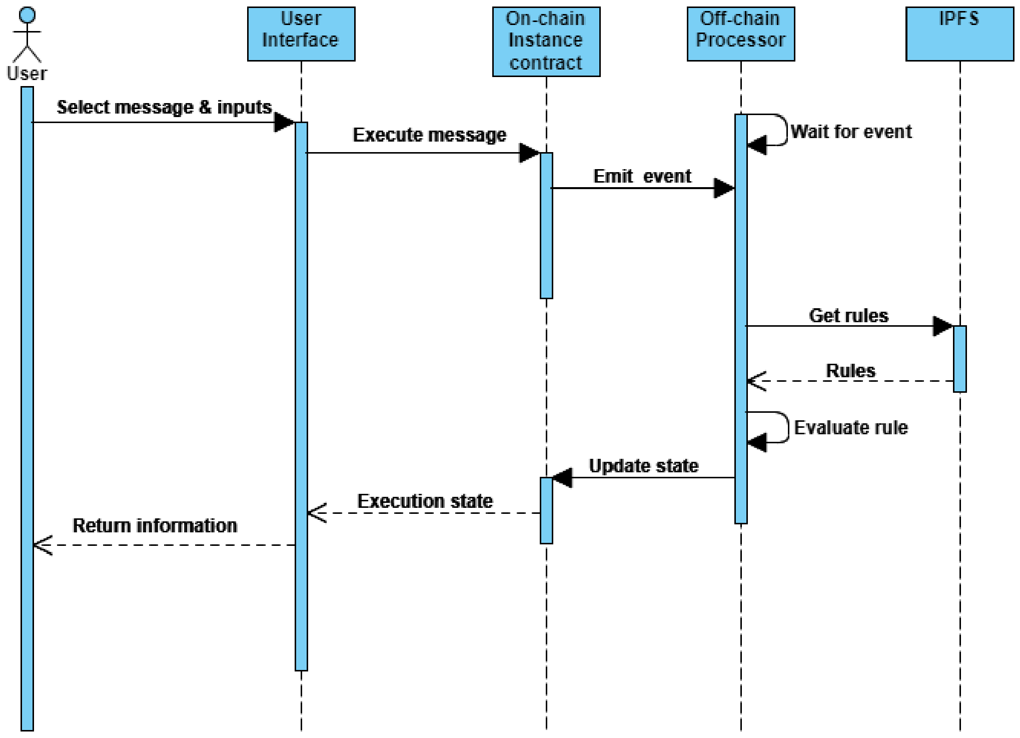
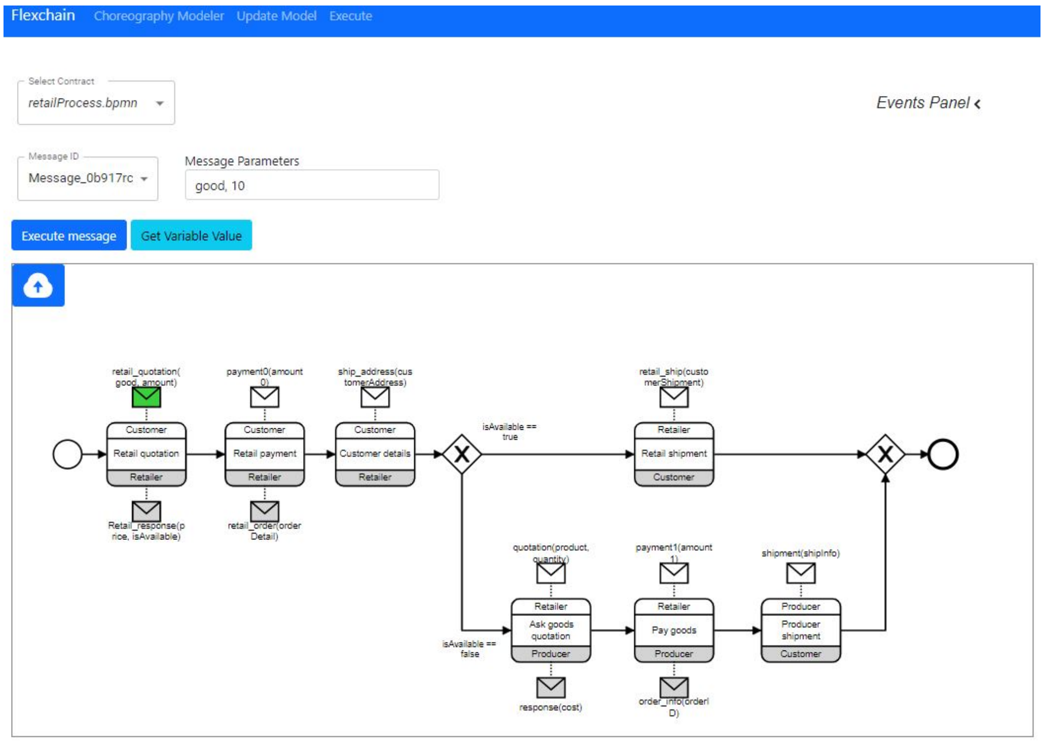


Figure 5.12: Sequence flow of the execution phase.



Select Contract: retailProcess.bpmn

Message ID: Message_0b917rc

Message Parameters: good, 10

Figure 5.13: Execution page.

Table 5.2: Cost Analysis for the X-rays process.

Transaction Name	**Gas Used**
Factory contract deploy	1,879,525
Template contract instantiation	1,936,369
IPFS hash upload	158,776
Update proposal	212,114
Vote for proposal	77,758
Average message execution	62,183

5.3 Experiments and Validation

Table 5.2 reports the cost analysis made on the execution of the proposed case study[6]. In particular, the main costs are related to the deployment of the factory contract (around 1.8 million units of gas) and to the instantiation of the template for a new choreography (around 1.9 million units of gas). However, the factory is deployed only once since it is responsible for deploying other processes. The template contract instead, is deployed every time a new choreography is uploaded and it will store the id of the IPFS location of the rules. This step requires a transaction consuming a significantly lower amount of gas (around 158,000 units of gas). Once the choreography is deployed, it can be changed by making an updated proposal to the contract (around 212,000 units of gas) that must be approved by the other participants (around 77,000 units of gas). Compared to the deployment operations, the update one comes with lower gas consumption. Indeed, thanks to our approach, instead of deploying new contracts every time a change is done, it is only necessary to complete the voting mechanism storing the new IPFS referrals. Finally, users can participate in the execution phase by sending messages. This is the less consuming action since it requires only around 62,000 units of gas. Indeed, it is important to maintain low consumption due to the high frequency of exchanged messages regarding the deployment operations. In general, the overall approach can be considered gas efficient since only one-time operations have a major consumption of gas, in favour of the most efficient execution.

[6]The execution traces of the factory and instance contracts are available respectively at `https://sepolia.etherscan.io/address/0x48fb86c35c0Ec23D671139A5Befb5a01d1e43c4f` and `https://sepolia.etherscan.io/address/0xb6eeff070e0a199587c34022325633be168c645c`.

5.4 Comparison with Existing Approaches

The challenge of providing flexibility during the execution of process-aware information systems is largely discussed in the literature and many process management systems supporting flexibility are currently developed [103, 79]. Also, different approaches are provided in many contexts, for example, in [50] the authors proposed a real-time system for dealing with the flexibility of cloud service workflows. In particular, they combine BPMN and UML which are used to represent the functional and behavioural views of the cloud workflows. Then, during the execution phase, the proposed system uses defined QoS (Quality of Service) attributes to control whenever an anomaly occurs correcting it.

Differently, support for flexibility in business process modelling and execution can be achieved in different ways. In [78] the authors proposed an extension for BPMN named CF4BPMN in which the main objective was to obtain controlled flexibility. Indeed, thanks to this extension, process designers can model which business process element can change and under which conditions by using specific expressions.

In [60] instead, the versioning is faced. Here an approach supporting dynamic changes in business processes is proposed and issues on version management for process changes are discussed. The proposed solution is based on change patterns that are concretely supported by a process designer extending BPEL (Business Process Execution Language).

Finally, in [116] dynamic business processes are executed using rules that are selected dynamically based on the surrounding context. In this way, it is possible to support changes at run-time by executing new actions and rules coming from the updated context.

All the approaches described above show how it is possible to obtain flexibility in different kinds of business processes. However, all of them significantly differ FlexChain since its focus concerns the execution of blockchain-based processes guaranteeing trust while achieving flexibility. This topic is still almost unexplored, only a few approaches are available.

In [73] the authors propose a dynamic role binder for run-time choice of sub-processes. They provide functionalities for the binding and unbinding of actors to a role in a dynamic way, supported by consistency verification and based on agreements. However, the proposal is mainly a run-time selection of already designed situations, with a focus on dynamic role-binding. FlexChain, instead, wants to provide flexibility during the execution of the business process, without the need of specifying in the modelling phase the elements of the model that could be modified at execution time.

In [2], the work focuses on reaching flexibility on the technologies executing business processes. The resulting system is indeed developed with on-chain and off-chain components based on federated blockchains. However, here the focus is on the flexible interactions between different technologies. FlexChain instead, wants to achieve flexibility in the execution of the business process itself and not on the underlying technologies.

Finally, in [62] the issue of upgradeability is faced. The authors investigate how starting from a collaboration diagram, it is possible to upgrade the smart contracts representing the business process, its state and its participants. In particular, each participant involved in the collaboration is translated into a separate smart contract. Then, different patterns are used to permit an upgrade of the collaboration. Contracts version are indeed associated with their names in a dedicated structure, such as storage which is separated from the business logic. Also, to invoke a contract function, a proxy forwards the request to the logic contract. In this way, whenever an upgrade occurs (after voting), the proxy contract updates the logic contract's address reference to the latest. In this case, the differences with FlexChain consist of different aspects. First of all, FlexChain aims at providing a flexible execution at run-time of active instances of business processes, here instead the authors focused on the dynamic versioning of inactive instances. Then, the works represent the integration between parties with a BPMN collaboration diagram, later translated into blockchain code. Finally, the proposed infrastructure of different smart contracts introduce an elevated level of complexity and high deployment costs.

Table 5.3: Table of identified related works and their characteristics.

Source	BPMN	Blockchain	Flexibility
[50]	✓	✗	Scaling deployed services during the workflow
[78]	✓	✗	Modelling and executing controlled flexibility
[60]	✗	✗	Dynamic process version selection
[116]	✓	✗	Rule- and context-based modelling and simulation
[73]	✓	✓	Run-time selection of predesigned elements
[2]	✓	✓	Dynamic blockchain selection
[62]	✓	✓	Upgradeability and versioning approach

Table 5.3 summarises the analysed works concerning flexibility and considering three main characteristics: (i) if the work explicitly uses BPMN diagrams for modelling, (ii) if the work relies on blockchain, and (iii) which flexibility aspects the work supports, and how it implements them. The table shows that in most cases flexibility is achieved in a setting that does not rely on blockchain technology, hence without facing all issues raised by the immutability of the blockchain. In the approaches that, instead, adopt blockchain for a trusted environment, the concept of flexibility they consider differs from work to work. In [2], the authors consider flexibility as the possibility to select different blockchains, while in [73] the authors provide the

possibility to bind actors and control-flow decisions at run-time. In [62], the work focuses on the smart contract upgrading mechanism, thus proposing a versioning approach. Differently, FlexChain focused on a richer flexibility mechanism for business processes, whose business logic and execution state is stored in the blockchain, supporting a voting mechanism that permits arbitrary changes in the model at run-time.

CHAPTER 6

MICHAIN: SUPPORTING MULTIPLICITY

In this chapter, the thesis faces the multiplicity challenge of executing inter-organisational business processes on blockchain. Indeed, the proposed ChorChain framework and the derived versions focus on inter-organisational systems that involve single components for each participant willing to collaborate. However, such a single-instance modelling approach seriously hinders the applicability of these model-driven solutions to those systems where the number of instances covering a particular role can change arbitrarily. In particular, when multiplicity is represented using single-instance elements, the modelling complexity drastically increases, due to the high number of elements and conditions leading to error-prone implementations. For this reason, using multi-instance representations can reduce the modelling complexity and enable more compact models.

Typical examples of such kinds of systems are represented by the described retail process (see Section 2.2.2), auctions and the Internet of Things (IoT). In those cases, there is a strong need to represent multiple actors such as producers, sensors or bidders. However, when considering choreographies, the main issue is related to the semantic definition of multi-instance elements due to the absence of clear formalisation in the BPMN standard [21]. For this reason, the thesis introduces the MIChain framework, a revised version of ChorChain for supporting the modelling and execution of choreographies with multiple instance elements. In particular, MIChain adopts a novel solution for including multiplicity in both participants and their operations, enabling new kinds of scenarios to be created.

The following sections introduce the revised framework with the conceptual description of the supported multi-instance elements and the introduced attributes. Then the implemented tool is presented with a particular focus on the proposed support for multiplicity aspects. Then, the experiments on the running example are presented with a final overview of already existing approaches.

6.1 MIChain: Supporting Multiplicity

This section illustrates the novel strategies adopted inside the proposed model-driven approach for dealing with multiple instances in inter-organisational business processes. MIChain permits indeed to enable a richer set of executable scenarios while inheriting the fundamentals of the ChorChain framework to address trust concerns.

6.1.1 Framework phases

Figure 6.1 reports the various phases based on the ChorChain framework extended to support multiplicity aspects. The resulting MIChain framework encompasses novel strategies, focusing on the modelling of a choreography diagram and on its execution on the blockchain. In particular, a novel mechanism is adopted for automatically translating the choreography into Solidity code and deploying it on the blockchain.

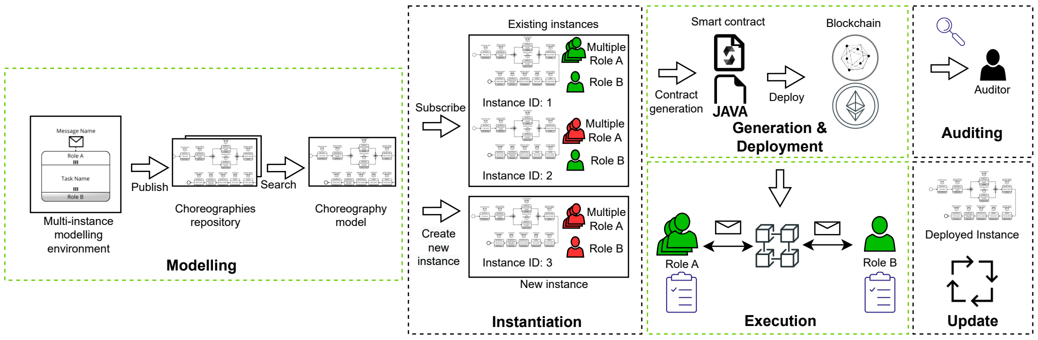


Figure 6.1: MIChain framework: supporting multiplicity.

The first phase foresees the **modelling** of a process using the choreography modelling environment. In this phase, the integrated environment was extended to fully support the creation of multiple-instance elements, and the connected attributes specification. Indeed, at design time, MIChain requires the developer to mark roles dependently if they are single- or multi-instance. This will result in a different binding procedure between participants and roles, without having an explicit subscription functionality. More details on how participants are defined in choreography are provided in the next sections.

In the **generation** phase, the completed model is processed by a translator that automatically creates a smart contract which is then deployed on the blockchain. The smart contract implements the different activities that participants have to perform as defined in the corresponding choreography model. For each activity in the model, a function enforces the specified multiplicity of invocations, the cardinality of participants, and the exchanged data. Notice, in the current state of MIChain there is no support for explicit multiple instantiations of the model since this aspect is concealed in favour of a direct instantiation for each created model. Indeed, the definition of multi-instance roles can differ for each instance thus requiring additional technicalities for their management.

After the generated contract is deployed on the blockchain, the final foreseen phase refers to the **execution**, in which the parties of a system communicate by exchanging messages. This is done by invoking the deployed smart contract functions that then coordinate the overall execution, keeping an immutable track of all the exchanged messages and their data. The contract contains also a set of controls based on the state of the messages and of the multi-instance elements. These mechanisms enforce the right execution flow and ensure that all the steps are performed accordingly to the initial choreography.

6.1.2 Modelling multi-instance elements

This section describes how the modelling of multiple instances is supported by the proposed approach and the resulting choices addressing the lack of the BPMN standard. First, a definition of the multi-instance concept is given, and then the cases supported by MIChain are shown.

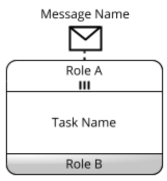
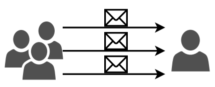
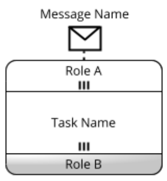
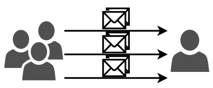
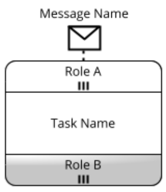
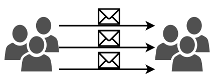
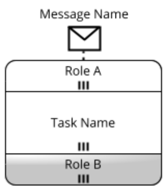
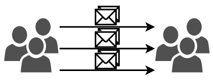
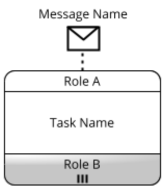
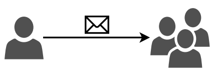
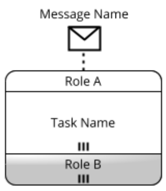
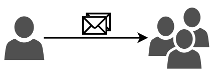
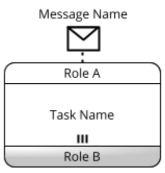
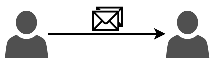
In BPMN, the multiple instance behaviours can be identified on participants or tasks, marking them with a three-vertical dashes symbol. This kind of behaviour is very useful to avoid the creation of loops and additional elements improving the understandability of the model. Furthermore, multiple instances allow the expression of complex behaviours in a compact way. This marker expresses the parallel behaviour where participants can interact simultaneously in the same task. Table 6.1 reports all the possible supported cases of multiple instances. In the first case on the left column (case 1), the initiator is marked as multi-instance so multiple participants can cover the *Role A* sending a single message each to the receiver *Role B*. In the second example (case 2), both the initiator and the receiver are marked as multi-instance. This time, *Role A* will have many participants sending a single message to all receivers covering *Role B*. This means that each sender will send a broadcast message to all the receivers. The third example (case 3) considers only the receiver *Role B* as multiple. This case is rather simple since at run-time there is a single sender associated with *Role A* that sends the message in broadcasts to all the receivers. The next case considers multi-instance tasks. In particular, the behaviour of tasks is translated as a repetition of the defined action (i.e., sending a message). Indeed, in MIChain, during the translation, multi-instance attributes affect directly the messages that will include the handlers for multiple participants and tasks. In the most simple case, a multi-instance task is surrounded by single-instance roles (case 4). This means that a single sender delivers many times the same type of message to the same receiver, but with a different payload. The other cases are related to the combination of a multi-instance task with multi-instance roles. Indeed, in the case of multiple senders *Role A* and multiple tasks (case 5), each participant delivers many times a message to a single receiver. When both task and the two roles are multi-instance, multiple messages are sent in broadcast to all the receivers (case 6). Finally, in the last case, the same sender will deliver a message multiple times in broadcast to all the receivers (case 7).

6.1.3 Multi-instance attributes

Despite the choreography model permits the representation of multi-instance elements, additional information supporting their execution is necessary. In particular, those are referred to as *attributes* and depending on the case, specify information about the run-time execution of an element following the logic reported in Table 6.2.

The case of a single-instance role is the most simple one and it only requires the **address** attribute for representing the blockchain account of a fixed participant. Indeed, due to the different structure of the smart contract, this data is used for managing the reception of messages, so a single receiver must be explicitly specified in the model. In the case of multi-instance participants instead, they can directly join at run-time without any restriction on their identity (i.e., address). During the design, only the cardinality of senders/receivers must be specified, setting the **Minimum** and the **Maximum** attributes for each specific role. Finally, in a multi-instance task the **loopCardinality** attribute indicates the maximum number of times that a task can be executed. It is worth noting that during the generation phase, the attributes and behaviours of tasks and participants are inherited by the messages. Indeed, all the smart contract functions derived from messages will contain information related to multiple instances.

Table 6.1: Modelling cases of multi-instance participants and tasks.

	Multi-instance case	Behaviour		Multi-instance case	Behaviour
1)			5)		
2)			6)		
3)			7)		
4)					

6.1.4 Translation approach: BPMN to Solidity

The last aspect introduced in MIChain concerns the automatic generation of a smart contract implementing the multiplicity aspects above mentioned. In general, the translation lays its foundations on the one proposed in ChorChain which considers choreography messages and control flow elements as shown in Table 3.1. For each of them, a function inside the smart contract is created. In particular, messages represent the operations that each participant can perform and they are translated as a public function. On the contrary, gateways and events are automatically completed internally to the contract since they define the overall execution flow.

The main novelty introduced in the MIChain translation consists of the additional logic generated to manage multi-instance participants and messages. Indeed, each element contains not only the execution state but also the inherited multi-instance attributes defined in the modelling phase (if

Table 6.2: Supported multi-instance attributes.

Case	Attribute Name	Behaviour
Single instance participant	Address	Defines participant account
Multi-instance participant	Minimum	Minimum num. of participants executing the message
Multi-instance participant	Maximum	Maximum num. of participants executing the message
Multi-instance task	Loop cardinality	Num. of executable times

present). These are then used inside the functions for enforcing the cardinality of each execution with a double check both on the participants and on the message.

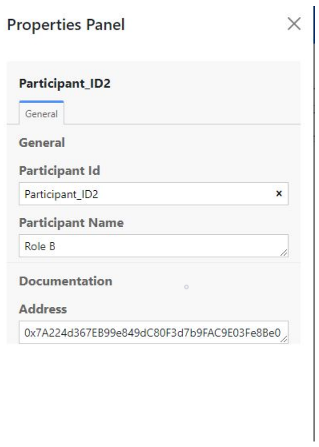
To handle the interaction between parties, after the invocation, each message function emits an event containing the input value of the message and a receiver address. This event is handled by the target receiver outside the blockchain, who takes appropriate actions accordingly to the received information and proceeds to the next step. In the case of a single-instance message, the function additionally stores the input data in the smart contract state, this information is used on-chain for verifying the conditions of exclusive gateways.

6.2 MIChain Tool

This section presents the MIChain implemented framework focusing in particular on the solution conceived to integrate relevant multiplicity aspects in the model, having a clear decoupling between the behaviour and the attributes of the multiple instances. To do this, a new architecture is provided including new smart contracts dynamically managing multiple participants and messages, enabling complex scenarios with role binding at run-time. All these aspects are combined in a new implementation covering all the multiple-instance phases starting from the design until their execution.

6.2.1 Modelling

This section shows how implemented MIChain tool supports the creation of choreography diagrams and, in particular, of multi-instance elements and attributes. Also, this section provides an example using the retail process


Properties Panel

Participant_ID2

General

General

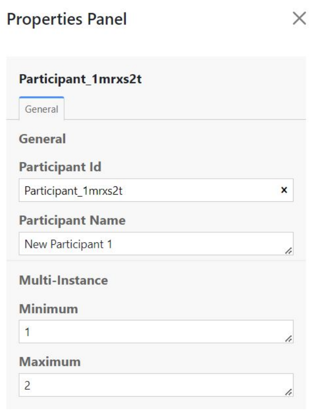
Participant Id: Participant_ID2

Participant Name: Role B

Documentation

Address: 0x7A224d367EB99e849dC80F3d7b9FAC9E03Fe8Be0

(a) A single-instance participant.


Properties Panel

Participant_1mrxs2t

General

General

Participant Id: Participant_1mrxs2t

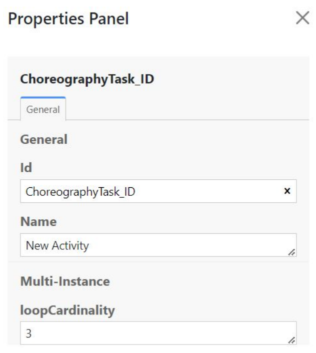
Participant Name: New Participant 1

Multi-Instance

Minimum: 1

Maximum: 2

(b) A multi-instance participant.


Properties Panel

ChoreographyTask_ID

General

General

Id: ChoreographyTask_ID

Name: New Activity

Multi-Instance

loopCardinality: 3

(c) A multi-instance task.

Figure 6.2: Multi-instance attributes modelling.

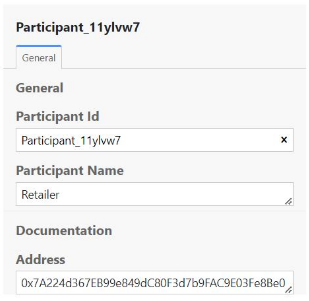
scenario in the implemented tool. In MIChain, the creation of a choreography diagram is supported in the provided modelling environment based on chor-js. This makes it possible also to create multi-instance participants and tasks already defined in the BPMN without any particular addition.

Retail Process Example (Modelling Elements) The proposed running example including multi-instance elements is described in Section 2.2.2. Here the model includes new behaviours such as the possibility for a customer of asking for more goods and for a retailer for asking a quotation from multiple producers, selecting then the most proper one.

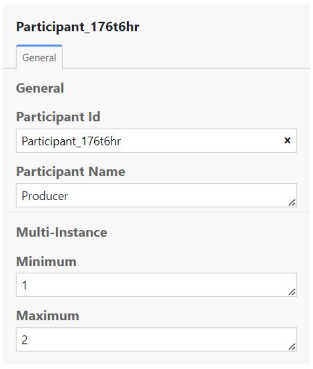
Multi-instance attributes A novel implementation was instead provided for the specification of multi-instance attributes. Indeed, this step can be performed using an extended properties panel in the proposed modelling environment as visible in Figure 6.2.

As described in Section 6.1.3, for single-instance roles, the panel only requires the **address** representing the blockchain account of the participant as visible in Figure 6.2a. In particular, this information is used for managing the reception of messages. While the address of the sender is automatically associated with the party executing the message, the receiver must be explicitly specified in the model. The panel of a multi-instance participant is reported in Figure 6.2b, where this time there is no need of setting the specific participant address. Indeed, this case requires the definition of the **Minimum** and the **Maximum** attributes. Finally, Figure 6.2c shows the property panel of a multi-instance task. Here only the **loopCardinality** attribute has to be set, indicating the number of times that the task (so the attached message) can be executed.

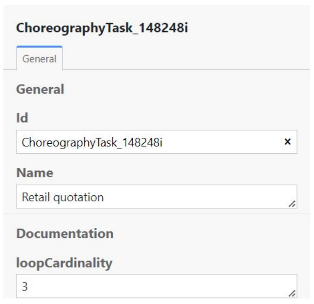
Retail Process Example (Modelling Attributes) Considering the proposed running example, Figure 6.3 shows the property panels for modelling the above-mentioned attributes. In particular, Figure 6.3a reports the *Retailer* single participant with the related address. Figure 6.3b and Figure 6.3c shows instead the definition of multi-instance *Producer* participant and *Retail quotation* task. For demonstration purposes, in the first case, the prouder has a minimum and a maximum of 1 and 2. The task instead, has a cardinality of 3.


Participant_11ylvw7
General
General
Participant Id
Participant_11ylvw7
Participant Name
Retailer
Documentation
Address
0x7A224d367EB99e849dC80F3d7b9FAC9E03Fe8Be0

(a) Retailer participant.


Participant_176t6hr
General
General
Participant Id
Participant_176t6hr
Participant Name
Producer
Multi-Instance
Minimum
1
Maximum
2

(b) Producer participant.


ChoreographyTask_148248i
General
General
Id
ChoreographyTask_148248i
Name
Retail quotation
Documentation
loopCardinality
3

(c) Retail quotation task.

Figure 6.3: Retail process multi-instance attributes modelling.

Messages The last aspect of the modelling phase is the definition of choreography messages. In this case, the syntax is derived by ChorChain. A special case relates to the *address* parameter as used in message2 and shown in Figure 6.4. In this case, the address parameter will identify a single receiver being part of a previous multi-instance interaction. For example, the participant playing *Role B*, after receiving messages from the participants playing *Role A*, will respond to one of them, only.

Retail Process Example (Modelling Messages) In the retail process, the new case introduced using the address parameter is visible in Figure 6.5 with the *payment0* message which also includes the parameter *_producer* to identify the single participant receiving the payment and the additional information.

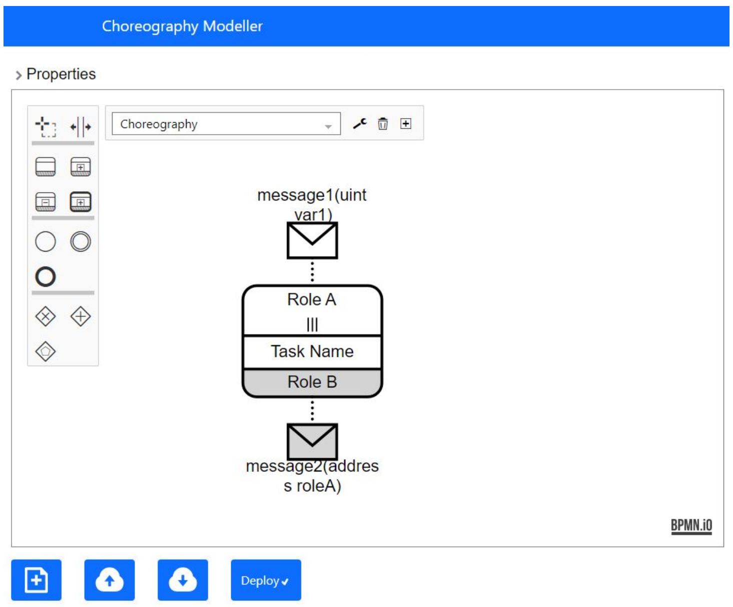


Figure 6.4: Provided Modeller.

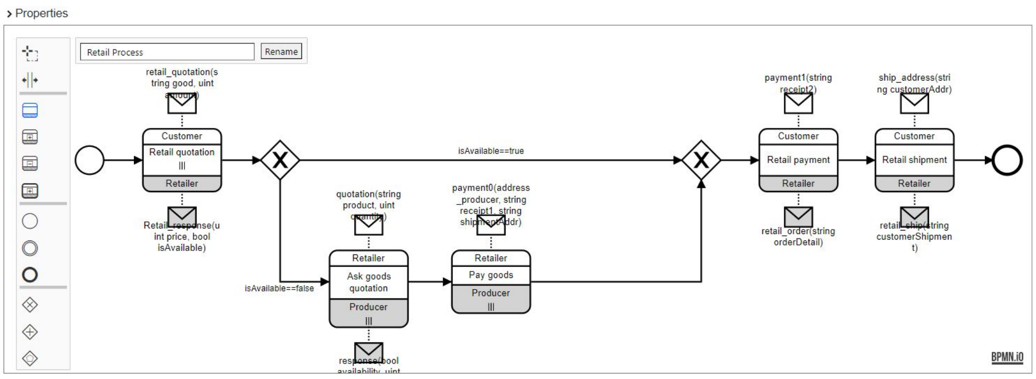

> Properties
Retail Process
Rename
retail_quotation(string good, uint amount)
Customer
Retail quotation
Retailer
Retail_response(uint price, bool isAvailable)
isAvailable==true
isAvailable==false
quotation(string product, uint quantity)
Retailer
Ask goods quotation
Producer
response(bool availability, uint
payment0(address _producer, string receipt1, string shipmentAddr)
Retailer
Pay goods
Producer
payment1(string receipt2)
Customer
Retail payment
Retailer
retail_order(string orderDetail)
ship_address(string customerAddr)
Customer
Retail shipment
Retailer
retail_ship(string customerShipment)
BPMN.iO

Figure 6.5: MIChain modelling page.

6.2.2 Generation

This part provides the implementation details of the smart contract generated by the translation passing in input the BPMN choreography diagram as described in Section 6.1.4.

Contract state The first part of the contract is reported in Listing 6.1 and it defines the core data infrastructure supporting the execution of multi-instance elements. In particular, the first 'struct' in Lines 214-221 defines how an *Element* is represented inside the smart contract and maintains information related to multi-instance attributes. The first variable in the struct represents the enabling state of the element (Line 215). Indeed, after the execution of each contract function, the corresponding element is disabled to prevent unexpected behaviours. Then, the *loopCardinality* variable (Line 216) contains the maximum number of times that a participant can interact with that element (e.g., sending a message). The *uniqueParticipants* array (Line 217) contains instead the unique addresses of participants that have interacted with that element. This will be used to check if the number of participants has reached the maximum allowed one as specified by the modelled attribute. This check will use the *participantMax* variable (Line 219) that, along with the *participantMin* one (Line 218) defines, the number of participants that have to invoke a function before it can be considered complete. Finally, the *cardinalities* mapping (Line 220) associates each invoking participant with the number of invocations, and it is used to cover multi-instance message cases. The overall list of elements is then stored in the *elements* mapping that associates each Id to the respective struct (Line 222). In Lines 223-225, the events representing the message exchange of the smart thermostat are defined. In particular, for each message in the model, an event is automatically generated. This event contains the actual sender, the receiver address and the input variables. Notice, in the case of a single-instance receiver, its address is identified by the specific one defined in the element attribute during the modelling phase. In multi-instance cases instead, the broadcast reception uses a receiver address in the form of all zeros.

Lines 226-230 show the global state of the contract that is represented by the set of variables exchanged only by single instance participants. As previously mentioned, these kinds of variables are unique, and they are used as enforcing mechanisms when an exclusive gateway is met. Finally, the constructor of the contract in Lines 232-235 creates the start event element and starts its execution. In particular, in line 233 the related element is created with a loop cardinality, minimum and maximum participants of 1.

This is due to the fact that events, such as gateways, are executed only internally and only one time. In this way, if on the one hand, this information can be unnecessary, on the other hand, it permits to have a flexible structure that can easily be extended or modified in case of new needs. Notice, to optimise and divide the overall costs, there is not a single initialisation for all the elements, but each of them is created in the previous function and, in case it is not a message, it is also executed.

```solidity
contract RetailProcess{
    struct Element{
        bool isEnabled;
        uint8 loopCardinality;
        address[] uniqueParticipants;
        uint8 participantMin;
        uint8 participantMax;
        mapping(address => uint8) cardinalities;
    }
    mapping(string => Element) elements;
    event retail_quotation(address from, address to, string good,
        uint amount);
    ...
    event order_info(address from, address to, string orderID);
    struct GlobalState{
        string good;
        ...
        string customerShipment;
    }
    GlobalState global;
    constructor(){
        createNextElement("StartEvent_102vawy", 1, 1, 1, true);
        StartEvent_102vawy();
    }
    ...
}
```

Listing 6.1: Contract state variables definition.

Execution functions The next part of the smart contract is reported in Listing 6.2 and it contains the function representing the *payment0* message. The first step retrieves the element from the related mapping (Line 239) later used for checks and data updates. Then, two *require* statements check the initial conditions. The first in Line 240 needs the message to be enabled. The second one in Line 241 has instead two parts, and it checks if the user has reached the maximum number of invocations (given by the *loopCardinality* variable) and if the message has reached the maximum number of invoking participants. If those conditions are successfully evaluated, the next step in Lines 242-244 checks if this is the first invocation of the user and, in such a case, it includes the participant to the *uniqueParticipants* array. Successively, the number of executions for the user is increased (Line 245), and the payment is made to the indicated participant (Line 246). Then, *receipt0* and *shipment_ address* state variables are updated (Lines 247-248). After this, the event containing the input variable to send to the receiver is emitted

(Line 249). This event contains the input variables and it uses the address of the sender as *from* field, while the receiver is indicated with the *_producer* address since it targets a single instance among the multiple ones. After this, Line 250 checks if the cardinality for that user and element unique participants have both reached the maximum. In that case, this means that the invoker participant was the last admissible one and that the last possible execution was performed. This kind of check covers the possible presence of both multi-instance messages and participants. After the last execution is reached, the state of the current message element is set to false (Line 251) and the next one is created (Line 252). To deal with the possible loops that can require a function to be executed again, the step in Line 253 resets the cardinalities, and the addresses inside the executed element, so as to be executable again if necessary. Finally, since the next element is a gateway, it is also executed (Line 254).

```
238 function Message_1v7cac0(string memory receipt0, string memory
        shipment_address, address _producer) public payable{
239     Element storage e = elements["Message_1v7cac0"];
240     require(elements["Message_1v7cac0"].isEnabled == true);
241     require(e.cardinalities[msg.sender] < e.loopCardinality &&
            e.uniqueParticipants.length <= e.participantMax);
242     if(e.cardinalities[msg.sender] == 0){
243         e.uniqueParticipants.push(msg.sender);
244     }
245     e.cardinalities[msg.sender]++;
246     payable(_producer).transfer(msg.value);
247     global.receipt0 = receipt0;
248     global.shipment_address = shipment_address;
249     emit payment0(msg.sender, _producer, receipt0, shipment_address);
250     if((e.cardinalities[msg.sender] == e.loopCardinality) &&
            (e.uniqueParticipants.length == e.participantMax)){
251         setState("Message_1v7cac0", false);
252         createNextElement("ExclusiveGateway_1johog7", 1, 1, 1, true);
253         resetCardinalities("Message_1v7cac0");
254         ExclusiveGateway_1johog7();
255     }
256 }
```

Listing 6.2: retail quotation message function.

In addition to messages, also gateways are translated as functions and they are automatically executed internally by the smart contract. Listing 6.3 reports the exclusive gateway after the *Retail quotation* task verifying the *isAvailable* variable. As for the messages, the state of the current element is checked and then disabled, resetting also the cardinalities for a possible other execution (Lines 258-260). Then, the condition on the isAvailable state variable is evaluated. As reported in the model, if the variable is true it means that the good is available and the process can continue with the shipment (Lines 261-264). In the other case, the good is not available and the retailers have to be involved (Lines 265-267).

```
257 function ExclusiveGateway_042aut8() private {
258     require(elements["ExclusiveGateway_042aut8"].isEnabled == true);
```

```
        setState("ExclusiveGateway_042aut8", false);
        resetCardinalities("ExclusiveGateway_042aut8");
        if(global.isAvailable == true){
            createNexElement("ExclusiveGateway_1johog7", 1, 1, 1, true);
            ExclusiveGateway_1johog7();
        }
        else if(global.stopThermostat == false){
            createNextElement("Message_1h3ew61", 1, 1, 1, true);
        }
    }
```

Listing 6.3: Exclsuvie gateway function.

6.2.3 Execution

The last phase of the presented framework concerns the execution of the generated smart contract[1]. To clarify this aspect, Figure 6.6 reports the sequence flow describing a simple interaction between a sender and a receiver in a two-way task. The first step is made by the sender that invokes the function in the generated smart contract corresponding to the request message to send. The contract emits then an event in the blockchain containing the message data that is captured by the receiver through an event listener. At this point, the receiver invokes the contract for delivering the response message that will be finally captured by the initial sender. In the implemented smart thermostat scenario, the participants correspond to the presented devices, interacting accordingly with the steps described in the sequence flow. The event listener instead, is a custom piece of software that captures events from the blockchain forwarding information.

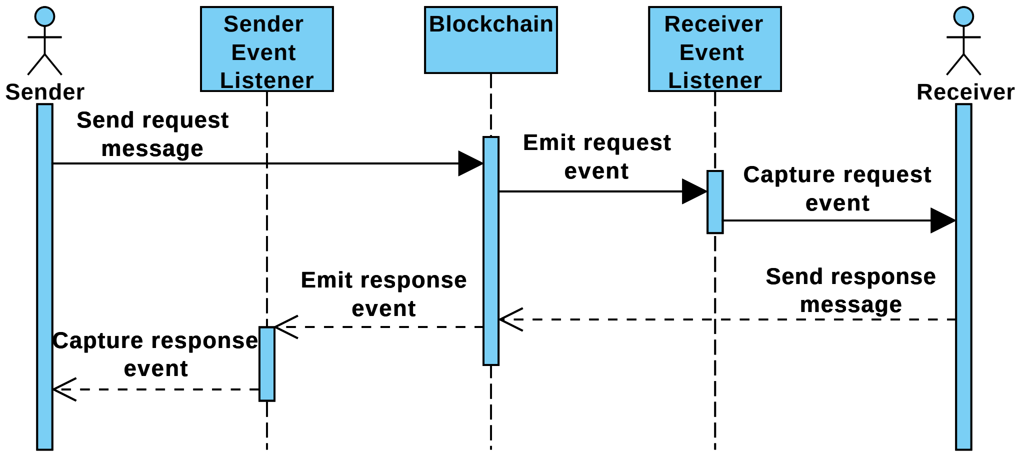


Figure 6.6: Two-way task execution sequence flow.

[1]Tool, smart contract and choreography code available at `https://bitbucket.org/proslabteam/michain/src/master/`

Table 6.3: Cost analysis for retail process scenario.

Executed message	**Gas Used**
Deploy	3,324,906
retail_quotation1	143,991
retail_quotation2	50,319
retail_quotation3	95,284
retail_response1	121,841
retail_response2	45,257
retail_response3	140,511
quotation	156,649
response1	97,710
response2	114,289
payment0	206,711
payment1	143,952
retail_order	138,428

6.3 Experiments and Validation

To assess the feasibility of the MIChain approach, this section reports the experiments made on the retail process scenario, generating the corresponding smart contract and executing it simulating the reading and writing from/to the different parties[2]. In particular, the choreography was executed entirely, following the longest path until reaching the end event, analysing the resulting performances in terms of gas costs. The experiments were based on the Mumbai testnet of the Polygon blockchain, one of the most prominent scaling solutions of Ethereum. This choice was made as a first attempt to scale the MIChain approach due to the potentially high number of involved parties. However, since Polygon is EVM based, the gas consumed by the contract can be considered independent from the underlying blockchain. Indeed, the chosen implementation mainly affects the execution times (depending on the consensus) and the final fee (depending on the currency exchange value).

Table 6.3 reports the execution cost for each message considering the possibility of having multiple executions and multiple participants. Indeed, some messages are executed more than once since they can be part of a multi-instance task or a multi-instance participant. Analysing the results of the experiment, the higher cost is related to the deployment of the generated smart contract, with a total of 3,324,906 gas used. The message executions instead are rather cheap, as they consume a minimum of 50,319 units of gas for the second execution of *retail quotation* message to a maximum of

[2]The executed smart contract is available at `https://mumbai.polygonscan.com/address/0xF053e29fd2D82ABAf84f4583A039Cb746cfE6038`

206,711 units of gas for the *payment0* message. This variation is given by the activation of gateways that can follow a certain message. In this case, indeed, a transaction will also include the execution of the gateway, thus increasing the cost. However, the proposed approach can be considered feasible in terms of costs, since also the aggregate measures are quite affordable. Indeed, the total amount for the deployment and message execution is 5,058,052 units of gas. Furthermore, the total cost has to be divided among the involved participants depending on the executed function.

6.4 Comparison with Existing Approaches

The support of multiple instances in BPMN is a topic that has encountered many proposals and tools [41, 39, 40]. However, most of them target Collaboration diagrams without considering Choreographies. Moreover, this is still unexplored when blockchain is used for trusted execution.

In [47] the authors propose an approach to model complex distributed systems dealing with the multiplicity of role instances and service sessions. To this purpose, the choreographed behaviour is represented using UML activity diagrams. To support the multiplicity aspects, multiple roles are indicated inside the activity diagram while multiple activities are identified by a separate model. Finally, to specify the overall sequence of actions, a flow-global choreography relying on UML is used. For what concerns the modelling aspect, MIChain aims at supporting the modelling and execution of choreographies using BPMN standards. Indeed, BPMN allows defining in a single model the distributed coordination between components and their multi-instance attributes. For this reason, the focus is on supporting multiplicity by extending the BPMN standard, without the need for additional elements or models. In [70] the authors propose the automatic detection of synchronisation points of choreographed models. In this case, the overall choreography is derived by (i) the process model, (ii) the data objects, and (iii) their conceptual model. To manage multi-instance activities, loop and parallel activities are supported and implemented by extending *Activity*^{TM}. Also in this case MIChain differs from the modelling aspect as it considers a single BPMN choreography containing all the information needed to represent multi-instance concepts. Moreover, MIChain also considers blockchain as a trusted execution environment. Support for multiple instances elements on blockchain-based settings was proposed in the Caterpillar engine [75] already described in Section 3.4. In this work, the authors include sequential and parallel multiple-instance activities such as call activities and sub-processes. In those cases, Caterpillar generates a separate Solidity contract to encode the multi-instance sub-process. This contract is also instantiated once for each specified instance. The main difference with this thesis is the types of treated BPMN elements and their translated blockchain behaviour. Indeed, the authors consider BPMN process model multi-instance elements as separate sub-processes that are reusable many times. MIChain instead, considers BPMN choreography elements and, in particular, multi-instance tasks and participants which permit the expression of different behaviours. This affects also the smart contract generation which encodes the multi-instance behaviour directly in the main smart contract. The resulting code regulates the users' invocations and the execution of messages.

CHAPTER 7

MULTICHAIN: SUPPORTING PRIVACY AND CONFIDENTIALITY

In this chapter, the thesis faces the privacy and confidentiality issues of blockchain technology. There are scenarios where an inter-organisational business process execution best fits with the demand for having pre-defined participants and restricted access to data so as to avoid, for instance, the disclosure of sensitive information. This results in a need for a permissioned environment, in which communications are isolated and only the interested parties have access to their data.

For this purpose, ChorChain was extended to fit different requirements and multiple blockchain technologies. This results in a more abstract engineering methodology, supported by a practical framework, named MultiChain that starting from the same high-level specification permits generating the low-level code specific for different blockchain platforms (Ethereum and Hyperledger Fabric). MultiChain was practically integrated into ChorChain making the overall approach reusable in a broader range of application scenarios. Notice, MultiChain does not aim at combining the different technologies in cross-chain communication, but it provides an isolated and alternative environment for executing business agreements with different requirements.

The following section introduces the MultiChain approach highlighting the differences and the extensions starting from ChorChain. In particular, the conceptual description of the translation mechanism that produces network artefacts and chaincode from a choreography is provided. Similarly, the resulting tool and the novel functionalities are introduced. Then, the

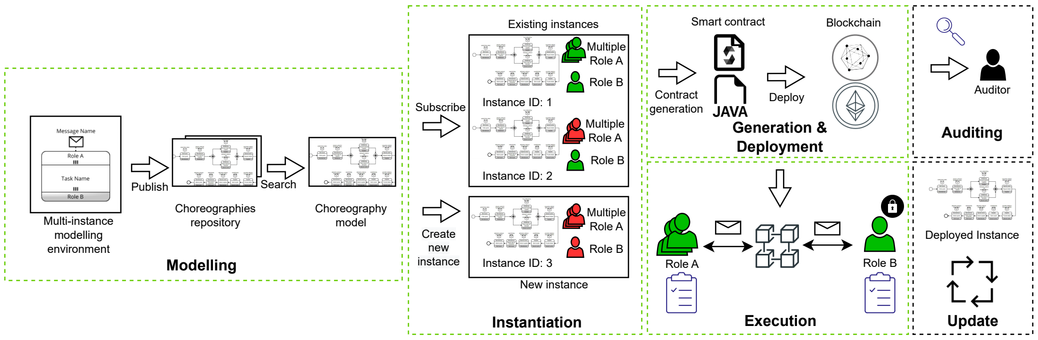


Figure 7.1: MULTICHAIN framework: supporting privacy and confidentiality.

experiment results on the running example are reported and a final overview of related works is given.

7.1 MULTICHAIN Conceptual Framework

This section presents the MULTICHAIN framework and the introduced extensions for supporting privacy and confidentiality during the execution of inter-organisational business processes. Thanks to the proposed infrastructure, MULTICHAIN targets trust concerns related to the underside discoloured of business data. This aspect is a strong contributor to the trustworthiness of the system [83].

7.1.1 Framework phases

The MULTICHAIN framework is depicted in Figure 7.1 and it extends some CHORCHAIN principles deriving the main phases, involved actors and components. To achieve privacy and confidentiality, MULTICHAIN includes a different kind of blockchain setting, thus affecting mainly the generation phase and the successive execution.

The initial **modelling** phase foresees the creation of a choreography diagram representing the business process. This model is later used for generating the low-level code that is different according to the selected blockchain. In this context, the first benefit of MULTICHAIN is the possibility of starting from a single model without requiring any extra information. This makes it possible to clearly separate the conceptual modelling from the code generation, simplifying the operation for creating the software infrastructure. Also in the **instantiation** phase, there are no significant operations that the developer has to perform. At this stage, only a simple selection of the final setting is needed. The **blockchain selection** phase instead, allows the developer to choose the desired blockchain infrastructure to be generated, depending on the requirements of the starting process described in section 2.1.3. At this point, the translator generates blockchain-specific artefacts. In the case of a permissionless setting, a smart contract is generated. When instead, a permissioned setting is chosen, MultiChain generates a different smart contract and also a network infrastructure including channels and private data repositories. In this way, in the **execution** phase, the interacting parties can benefit from private data and confidential communications. The exchanged data is indeed not exposed to external auditors that have no knowledge about the happening interactions.

7.1.2 Hyperledegr Fabric artefacts generation

In Ethereum, the instantiation of a choreography diagram leads to the generation of a single smart contract to be deployed on the public network. This contract explicitly includes the users' addresses that subscribed to the roles in the previous phase. In such a way once deployed the smart contract can be used only by the subscribed users, and every functionality is then enforced both considering the order of the operations and the roles, as specified in the choreography model.

In Fabric, the situation is rather different since there is no global network and so, for each deployment, it is necessary to create not only the smart contract (also called chaincode) but also the network infrastructure. In particular, in MultiChain, any choreography instance is represented by a Fabric channel, and each role is associated with a unique Fabric organisation. Each channel is composed of organisations, representing instance participants that interact with the chaincode representing the choreography instance behaviour.

When creating a channel, any user subscribed to a specific role becomes a member of the organisation representing that role in a specific choreography. Technically, the user is associated with the organisation through an identity released by exploiting cryptographic artefacts. To identify specific users, an attribute-based access control strategy is adopted. This encodes an attribute representing each user member's identity in the organisation, used to restrict the visibility of data on the deployed chaincode. This mechanism guarantees the privacy of exchanged information between different users covering the same role on two different instances that are both included in the same organisation. Indeed, in this way, each user can see only data related to the contracts in which is directly involved.

In addition to identities, each couple of interacting participants (i.e., the sender and the receiver of a task) also generates private data collection. This permits a subset of organisations to share in a private manner the exchanged data without exposing it to the other involved parties. On the contrary, a public collection (i.e., the world state) keeps in memory all the information shared by all the choreography participants.

To generate all of these artefacts, in MultiChain, an automatic procedure instantiates, following the model specification, the consortium and the organisations involved in the Fabric network. Then, every time a participant user subscribes to a role, the corresponding identity is created and added to the private network. When all the roles of the instance are fully covered, the deployment phase can start. Here the translator, according to the instance type (Ethereum or Fabric), generates the specific smart contract and deploys it in the respective network.

7.1.3 Translation approach: from BPMN to Javascript

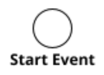
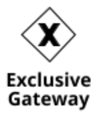
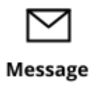
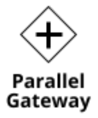
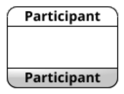
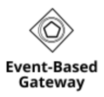
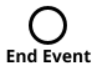
The smart contract generation is an automatic phase where the choreography instance is translated into code. The resulting smart contract permits the participants to interact according to the corresponding choreography protocol. In MultiChain, the generated code for the Fabric technology is based on Javascript and is reported in Table 7.1. The logic behind the code generation for the control flow and message elements is similar to ChorChain since it is directly derived by it. However, the generation foresees the introduction of novel and specific methods for each message exchanged and the introduction of mechanisms to support confidentiality and privacy. This asks to derive a complex transformation procedure for the definition of specific users' rights and their control, and the introduction of a private state for storing transaction data.

In general, the contract contains a common part with information on participants and elements. Additionally, this header contains the referrals to the previously created private collections.

The smart contracts generation continues by appending the functions corresponding to the translation of the elements included in the choreography model. Also, in this case, the concept of choreography *task* is concealed in favour of the connected messages. In particular, a one-way choreography task is represented by its message, and similarly, the two-way task is represented by its two messages. *Control flow* elements represent instead the sequence of execution.

In MultiChain, all the mentioned elements are represented with a JavaScript asynchronous function which logic remains similar to ChorChain, with the inclusion of checks on the execution states. The novel aspects regard the access control mechanism and the storage of data. Indeed, in a Fabric chaincode, a double check is done on participants' rights. The first is based on the caller participant whose identity has to correspond

Table 7.1: Translation approach from BPMN elements to Javascript.

BPMN element	Javascript code	BPMN element	Javascript code
1) 	• Private function • Check on its state • Activates next element	5) 	• Private function • Check on its state • Activates next element
2) 	• Asynchronous message function • Check on its state • Activate next element	6) 	• Asynchronous function • Check on its state • Activates next elements
3) 	• Private Data Collection • Identity Access Control	7) 	• Asynchronous function • Check on its state • Activates next elements
4) Sequence Flow	• Guard expression	8) 	• Asynchronous function • Check on its state

to a member of the organisation having the rights for executing the function. The second one examines the user's encoded attribute inside the identity certificate. This step allows to verify not only if the organisation corresponds to the right one but also if the specific user covers the right role. The second aspect is the introduction of private data collections, which are previously created and involve only subsets of subscribed participants. In this state, the information sent and received by a sender and a receiver is kept in memory. On the contrary, the public data state contains all the data shared by all such as a decision of a gateway and the standard execution flow. Control flow elements indeed follow the same logic of ChorChain and their main role is to enable the successive functions evaluating conditions.

7.2 MultiChain Tool

This section describes the implementation of the MultiChain tool and the main introduced functionalities for supporting a permissioned blockchain setting relying on Hyperledger Fabric. The tool is integrated into the ChorChain implementation, thus this section resumes the main concepts and focuses on the new technicalities[1].

[1]The reader can practically experiment with the framework deployed at `http://virtualpros.unicam.it:8080/MultiChain/`.

7.2.1 Modelling

The modelling phase is the starting point of the choreography creation. Before accessing it, it is necessary to register and login into the platform. These operations require a name and a password to create the user's identity inside the platform. These are only preliminary high-level credentials; for identifying the user inside the blockchain processes, an Ethereum address or a Fabric identity is later assigned. At this point, the user can access the modelling environment integrated into the framework as shown in Figure 7.2. This offers several functionalities, such as the creation, the import, the export and the saving of a model in the MultiChain repository.

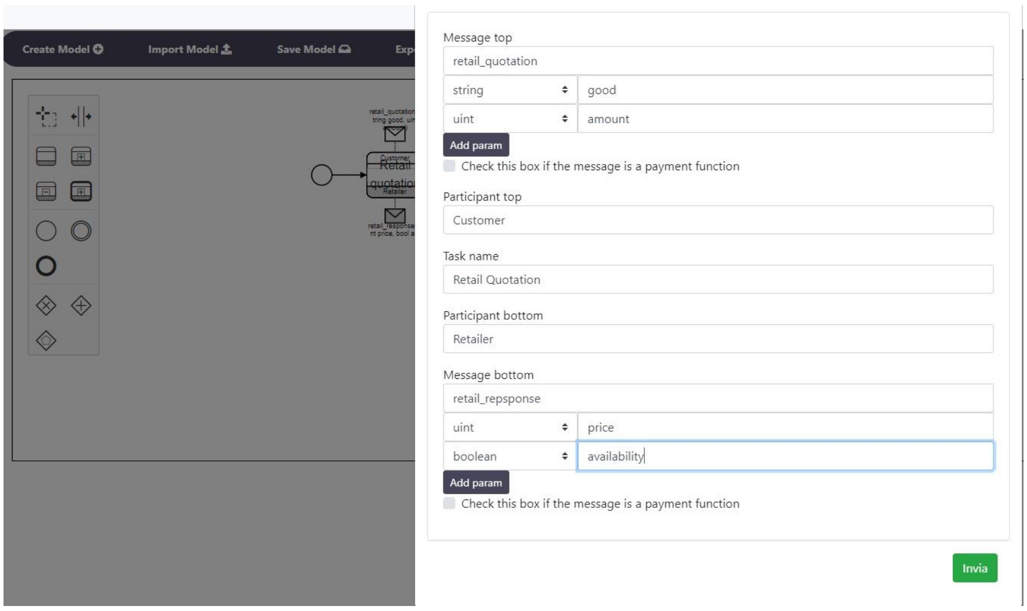
Due to its intended abstraction level, a choreography model does not include enough details to enable an automatic generation of code directly. For this reason, the modelling environment asks the developer for additional data about *(i)* messages and *(ii)* guards as described in Section 3.2. This data is needed to permit the deployment of blockchain infrastructure and it is valid also in the case of Fabric, without requiring any differentiation. Notice, in the case of Fabric, the absence of a native cryptocurrency does not allow the modelling of financial transactions.

Retail Process Example (Modelling) Considering the proposed running example, Figure 7.2 reports the panel in the MultiChain modelling environment for the definition of the *Retail quotation* task using a dedicated panel. In this case, the information inside the *Retail quotation* task is provided in Ethereum-like format but it is translated according to the run-time user decision. The created choreography can be indeed exploited also for generating a Fabric infrastructure, without requiring additional specifications.

7.2.2 Instantiation

This part describes the multiple blockchains support for instantiating a choreography specification. In particular, after the instantiation phase, the distinction between the two resulting artefacts to be deployed on a specific blockchain is evident, while till the publishing phase, the model is unique. Once a choreography is published into the MultiChain repository, it is visible in the homepage depicted in Figure 7.3, which shows the uploaded and instantiated retail process example. Here it is possible to create a new instance by clicking the dedicated button. In addition to the selection of mandatory and optional roles, MultiChain requires selecting if the instance to generate targets Ethereum or Fabric.

The resulting instance is then created and, independently from the chosen technology, it shows common information such as the model owner, the maximum number of involved participants and the required roles. Also, the



Message top
retail_quotation
string | good
uint | amount
Add param
Check this box if the message is a payment function

Participant top
Customer

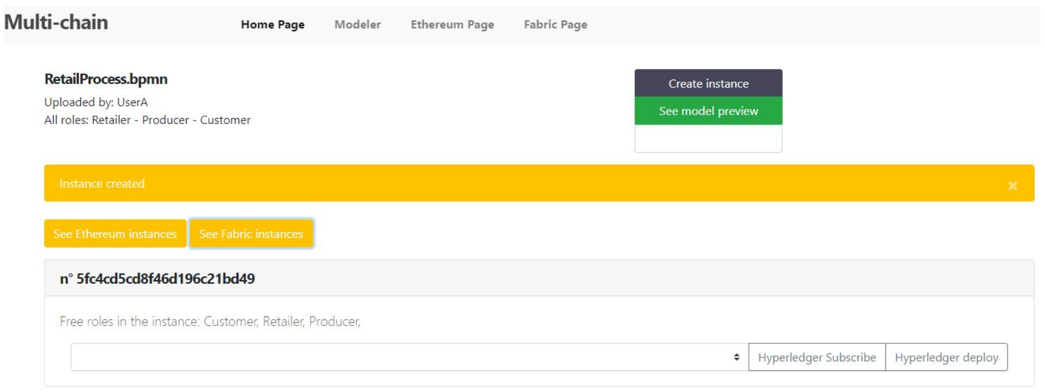
Task name
Retail Quotation

Participant bottom
Retailer

Message bottom
retail_repsponse
uint | price
boolean | availability
Add param
Check this box if the message is a payment function

Invia

Figure 7.2: Multi-Chain modeller.



RetailProcess.bpmn
Uploaded by: UserA
All roles: Retailer - Producer - Customer

Instance created

n° 5fc4cd5cd8f46d196c21bd49

Free roles in the instance: Customer, Retailer, Producer,

Figure 7.3: MultiChain homepage with a focus on Fabric instances.

preview of the graphical representation and the possibility of creating a new choreography contract is available.

Before deploying one of the two possible instances, the choreography participants must be filled by the users during the subscription phase. For Ethereum, when the user subscribes to a role, it's necessary to associate the Ethereum address through the Metamask browser plugin. For Fabric, the procedure is quite different since the user's identity is directly created after the subscription. Indeed, roles are associated with Fabric organisations so MultiChain automatically generates the artefacts for the user's identity that become a member of the organisation covering a certain role in a specific instance. Thus, the interface is necessary only to select the desired role, without the need for additional operations.

When one of the two choreography instances has no more vacant mandatory roles, the partnership is complete, and the smart contracts generation phase can start, deploying it on the chosen blockchain. If the contract has some optional roles, the subscription form remains enabled on the homepage with only the optional roles, also after its deployment.

Retail Process Example (Instantiation) Considering the retail process example, during the instance creation, depicted in Figure 7.3 the *Customer* and *Retailer* are mandatory roles. The *Producer*, instead, is set as optional, since its participation is not always required.

7.2.3 MultiChain Translator

Another peculiarity of the MultiChain framework is the blockchain infrastructures that are derived by means of the model-driven approach. This is the generation phase that takes place after all the mandatory users subscribe to an instance. In this section, a particular focus is on the translation from model to code, i.e., the contracts and network creation. Indeed, Ethereum and Hyperledger Fabric have been conceived with rather different objectives and usage scenarios. Therefore, the transformation from a choreography diagram to the blockchain infrastructure to be deployed over the specific technology is different.

The rest of the section describes the technical differences between the translation of an Ethereum contract described in Section 3.2.3 and a Fabric one by providing the relative examples using the retail process scenario described in Section 2.2.2.

Smart contract generation The generation of the code starts after the parsing of the choreography model performed using the Camunda library, properly extended to deal with the choreography diagrams syntax as defined in the standard.

Listing 7.1 shows the template for the header of the chaincode for Hyperledger Fabric (ChoreographyPrivateDataContract). In this case, it can be noted the introduction of two utility classes: (i) ChorographyState and (ii) ChoreographyPrivateState.

```
269 const chorID = '68e81c58-2ca9-4a92-b438-76f06f358fa3'
270 const contractName = 'contracte3158a2b-40b7-43b0-9ae2-d19dacb39839'
271 const Status = { DISABLED: 'disabled', ENABLED: 'enabled', DONE: 'done'
        };
272 const chorElements = ["StartEvent_102vawy", "ExclusiveGateway_042aut8",
        "Message_0b917rc", ...]
273 const roles = { Customer: 'Org1MSP5fe1cdac280183175ccb152e', Retailer:
        'Org2MSP5fe1cdac280183175ccb152e', Producer:
        'Org3MSP5fe1cdac280183175ccb152e'}
274 const collectionsPrivate = {CustomerRetailer: 'collection' +
        roles.Customer + roles.Retailer, ...}
275 const subscriptions = { Customer: '5fe0b7aa2801833b2c91a2d3', Retailer:
        '5fe0b82f2801833b2c91a2e1', Producer: '5fe1ce56280183175ccb153a'}
276
277 class ChoreographyPrivateDataContract extends Contract {
278     constructor() {
279         super(contractName)
280     }
281
282 }
```

Listing 7.1: Hyperledger: ChoreographyPrivateDataContract class

In Fabric, the contract is defined through the `ChoreographyPrivate-DataContract` class. Like Ethereum, the class keeps tracking each element of the choreography within an associative object. Also, in this case, the element life-cycle is composed of three states, listed in the `Status` object (Line 271). The `chorElements` (Line 272) object maps choreography elements to their individual state, and it is included in the ledger state. Additional information like the contract name and the id of the choreography are reported in Lines 269-270. The roles declared in Line 273 map choreography roles to the Fabric Membership Service Providers belonging to the related organisations to guarantee confidentiality. In Line 274, the definition of the private collections is done by coupling all the different roles of the model inside the `collectionsPrivate` object. Also, in Line 275 the roles are associated with the subscribed users, which identities were previously created inside the organisations.

Just after the header, in the contract an access control function is introduced to define the participants in a message exchange. In fabric, this is done by using an identity check.

```
283 roles.Customer === ctx.stub.getCreator().mspid &&
        ctx.clientIdentity.assertAttributeValue('role',
        subscriptions.Customer)
```

Listing 7.2: Hyperledger: Enforcing controls

This check is reported in Listing 7.2 and it consists of two main controls before executing a message. This is done by checking the MSP of the transaction creator, with respect to the `roles` list defined in the contract header. In the second check, the identity certificate of the caller is exploited. In this way, the caller attribute is first retrieved and then compared to the id of the right role inside the `subscriptions` object.

Listing 7.3 shows the implementation of a message in Fabric. In Line 285 the actual public state of the choreography is retrieved from the external utility class `ChoreographyState` and it is used for the next operations. Line 286 reports the controls performed before allowing the execution of the message. In the condition of the conditional statement, there is the check of the status of the actual message, identified by its `Message_id`. This permits the enforcement of the execution of the right sequence of functions. The other two expressions in the condition are related to the check of the right user and organisation as described in listing 7.2. If the user is the right one, the private state associated with him is recovered (Line 287), calling the external class `ChoreographyPrivateState`. This state concerns the participants' interaction in which only the information changed between them is stored. In particular, to get the private state some information must be passed: (i) the Fabric ctx, (ii) the private collection between the sender and the receiver of the message and (iii) the id of the choreography, automatically set by the translator in the generation phase. The actual message is then set as *Done* and the next one is *enabled* (Lines 288-289). Finally, the public and the private state of the choreography are updated (Lines 290-291). In particular, these operations are done without passing directly the information inserted by the user but exploiting information stored in the context object (ctx). Indeed, the ctx encapsulates the transient data that are then extracted in the invoked functions, in this way, are not explicitly visible in these operations.

```
284   async Message_0b917rc(ctx) {
285       const choreography = await ChoreographyState.getState(ctx, chorID)
286       if(choreography.elements.Message_0b917rc === Status.ENABLED &&
              roles.Customer=== ctx.stub.getCreator().mspid &&
              ctx.clientIdentity.assertAttributeValue('role',
              subscriptions.Customer)) {
287           const choreographyPrivate = await
                  ChoreographyPrivateState.getPrivateState(ctx,
                  collectionsPrivate.CustomerRetailer, chorID)
288           choreography.setDone('Message_0b917rc')
289           choreography.setEnable('Message_1xxdwx2')
290           await choreographyPrivate.updatePrivateState(ctx,
                  collectionsPrivate.CustomerRetailer)
291           await choreography.updateState(ctx)
292           return { choreography, choreographyPrivate }
293       } else {
294           throw new Error('Element Message_0b917rc is not ENABLED or
                  submitter not allowed, only the Customer can send this
                  transaction')
295       }
296   }
```

Listing 7.3: Hyperledger: Message Function

Also in the case of gateways, there is a similar implementation. Here below, only the *exclusive gateway* implementation is shown since there is a similar structure also for all the other gateways.

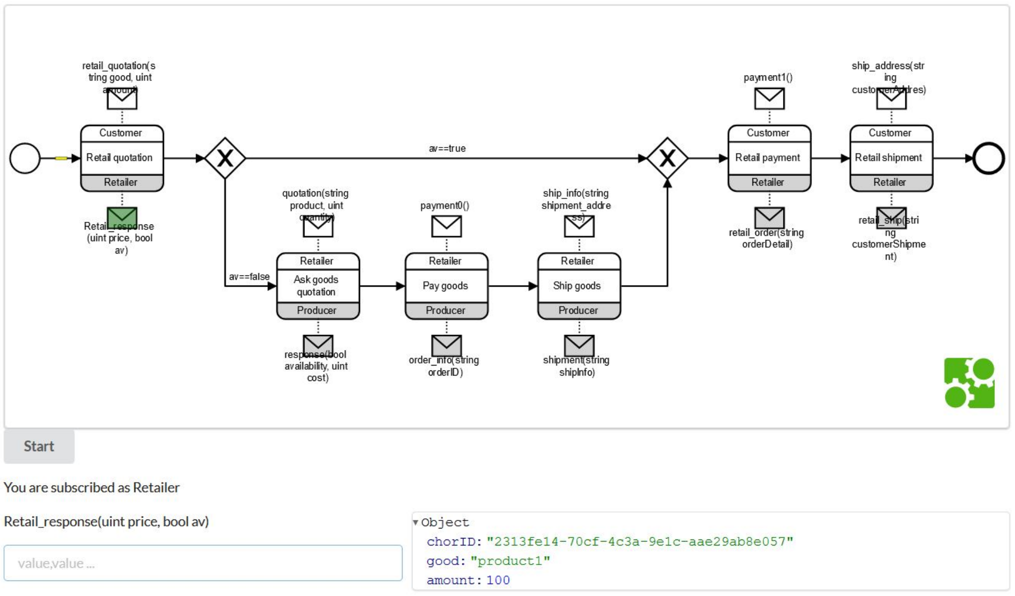
Listing 7.4 shows the code used inside the Fabric contract. Firstly the status of the actually invoked gateway identified by `Gateway_id` is checked (Line 298). Then its status is set to done (Line 299), and the evaluation of the variable is performed. Depending on its value, the function enables the next element to be a message or a gateway, identified by its id. At this point, if the next element is a message, the public state is updated by calling the external function `updateState` that inserts the new status of the elements (Line 302); otherwise, it is directly called (Line 305), and the public state is updated in the next functions.

```
297 async ExclusiveGateway_042aut8(ctx, choreography, choreographyPrivate)
        {
298     if(choreography.elements.ExclusiveGateway_042aut8 ===
            Status.ENABLED) {
299         choreography.setDone('ExclusiveGateway_042aut8')
300         if(choreographyPrivate.av==false) {
301             choreography.setEnable('Message_1h3ew61')
302             await choreography.updateState(ctx)
303         } else if(choreographyPrivate.av==true) {
304             choreography.setEnable('ExclusiveGateway_1johog7')
305             await this.ExclusiveGateway_1johog7(ctx, choreography,
                    choreographyPrivate)
306         }
307     } else {
308         throw new Error('ExclusiveGateway_042aut8 not ENABLED')
309     }
310 }
```

Listing 7.4: Hyperledger: Exclusive gateway implementation

7.2.4 Deployment

Once the contract has been generated, the framework automatically deploys it into the selected blockchain. Depending on the chosen technology, the deployment operation is different. For Ethereum, the server generates a transaction that deploys the generated Solidity contract on the blockchain. For Fabric, the procedure is more complicated since, for each new instance, a new channel must be created. This happens in the back-end, where the system automatically generates the artefacts related to organisations, orderers, channels and chaincodes. In particular, organisations' artefacts are produced after the model publishing, users' identities in the subscription phase, chaincode and the channels in the deployment.


retail_quotation(string good, uint amount)
Customer
Retail quotation
Retailer
Retail_response(uint price, bool av)
av==true
av==false
quotation(string product, uint quantity)
Retailer
Ask goods quotation
Producer
response(bool availability, uint cost)
payment0()
Retailer
Pay goods
Producer
order_info(string orderID)
ship_info(string shipment_address)
Retailer
Ship goods
Producer
shipment(string shipInfo)
payment1()
Customer
Retail payment
Retailer
retail_order(string orderDetail)
ship_address(string customerAddres)
Customer
Retail shipment
Retailer
retail_ship(string customerShipment)
Start
value,value ...

You are subscribed as Retailer

Retail_response(uint price, bool av)

```
▾Object
  chorID: "2313fe14-70cf-4c3a-9e1c-aae29ab8e057"
  good: "product1"
  amount: 100
```

Figure 7.4: Fabric execution page.

7.2.5 Execution

Once a new contract is deployed into the blockchain, the execution phase takes place, and the participants can collaborate using the functions exposed by the contract. To facilitate these interactions, there are two execution pages accessible to each participant. These pages enable interaction with Ethereum contracts or with Fabric ones.

Figure 7.4 shows the Fabric page concerning the deployed retail process example. However, this execution page has some common characteristics with the Ethereum one. On the left-hand side, the interface reports a list of all contracts to which the participant is subscribed. On the right-hand side, a preview of the model is shown: in green, the messages completed are indicated, and the ones actually active. For these, the window also includes the forms that are dynamically constructed by the tool. Each form contains many information, like the name of the message, the role of the participant, the space for inserting all the required parameters, and the submit button. Notably, the submission form is visible only to the participant in charge of sending the enabled message.

By double-clicking on a completed message, a little panel with the exchanged values is shown. However, for Ethereum, the data will be visible to all the participants of the process. In Fabric instead, since privacy is one of the main requirements, the exchanged information will be directly visible only to the sender and receiver participants. When an Ethereum message is sent, the transaction has to be confirmed using the Metamask pop-up.

It contains the gas price plus the total amount of Ether to spend for the transaction. As soon as the transaction is included in a block, the related event is emitted. The front-end uses this event to update the interfaces of all participants involved in the choreography with the new contract status, thus enabling the next admitted message(s). In Fabric instead, a function execution does not require the payment of any fee, and in general, the transaction process is faster. It is worth noticing that the choreography is executed in a distributed manner, since the participants interact via the front-end directly with the blockchain, without referring anymore to the back-end component.

Retail Process Example (Execution) The example in Figure 7.4 depicts the execution page for the *Retailer* participant involved in the Fabric network. The *retail_response* message in green on the model represents the enabled activity that can be executed by the Retailer. On the right side of the page instead, the data panel reports the previously sent message. In particular, the panel lists the *good* and the *amount* parameters sent by the customer. The *Retailer* at this point answers using the dedicated form, inserting the availability as a boolean value and its cost. It is important to notice that in the Fabric case, the quotation information is visible only to the Retailer and only inside this process. This means that the Producer since it only interacts with the Retailer, has no vision of the data exchanged between the Customer and the Retailer. In the same way, external observers have no views about the current state of the process.

7.3 Experiments and Validation

This section reports the results of the experiments made on MultiChain. In particular, the permissionless infrastructure is the same as ChorChain, evaluated in Section 3.3. For this reason, this section focuses on assessing the performances of Fabric translation, deployment, transaction execution and network creation.

The first experiment is reported in Figure 7.5 and shows the times the translator needs to generate 10 smart contracts for the Fabric blockchain, derived from the running example model. The average time for the translation is 24 ms with a minimum of 18 ms and a maximum of 31 ms. These measures can be considered efficient since they depend on the MultiChain translator and the target code does not have a real impact on the generation's performance.

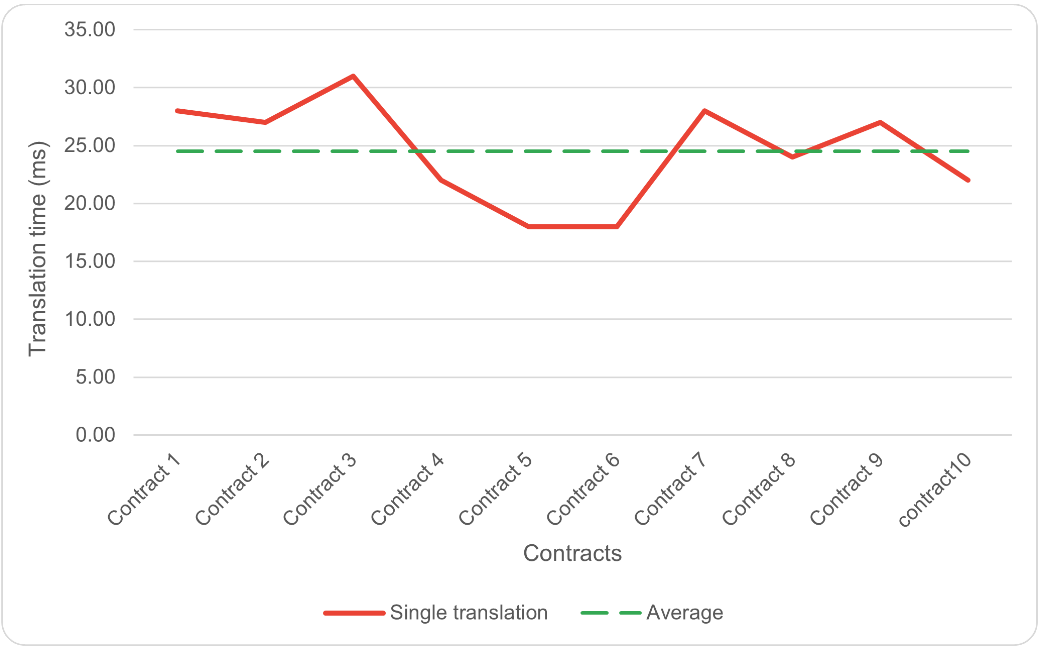


Figure 7.5: Translation time of 10 running example choreography instances.

The next experiment is reported in Figure 7.6 and represents the deployment time of the contracts previously created. In this case, the average measured time is 79 s with a minimum of 75 s and a maximum of 87 s. Certainly, the deployment of a Fabric smart contract is less efficient if compared to an Ethereum one, but this mainly depends on the different blockchain deployment procedures. Indeed, Ethereum relies on consensus protocols that make transactions included in certain slots. In Fabric instead, each peer of the network has to approve and verify the contract following a sequence of stages. This procedure can depend on different variables such as the number of network peers, channels, and the used hardware, making the deployment process slower on average.

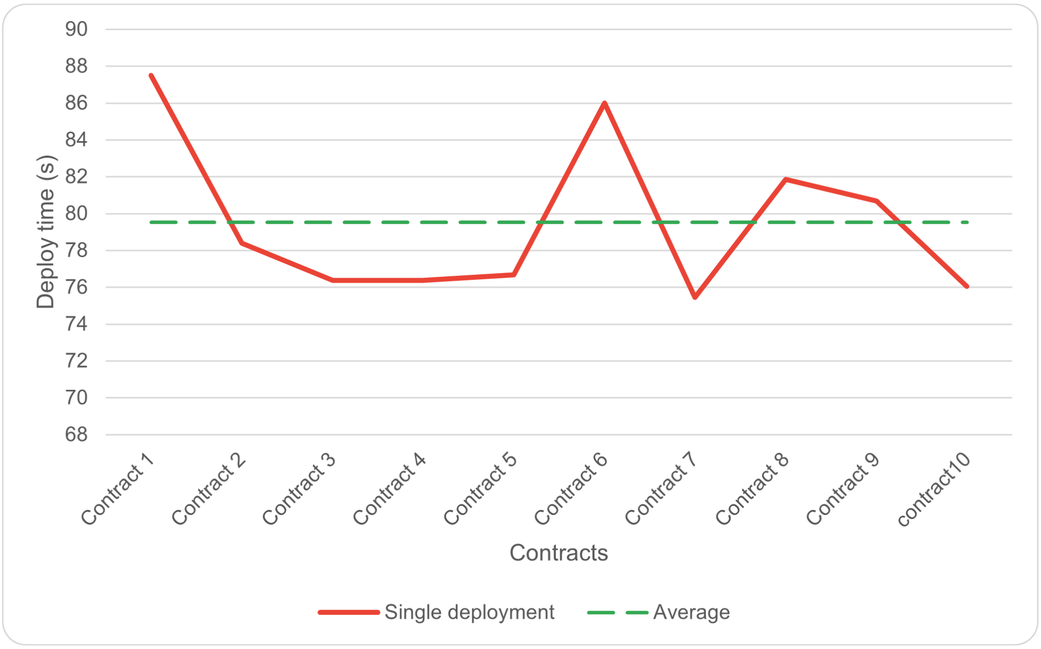


Figure 7.6: Deploy time of 10 running example choreography instances.

Figure 7.7 compares the average time required for executing a specific transaction of the running example. Each transaction was executed 10 times using the smart contracts previously created and deployed. Here the impact of using Fabric is more evident.

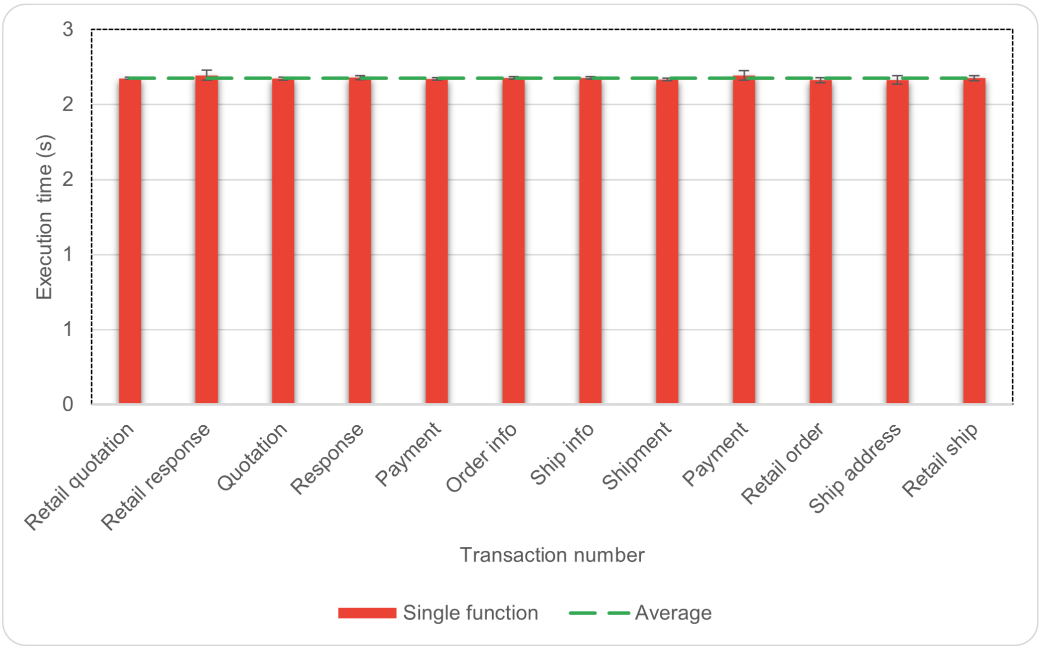


Figure 7.7: Average transactions execution time for the running example.

Indeed, the execution time is much more regular and there are no significant differences between separate transactions. In general, the average time necessary corresponds to 2 s with a standard deviation that is not relevant. This behaviour depends on the different consensus needed to approve a state change. In Ethereum, as for the deployment, each smart contract execution has to pass through the blockchain consensus, while in Fabric, the execution of a function has a much lighter process.

The last experiment in Figure 7.8 reports the performance of the network creation process. This concept is valid only for Fabric since Ethereum is based on a public network that is already available and is maintained from the various nodes. On the contrary, Fabric is based on a private network; therefore, for each new choreography, it is necessary to create a new preconfigured network considering the involved organisations and successive participants' identities. In the experiments, 10 different network instantiations were isolated and observed, with an average generation time of 20 s with a minimum of 17 s and a maximum of 22 s. This additional time should be considered one-time only during the first upload of the choreography so it does not significantly affect the usage of MultiChain.

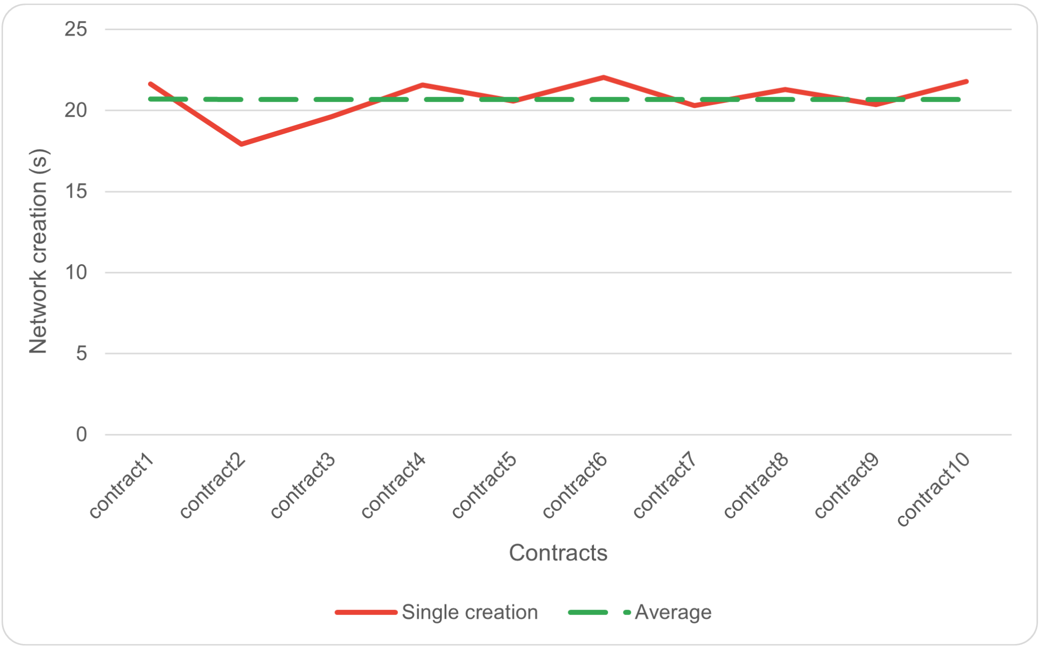


Figure 7.8: Time required for creating a Fabric network.

To resume the above-described experiments, it is possible to notice that having a private permissioned infrastructure like Fabric, requires an additional set-up phase including the creation of the network, channels and identities that influence the overall MultiChain performance. In a public permissionless context like Ethereum instead, the use of a public network avoids the mentioned steps. However, the trend is different during the smart contract execution, in which the use of the Ethereum consensus protocol highly impacts the performance. Differently, the Fabric execution phase exploits a lighter peers agreement which makes it possible to create more scalable applications thanks to the faster function execution.

7.4 Comparison with Existing Approaches

Considering the model-driven engineering of systems with a focus on confidentiality, this is still an open topic in the literature. In [95] the authors analyse the implementation of BPMN collaboration processes focusing on the privacy of the information exchanged. In particular, Fabric Composer is used to create a network with the involved participants, while the information is regulated by the native access control of Fabric. However, starting from the model, there is no automatic generation of components, and, actually, Composer is deprecated in favour of a new version of Fabric. A permissionless solution is instead considered in [29, 28], where BPEL processes are translated into smart contracts with a particular focus on solutions to ensure data confidentiality in the presence of an untrusted oracle. The proposed solution presents the use of confidential smart contacts that encrypt sensible data, enabling only allowed participants to use it. In [129] the authors propose the ShadowEth system prototype. It focuses on privacy during the execution of smart contracts, which they assume is more important than the privacy of the entire blockchain. For this purpose, they separate public chain and private smart contracts and define a protocol between them. In particular, they take advantage of the public blockchain to ensure the availability and integrity of the data, while the hardware enclave is used to guarantee privacy. In addition, requirements for building security around information exchanged are highlighted in [98] NASA Ames Research Center: i) Participants must be identified/identifiable; ii) Networks are required to be permissioned; ii) High transaction throughput performance; iii) Low latency of transaction confirmation; iv) Authentication, privacy, and confidentiality of tracking and communication between clients and providers are consistently supported. These requirements emerged from the adoption of the Hyperledger Fabric permissioned blockchain that is exploited to preserve the privacy of radar-based air traffic service for military and corporate operations. Another important aspect highlighted by the author is the private channels service that Hyperledger Fabric provides as a means to communicate private information at a comparatively high bandwidth.

PART III

CONCLUSIONS & FUTURE WORKS

CHAPTER 8
CONCLUSIONS & FUTURE WORKS

8.1 Conclusions

The thesis tackled different challenges concerning inter-organisational business processes. In particular, the first challenge is related to the trust required to execute the interactions defined as a process between unknown parties in a distributed manner. Trust is indeed a fundamental aspect in a collaborative scenario which is perceived differently by organisations. This perception is mainly influenced by all those attributes concerning the used systems and interactions among participants. To this purpose, blockchain was used as enabling technology, thanks to its property of security, distribution, transparency, immutability and enforcement. From this, other relevant challenges were derived and they are related to auditing, flexibility, multiplicity, confidentiality and privacy. To face those aspects, the thesis defines some relevant research objectives, proposing different solutions relying on a common model-driven engineering methodology.

The first objective encounters the requirement of trust, the thesis first introduced ChorChain, a novel model-driven methodology which relies on a blockchain infrastructure to enable the execution of inter-organisational business processes. The methodology's starting point is the BPMN standard and in particular choreography diagrams, which make it possible to represent inter-organisational business processes from a high-level perspective. The methodology is concretely supported by an implemented framework integrating mechanisms for the management of choreographies from their modelling to their execution. Once a choreography representing a system is created, it is executed over the blockchain, taking advantage of its characteristics for providing trust. A relevant aspect of the proposed framework is the translation of a choreography specification into a smart contract, once the participating partners have been identified. The smart contract is then deployed on the blockchain infrastructure and drives the choreography execution. Interactions are enabled according to the choreography specification, and when performed the details related to their occurrence are stored in the blockchain.

The capability to observe the data stored in the blockchain enables the possibility of auditing the actual execution of choreography instances, thus further reducing trust-related issues. Indeed, the second research objective refers to auditing, which is an important aspect of the business process since it ensures that exchanged information is represented fairly and accurately. To support this aspect, the ChorChain framework was extended with a complex query mechanism defined on the premises of the conceptual model related to main actors and components. These queries were integrated into the tool enabling internal and external auditors to have a complete view of the executed processes.

Another objective concerns the flexibility of processes that have to react to unexpected changes and situations while dealing with the immutability of the blockchain. With FlexChain the thesis aims at providing a framework for dealing with run-time flexibility while maintaining the fundamental properties of trust and transparency. FlexChain revises the main ChorChain concepts, introducing a new update functionality. To deal with the immutability of the blockchain, the proposed solution decouples the state of the process (stored on-chain) from the execution logic (stored off-chain). Indeed, while the first is a set of data stored in a smart contract, the second one is expressed as rules and they are stored in the IPFS. This infrastructure allows changing rules at run-time without having to modify the on-chain code that, in other cases, results infeasible.

Considering instead the proposed model-driven methodology, some limitations arise when assessing scenarios with multiple actors performing multiple actions. Indeed, multiplicity is a crucial aspect that enables the creation of complex behaviours while reducing the complexity of the modelling. However, the definition of multi-instance elements in choreographies suffers from a lack in the BPMN standard. Furthermore, the use of blockchain infrastructure for a trustworthy execution requires novel solutions for such aspects. In this context, the thesis proposes MIChain a revised version of ChorChain for the modelling and execution on the blockchain of multiple instance elements and attributes in a choreography. To this purpose, MIChain permits the definition of elements and attributes for handling multiplicity, with their related translation in a blockchain setting.

All of the above-mentioned frameworks aim at facing the related challenges of relying on a public blockchain infrastructure, having transparency and auditability as fundamental characteristics. However, there are situations in which privacy and confidentiality become challenges due to the need of protecting business interactions. For this reason, the thesis introduces the MultiChain framework for supporting use cases from different contexts, thanks to the use of both permissionless and permissioned blockchains, allowing users to have complete coverage of their needs. Depending on the requirements, the proposed implementation supports an Ethereum or Hyperledger Fabric smart contract. Moreover, in Fabric case, the framework automatically constructs the appropriate network according to the organisation's specifications.

To show the current implementations of the frameworks, the thesis provided a Retail process running example, on which several experiments were conducted in order to assess the overall feasibility and performances. In general, the obtained results are encouraging, as they demonstrate that the various frameworks are sustainable in terms of consumed gas. Indeed, the related fees highly depend on the chosen blockchain implementation without limiting the overall validity.

8.2 Future Works

In future works, the first objective is to continue the development of the approaches and frameworks described in the thesis. In particular, this can be done both in terms of single extensions and integration. For this last aspect, in the future, the plan is to implement a single general framework and resulting implementation integrating all the presented frameworks. Indeed, each proposed framework faces a challenge in an isolated way, starting from the ChorChain methodology, but with different implementations and technical choices. For this reason, after having a complete evaluation of the solutions, the next step can be recognised in an overall integration into a single framework.

The second aspect to consider is the optimisation of the performance both in terms of execution time and fee cost. Indeed, while some optimisation was already considered for what concerns the smart contract translation, there is still a need to explore additional technologies. Layer 2 is one of the most prominent filed working for the scaling of blockchain technology. Different solutions are already available for reducing transaction inclusion times and fees. In addition, new blockchain implementations are gaining interest and popularity (e.g., Algorand) thanks to their new algorithms permitting better performances. However, both solutions should be first analysed since in some cases, layer 2 technologies use off-chain storage and computation thus, affecting the overall perceived trust. New blockchains instead are still young and they may lack concrete and efficient support for writing and executing smart contracts.

Appendix A - Questionnaire

Here is reported the System Usability Scale questionnaires administered to students: one concerns the 'Publishing, Subscription, instantiation and execution phases', while the other one the 'Auditing phase'. The results of the validation are discussed in Section 4.3.2.

Question 1 - I think that I would like to use ChorChain frequently.

	Strongly disagree				Strongly agree
Publishing, Subscription, instantiation and execution phases	1	2	3	4	5
Auditing phase	1	2	3	4	5

Question 2 - I found ChorChain unnecessarily complex.

	Strongly disagree				Strongly agree
Publishing, Subscription, instantiation and execution phases	1	2	3	4	5
Auditing phase	1	2	3	4	5

Question 3 - I thought ChorChain was easy to use.

	Strongly disagree				Strongly agree
Publishing, Subscription, instantiation and execution phases	1	2	3	4	5
Auditing phase	1	2	3	4	5

Question 4 - I think that I would need assistance to be able to use ChorChain.

	Strongly disagree				Strongly agree
Publishing, Subscription, instantiation and execution phases	1	2	3	4	5
Auditing phase	1	2	3	4	5

Question 5 - I found the various functions in ChorChain were well integrated.

	Strongly disagree				Strongly agree
Publishing, Subscription, instantiation and execution phases	1	2	3	4	5
Auditing phase	1	2	3	4	5

Question 6 - I thought there was too much inconsistency in ChorChain.

	Strongly disagree				Strongly agree
Publishing, Subscription, instantiation and execution phases	1	2	3	4	5
Auditing phase	1	2	3	4	5

Question 7 - I would imagine that most people would learn to use ChorChain very quickly.

	Strongly disagree				Strongly agree
Publishing, Subscription, instantiation and execution phases	1	2	3	4	5
Auditing phase	1	2	3	4	5

Question 8 - I found ChorChain very cumbersome/awkward to use.

	Strongly disagree				Strongly agree
Publishing, Subscription, instantiation and execution phases	1	2	3	4	5
Auditing phase	1	2	3	4	5

Question 9 - I felt very confident using ChorChain.

	Strongly disagree				Strongly agree
Publishing, Subscription, instantiation and execution phases	1	2	3	4	5
Auditing phase	1	2	3	4	5

Question 10 - I needed to learn many things before I could get going with ChorChain.

	Strongly disagree				Strongly agree
Publishing, Subscription, instantiation and execution phases	1	2	3	4	5
Auditing phase	1	2	3	4	5

BIBLIOGRAPHY

[1] Pedro W Abreu, Manuela Aparicio, and Carlos J Costa. Blockchain technology in the auditing environment. In *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6. IEEE, 2018.

[2] Michael Adams, Suriadi Suriadi, Akhil Kumar, and Arthur HM ter Hofstede. Flexible integration of blockchain with business process automation: A federated architecture. In *Int. Conference on Advanced Information Systems Engineering*, volume 386 of *LNBIP*, pages 1–13. Springer, 2020.

[3] Ashar Ahmad, Muhammad Saad, Mostafa Bassiouni, and Aziz Mohaisen. Towards blockchain-driven, secure and transparent audit logs. In *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 443–448. ACM, 2018.

[4] Ashar Ahmad, Muhammad Saad, Laurent Njilla, Charles Kamhoua, Mostafa Bassiouni, and Aziz Mohaisen. Blocktrail: A scalable multichain solution for blockchain-based audit trails. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[5] Aitor Aldazabal, Terry Baily, Felix Nanclares, Andrey Sadovykh, Christian Hein, and Tom Ritter. Automated model driven development processes. In *Model Driven Tool and Process Integration*, pages 361 – 375. Fraunhofer IRB Verlag, 2008.

[6] Saiqa Aleem, Sanja Lazarova-Molnar, and Nader Mohamed. Collaborative business process modeling approaches: a review. In *Proc. of the 2012 IEEE 21st International workshop on Enabling Technologies:*

Infrastructure for Collaborative Enterprises, pages 274–279. Citeseer, 2012.

[7] Midhat Ali, Guglielmo De Angelis, and Andrea Polini. Servicepot–an extensible registry for choreography governance. In *7th International Symposium on Service-Oriented System Engineering*, pages 113–124. IEEE, 2013.

[8] Samantha Almeida, Adriano Albuquerque, and Andreia Silva. An approach to develop software that uses blockchain. In *Software Engineering and Algorithms in Intelligent Systems*, volume 763 of *AISC*, pages 346–355. Springer, 2018.

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

[10] Elli Androulaki, Angelo De Caro, Matthias Neugschwandtner, and Alessandro Sorniotti. Endorsement in hyperledger fabric. In *IEEE International Conference on Blockchain, Blockchain 2019, Atlanta, GA, USA, July 14-17, 2019*, pages 510–519. IEEE, 2019.

[11] Deniz Appelbaum and R Nehmer. Designing and auditing accounting systems based on blockchain and distributed ledger principles. *Feliciano School of Business*, pages 1–19, 2017.

[12] Alvin A Arens, Randal J Elder, and Beasley Mark. *Auditing and assurance services: an integrated approach.* Boston: Prentice Hall, 2012.

[13] M. Autili, P. Inverardi, and M. Tivoli. Choreos: Large scale choreographies for the future internet. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering*, pages 391–394, 2014.

[14] Marco Autili, Amleto Di Salle, Francesco Gallo, Claudio Pompilio, and Massimo Tivoli. Model-driven adaptation of service choreographies. In *33rd Annual ACM Symposium on Applied Computing*, pages 1441–1450. ACM, 2018.

[15] Marco Autili, Francesco Gallo, Paola Inverardi, Claudio Pompilio, and Massimo Tivoli. Introducing trust in service-oriented distributed systems through blockchain. In *International Workshop on Governing Adaptive and Unplanned Systems of Systems*, 2019.

[16] Sidechains Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew K. Miller, Andrew Poelstra, and Jorge Timón. Enabling blockchain innovations with pegged. 2014.

[17] Hillol Bala and Viswanath Venkatesh. Assimilation of interorganizational business process standards. *Information Systems Research*, 18(3):340 – 362, 2007.

[18] Aaron Bangor, Philip T Kortum, and James T Miller. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, 24(6):574–594, 2008.

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

[20] Sarah Benyagoub, Meriem Ouederni, Yamine Aït-Ameur, and Atif Mashkoor. Incremental construction of realizable choreographies. In *NASA Formal Methods Symposium*, volume 10811 of *LNCS*, pages 1–19. Springer, 2018.

[21] Egon Börger. Approaches to modeling business processes: a critical analysis of bpmn, workflow patterns and YAWL. *Softw. Syst. Model.*, 11(3):305–318, 2012.

[22] Khoutir Bouchbout and Zaia Alimazighi. Inter-organizational business processes modelling framework. In *ADBIS 2011, Research Communications, Proceedings II of the 15th East-European Conference on Advances in Databases and Information Systems*, volume 789 of *CEUR Workshop Proceedings*, pages 45–54. CEUR-WS.org, 2011.

[23] Abdelaziz Bouras, Houssem Gasmi, Abdelhak Belhi, Assam Hammi, and Belaïd Aouni. Enterprise information systems enhancement: A hyperledger fabric based application. In Asaf Varol, Murat Karabatak, and Ihan Varol, editors, *9th International Symposium on Digital Forensics and Security, ISDFS 2021, Elazig, Turkey, June 28-29, 2021*, pages 1–5. IEEE, 2021.

[24] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[25] Giorgio Bruno, Giulia Bruno, and Marcello La Rosa. On collaborations and choreographies. In *Technologies for Collaborative Business Process Management, Proceedings of the 1st International Workshop on Technologies for Collaborative Business Process Management*, pages 3–12. INSTICC Press, 2006.

[26] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2014.

[27] Christian Cachin et al. Architecture of the hyperledger blockchain fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*, volume 310, pages 1–4. Chicago, IL, 2016.

[28] Barbara Carminati, Elena Ferrari, and Christian Rondanini. Blockchain as a platform for secure inter-organizational business processes. In *Collaboration and Internet Computing*, pages 122–129. IEEE, 2018.

[29] Barbara Carminati, Christian Rondanini, and Elena Ferrari. Confidential business process execution on blockchain. In *Web Services*, pages 58–65. IEEE, 2018.

[30] Fran Casino, Thomas K Dasaklis, and Constantinos Patsakis. A systematic literature review of blockchain-based applications: current status, classification and open issues. *Telematics and Informatics*, 36:55–81, 2019.

[31] Jing Chen, Shixiong Yao, Quan Yuan, Kun He, Shouling Ji, and Ruiying Du. Certchain: Public and efficient certificate audit based on blockchain for tls connections. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2060–2068. IEEE, 2018.

[32] Michele Chinosi and Alberto Trombetta. Bpmn: An introduction to the standard. *Computer Standards & Interfaces*, 34(1):124–134, 2012.

[33] Riccardo Cognini, Flavio Corradini, Stefania Gnesi, Andrea Polini, and Barbara Re. Research challenges in business process adaptability. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, page 1049–1054, 2014.

[34] Riccardo Cognini, Flavio Corradini, Stefania Gnesi, Andrea Polini, and Barbara Re. Business process flexibility-a systematic literature review with a software systems perspective. *Inf. Systems Frontiers*, 20(2):343–371, 2018.

[35] Ivan Compagnucci, Flavio Corradini, Fabrizio Fornari, and Barbara Re. Trends on the usage of bpmn 2.0 from publicly available repositories. In *Perspectives in Business Informatics Research*, pages 84–99. Springer International Publishing, 2021.

[36] Mario Cortes Cornax, Sophie Dupuy-Chessa, Dominique Rieu, and Marlon Dumas. Evaluating choreographies in BPMN 2.0 using an extended quality framework. In *Business Process Model and Notation - Third International Workshop, BPMN*, volume 95 of *Lecture Notes in Business Information Processing*, pages 103–117. Springer, 2011.

[37] Flavio Corradini, Fausto Marcantoni, Andrea Morichetta, Andrea Polini, Barbara Re, and Massimiliano Sampaolo. Enabling auditing of smart contracts through process mining. In *From Software Engineering to Formal Methods and Tools, and Back*, volume 11865 of *LNCS*, pages 467–480. Springer, 2019.

[38] Flavio Corradini, Andrea Morichetta, Andrea Polini, Barbara Re, and Francesco Tiezzi. Collaboration vs. choreography conformance in BPMN 2.0: From theory to practice. In *EDOC*, pages 95–104. IEEE Computer Society, 2018.

[39] Flavio Corradini, Chiara Muzi, Barbara Re, Lorenzo Rossi, and Francesco Tiezzi. Animating multiple instances in BPMN collaborations: From formal semantics to tool support. In *Business Process Management - 16th International Conference, BPM*, volume 11080 of *LNCS*, pages 83–101. Springer, 2018.

[40] Flavio Corradini, Chiara Muzi, Barbara Re, Lorenzo Rossi, and Francesco Tiezzi. MIDA: multiple instances and data animator. In *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018 co-located with 16th International Conference on Business Process Management*, volume 2196 of *CEUR Workshop Proceedings*, pages 86–90. CEUR-WS.org, 2018.

[41] Flavio Corradini, Chiara Muzi, Barbara Re, Lorenzo Rossi, and Francesco Tiezzi. Formalising and animating multiple instances in BPMN collaborations. *Inf. Syst.*, 103:101459, 2022.

[42] Jun Dai and Miklos A Vasarhelyi. Toward blockchain-based accounting and assurance. *Journal of Information Systems*, 31(3):5–21, 2017.

[43] Erik Daniel and Florian Tschorsch. IPFS and friends: A qualitative comparison of next generation peer-to-peer data networks. *IEEE Commun. Surv. Tutorials*, 24(1):31–52, 2022.

[44] José Eduardo de Azevedo Sousa, Vinicius C. Oliveira, Júlia Valadares, Glauber Dias Gonçalves, Saulo Moraes Villela, Heder Soares Bernardino, and Alex Borges Vieira. An analysis of the fees and pending time correlation in ethereum. *Int. J. Netw. Manag.*, 31(3), 2021.

[45] Claudio Di Ciccio, Alessio Cecconi, Jan Mendling, Dominik Felix, Dominik Haas, Daniel Lilek, Florian Riel, Andreas Rumpl, and Philipp Uhlig. Blockchain-based traceability of inter-organisational business processes. In *International Symposium on Business Modeling and Software Design*, volume 319 of *LNBIP*, pages 56–68. Springer, 2018.

[46] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.

[47] Urooj Fatima and Rolv Bræk. Modelling multiplicity in choreography models. In Ana Moreira, Gunter Mussbacher, João Araújo, Nelly Bencomo, and Pablo Sánchez, editors, *International Workshop on Model-Driven Requirements Engineering, MoDRE 2013, Rio de Janeiro, Brasil, July 15, 2013*, pages 74–78. IEEE Computer Society, 2013.

[48] Walid Fdhila, Stefanie Rinderle-Ma, David Knuplesch, and Manfred Reichert. Change and compliance in collaborative processes. In *2015 IEEE International Conference on Services Computing*, pages 162–169. IEEE, 2015.

[49] Hans-Georg Fill and Felix Härer. Knowledge blockchains: Applying blockchain technologies to enterprise modeling. In Tung Bui, editor, *51st Hawaii International Conference on System Sciences, HICSS*, pages 1–10. ScholarSpace / AIS Electronic Library (AISeL), 2018.

[50] Imen Ben Fraj, Yousra Bendaly Hlaoui, and Leila Ben Ayed. A control system for managing the flexibility in BPMN models of cloud service workflows. In *13th IEEE International Conference on Cloud Computing*, pages 537–543. IEEE, 2020.

[51] Gilbert Fridgen, Sven Radszuwill, Nils Urbach, and Lena Utz. Cross-organizational workflow management using blockchain technology - towards applicability, auditability, and automation. In *Hawaii International Conference on System Sciences*, pages 1–10. AIS Electronic Library, 2018.

[52] Luciano García-Bañuelos, Alexander Ponomarev, Marlon Dumas, and Ingo Weber. Optimized execution of business processes on blockchain. In *Business Process Management*, volume 10445 of *LNCS*, pages 130–146. Springer, 2017.

[53] Andrew Gemino and Yair Wand. Complexity and clarity in conceptual modeling: comparison of mandatory and optional properties. *Data & Knowledge Engineering*, 55(3):301–326, 2005.

[54] Yuichi Hanada, Luke Hsiao, and Philip Levis. Smart contracts for machine-to-machine communication: Possibilities and limitations. In *Internet of Things and Intelligence System*, pages 130–136. IEEE, 2018.

[55] Felix Harer and Hans-Georg Fill. A comparison of approaches for visualising blockchains and smart contract. In *IRIS*, pages 133–140, 2019.

[56] Felix Härer and Hans-Georg Fill. Decentralized attestation of conceptual models using the ethereum blockchain. In Jörg Becker and Dmitry A. Novikov, editors, *21st IEEE Conference on Business Informatics, CBI*, pages 104–113. IEEE, 2019.

[57] Archana Prashanth Joshi, Meng Han, and Yan Wang. A survey on security and privacy issues of blockchain technology. *Mathematical foundations of computing*, 1:121, 2018.

[58] Dodo Khan, Low Tang Jung, and Manzoor Ahmed Hashmani. Systematic literature review of challenges in blockchain scalability. *Applied Sciences*, 11(20), 2021.

[59] Shafaq Naheed Khan, Faiza Loukil, Chirine Ghedira Guegan, Elhadj Benkhelifa, and Anoud Bani-Hani. Blockchain smart contracts: Applications, challenges, and future trends. *Peer-to-Peer Netw. Appl.*, 14(5):2901–2925, 2021.

[60] Dongsoo Kim, Minsoo Kim, and Hoontae Kim. Dynamic business process management based on process change patterns. In *International Conference on Convergence Information Technology*, pages 1154–1161. IEEE, 2007.

[61] Philipp Klinger and Freimut Bodendorf. Blockchain-based cross-organizational execution framework for dynamic integration of process collaborations. In *Entwicklungen, Chancen und Herausforderungen der Digitalisierung: Proceedings der 15. Internationalen Tagung Wirtschaftsinformatik, WI*, pages 893–908. GITO Verlag, 2020.

[62] Philipp Klinger, Long Nguyen, and Freimut Bodendorf. Upgradeability concept for collaborative blockchain-based business process execution framework. In *Conference on Blockchain*, volume 12404 of *LNCS*, pages 127–141. Springer, 2020.

[63] Christopher Klinkmüller, Alexander Ponomarev, An Binh Tran, Ingo Weber, and Wil van der Aalst. Mining blockchain processes: Extracting process mining data from blockchain applications. In *International Conference on Business Process Management*, volume 361 of *LNBIP*, pages 71–86. Springer, 2019.

[64] Stephen Kozlowski. An audit ecosystem to support blockchain-based accounting and assurance. In *Continuous Auditing: Theory and Application*, pages 299–313. Emerald Publishing Limited UK, 2018.

[65] Jan Ladleif, Anton von Weltzien, and Mathias Weske. chor-js: A modeling framework for BPMN 2.0 choreography diagrams. In *Proceedings of the ER Forum and Poster & Demos Session*, volume 2469 of *CEUR Workshop Proceedings*, pages 113–117. CEUR-WS.org, 2019.

[66] Jan Ladleif, Mathias Weske, and Ingo Weber. Modeling and enforcing blockchain-based choreographies. In *International Conference on Business Process Management*, volume 11675 of *LNCS*, pages 69–85. Springer, 2019.

[67] Christine Legner and Kristin Wende. The challenges of inter-organizational business process design - A research agenda. In *Proceedings of the Fifteenth European Conference on Information Systems ECIS*, pages 106–118. University of St. Gallen, 2007.

[68] Dongcheng Li, W. Eric Wong, and Jincui Guo. A survey on blockchain for enterprise using hyperledger fabric and composer. In *6th International Conference on Dependable Systems and Their Applications, DSA 2019, Harbin, China, January 3-6, 2020*, pages 71–80. IEEE, 2019.

[69] Thomas Locher, Sebastian Obermeier, and Yvonne-Anne Pignolet. When can a distributed ledger replace a trusted third party? In *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)*, pages 1069–1077. IEEE, 2018.

[70] María Teresa Gómez López, José Miguel Pérez-Álvarez, Ángel Jesús Varela-Vaca, and Rafael M. Gasca. Guiding the creation of choreographed processes with multiple instances based on data models. In *Business Process Management Workshops - BPM 2016 International Workshops, Rio de Janeiro, Brazil, September 19, 2016, Revised Papers*, volume 281 of *LNBIP*, pages 239–251. Springer Verlag, 2016.

[71] Orlenys López-Pintado, Marlon Dumas, Luciano García-Bañuelos, and Ingo Weber. Dynamic role binding in blockchain-based collaborative business processes. In *Advanced Information Systems Engineering*, volume 11483 of *LNCS*, pages 399–414. Springer, 2019.

[72] Orlenys López-Pintado, Marlon Dumas, Luciano García-Bañuelos, and Ingo Weber. Interpreted execution of business process models on

blockchain. In *23rd IEEE International Enterprise Distributed Object Computing Conference, EDOC*, pages 206–215. IEEE, 2019.

[73] Orlenys López-Pintado, Marlon Dumas, Luciano García-Bañuelos, and Ingo Weber. Controlled flexibility in blockchain-based collaborative business processes. *Information Systems*, 104:101622, 2022.

[74] Orlenys López-Pintado, Marlon Dumas, and Ingo Weber. Caterpillar: A blockchain-based business process management system. In *BPM Demo Track and BPM Dissertation Award*, volume 1920. CEUR-WS.org, 2017.

[75] Orlenys López-Pintado, Luciano García-Bañuelos, Marlon Dumas, Ingo Weber, and Alexander Ponomarev. Caterpillar: A business process execution engine on the ethereum blockchain. *Softw. Pract. Exp.*, 49(7):1162–1193, 2019.

[76] Mads Frederik Madsen, Mikkel Gaub, Tróndur Høgnason, Malthe Ettrup Kirkbro, Tijs Slaats, and Søren Debois. Collaboration among adversaries: distributed workflow execution on a blockchain. In *Symposium on Foundations and Applications of Blockchain*, page 8, 2018.

[77] Luana Marrocco, Eduardo Castelló Ferrer, Antonio Bucchiarone, Arnaud Grignard, Luis Alonso, Kent Larson, et al. Basic: Towards a blockchained agent-based simulator for cities. In *Massively Multiagent Systems*, volume 11422 of *LNCS*, pages 144–162. Springer, 2018.

[78] Ricardo Martinho, Dulce Domingos, and João Varajão. Cf4bpmn: a bpmn extension for controlled flexibility in business processes. *Procedia Computer Science*, 64:1232–1239, 2015.

[79] Asma Mejri, Sonia Ayachi Ghanouchi, and Ricardo Martinho. Evaluation of process modeling paradigms enabling flexibility. *Procedia Computer Science*, 64:1043–1050, 2015.

[80] Jan Mendling, Hajo A Reijers, and Jan Recker. Activity labeling in process modeling: Empirical insights and recommendations. *Information Systems*, 35(4):467–482, 2010.

[81] Jan Mendling, Ingo Weber, and et al. Blockchains for business process management - challenges and opportunities. *ACM Transactions on Management Information Systems*, 9(1):1–16, 2018.

[82] Nazila Gol Mohammadi and Maritta Heisel. Enhancing business process models with trustworthiness requirements. In *Trust Management X - 10th IFIP WG 11.11 International Conference, IFIPTM 2016,*

Darmstadt, Germany, July 18-22, 2016, Proceedings, volume 473 of *IFIP Advances in Information and Communication Technology*, pages 33–51. Springer, 2016.

[83] Nazila Gol Mohammadi, Sachar Paulus, Mohamed Bishr, Andreas Metzger, Holger Könnecke, Sandro Hartenstein, Thorsten Weyer, and Klaus Pohl. Trustworthiness attributes and metrics for engineering trusted internet-based software systems. In *Cloud Computing and Services Science - Third International Conference, CLOSER 2013, Aachen, Germany, May 8-10, 2013, Revised Selected Papers*, volume 453 of *Communications in Computer and Information Science*, pages 19–35. Springer, 2013.

[84] Roman Mühlberger, Stefan Bachhofner, Claudio Di Ciccio, Luciano García-Bañuelos, and Orlenys López-Pintado. Extracting event logs for process mining from data stored on the blockchain. In *International Conference on Business Process Management*, volume 362 of *LNBIP*, pages 690–703. Springer, 2019.

[85] Roman Mühlberger, Stefan Bachhofner, Eduardo Castelló Ferrer, Claudio Di Ciccio, Ingo Weber, Maximilian Wöhrer, and Uwe Zdun. Foundational oracle patterns: Connecting blockchain to the off-chain world. In *Business Process Management: Blockchain and Robotic Process Automation Forum - BPM 2020 Blockchain and RPA Forum*, volume 393 of *Lecture Notes in Business Information Processing*, pages 35–51. Springer, 2020.

[86] Marcel Müller, Nadine Ostern, and Michael Rosemann. Silver bullet for all trust issues? blockchain-based trust patterns for collaborative business processes. In *Business Process Management: Blockchain and Robotic Process Automation Forum - BPM 2020 Blockchain and RPA Forum,*, volume 393 of *LNBIP*, pages 3–18. Springer, 2020.

[87] Adriatik Nikaj, Mathias Weske, and Jan Mendling. Semi-automatic derivation of restful choreographies from business process choreographies. *Softw. Syst. Model.*, 18(2):1195–1208, 2019.

[88] OMG. BPMN by Example, 2011.

[89] OMG. Business Process Model and Notation, 2011.

[90] Thomas Osterland, Thomas Rose, and Clemens Putschli. On the implementation of business process logic in dlt nodes. In *Proceedings of the 2020 Asia Service Sciences and Software Engineering Conference*, pages 91–99. ACM, 2020.

[91] Oscar Pastor. Model-driven development in practice: From requirements. In *Theory and Practice of Computer Science*, volume 10139 of *LNCS*, pages 405–410. Springer, 2017.

[92] Isabel Pedrosa and Carlos J Costa. New trends on caatts: what are the chartered accountants' new challenges? In *Proceedings of the International Conference on Information Systems and Design of Communication*, pages 138–142. ACM, 2014.

[93] Julien Polge, Jérémy Robert, and Yves Le Traon. Permissioned blockchain frameworks in the industry: A comparison. *ICT Express*, 2020.

[94] Simone Porru, Andrea Pinna, Michele Marchesi, and Roberto Tonelli. Blockchain-oriented software engineering: challenges and new directions. In *Software Engineering Companion*, pages 169–171. IEEE/ACM, 2017.

[95] Vahid Pourheidari, Sara Rouhani, and Ralph Deters. 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)*, pages 1588–1594. IEEE, 2018.

[96] Sandro Psaila. Blockchain: A game changer for audit processes. *Deloitte Malta Article*, pages 1–4, 2017.

[97] Manfred Reichert and Barbara Weber. *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer, 2012.

[98] Ronald J Reisman. Air traffic management blockchain infrastructure for security, authentication, and privacy. 2019.

[99] Zabihollah Rezaee, Ahmad Sharbatoghlie, Rick Elam, and Peter L McMickle. Continuous auditing: Building automated auditing capability. *Auditing: A Journal of Practice & Theory*, 21(1):147–163, 2002.

[100] Henrique Rocha and Stéphane Ducasse. Preliminary steps towards modeling blockchain oriented software. In *Emerging Trends in Software Engineering for Blockchain*, pages 52–57. ACM, 2018.

[101] Pingcheng Ruan, Tien Tuan Anh Dinh, Dumitrel Loghin, Meihui Zhang, Gang Chen, Qian Lin, and Beng Chin Ooi. Blockchains vs.

distributed databases: Dichotomy and fusion. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 1504–1517. ACM, 2021.

[102] P Sajana, M Sindhu, and M Sethumadhavan. On blockchain applications: Hyperledger fabric and ethereum. *International Journal of Pure and Applied Mathematics*, 118(18):2965–2970, 2018.

[103] Helen Schonenberg, Ronny Mans, Nick Russell, Nataliya Mulyar, and Wil M. P. van der Aalst. Process flexibility: A survey of contemporary approaches. In *Advances in Enterprise Engineering I*, volume 10 of *Lecture Notes in Business Information Processing*, pages 16–30. Springer, 2008.

[104] Paul H Schurr and Julie L Ozanne. Influences on exchange processes: Buyers' preconceptions of a seller's trustworthiness and bargaining toughness. *Journal of consumer research*, 11(4):939–953, 1985.

[105] Fabian Stiehle and Ingo Weber. Blockchain for business process enactment: A taxonomy and systematic literature review. *CoRR*, abs/2206.03237, 2022.

[106] Christian Sturm, Jonas Scalanczi, Stefan Schönig, and Stefan Jablonski. A blockchain-based and resource-aware process execution engine. *Future Generation Computer Systems*, 100:19–34, 2019.

[107] Christian Sturm, Jonas Szalanczi, Stefan Schönig, and Stefan Jablonski. A lean architecture for blockchain based decentralized process execution. In *Business Process Management Workshops*, volume 342 of *LNBIP*, pages 361–373. Springer, 2018.

[108] Andrew Sutton and Reza Samavi. Blockchain enabled privacy audit logs. In *International Semantic Web Conference*, volume 10587 of *LNCS*, pages 645–660. Springer, 2017.

[109] Shigeya Suzuki and Jun Murai. Blockchain as an audit-able communication channel. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 516–522. IEEE, 2017.

[110] Pinyaphat Tasatanattakool and Chian Techapanupreeda. Blockchain: Challenges and applications. In *2018 International Conference on Information Networking (ICOIN)*, pages 473–475, 2018.

[111] An Binh Tran, Qinghua Lu, and Ingo Weber. Lorikeet: A model-driven engineering tool for blockchain-based business process execution and

asset management. In *BPM Dissertation Award, Demonstration, and Industrial Track*, volume 2196, pages 56–60. CEUR-WS.org, 2018.

[112] Dennis Trautwein, Aravindh Raman, Gareth Tyson, Ignacio Castro, Will Scott, Moritz Schubotz, Bela Gipp, and Yiannis Psaras. Design and evaluation of IPFS: a storage layer for the decentralized web. In Fernando Kuipers and Ariel Orda, editors, *SIGCOMM '22: ACM SIGCOMM 2022 Conference, Amsterdam, The Netherlands, August 22 - 26, 2022*, pages 739–752. ACM, 2022.

[113] Wil M. P. van der Aalst, Niels Lohmann, Peter Massuthe, Christian Stahl, and Karsten Wolf. Multiparty contracts: Agreeing and implementing interorganizational processes. *Comput. J.*, 53(1):90–106, 2010.

[114] Wil MP Van der Aalst. Business process management: a comprehensive survey. *International Scholarly Research Notices*, pages 1–37, 2013.

[115] Miklos A Vasarhelyi and Fern B Halper. The continuous audit of online systems. In *Auditing: A Journal of Practice and Theory*, pages 110–125. Citeseer, 1991.

[116] Olegas Vasilecas, Diana Kalibatiene, and Dejan Lavbič. Rule-and context-based dynamic business process modelling and simulation. *Journal of Systems and Software*, 122:1–15, 2016.

[117] Viswanath Venkatesh and Hillol Bala. Adoption and impacts of interorganizational business process standards: Role of partnering synergy. *Inf. Syst. Res.*, 23(4):1131–1157, 2012.

[118] Wattana Viriyasitavat and Danupol Hoonsopon. Blockchain characteristics and consensus in modern business processes. *Journal of Industrial Information Integration*, 13:32–39, 2019.

[119] Marko Vukolić. Hyperledger fabric: towards scalable blockchain for business. *Trust in Digital Life*, 2016.

[120] Ingo Weber, Xiwei Xu, Régis Riveret, Guido Governatori, Alexander Ponomarev, and Jan Mendling. Untrusted business process monitoring and execution using blockchain. In *Business Process Management*, volume 9850 of *LNCS*, pages 329–347. Springer, 2016.

[121] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures.* Springer, 2007.

[122] Florian Wessling and Volker Gruhn. Engineering software architectures of blockchain-oriented applications. In *Software Architecture Companion*, pages 45–46. IEEE, 2018.

[123] Stephen A White. Introduction to bpmn. *Ibm Cooperation*, 2(0):0, 2004.

[124] Karl Wüst and Arthur Gervais. Do you need a blockchain? *IACR Cryptol. ePrint Arch.*, page 375, 2017.

[125] Xiwei Xu, Qinghua Lu, Yue Liu, Liming Zhu, Haonan Yao, and Athanasios V Vasilakos. Designing blockchain-based applications a case study for imported product traceability. *Future Generation Computer Systems*, 92:399–406, 2019.

[126] Xiwei Xu, Cesare Pautasso, Liming Zhu, Qinghua Lu, and Ingo Weber. A pattern collection for blockchain-based applications. In *Proceedings of the 23rd European Conference on Pattern Languages of Programs*, EuroPLoP '18, pages 3:1–3:20. ACM, 2018.

[127] Xiwei Xu, Ingo Weber, Mark Staples, Liming Zhu, Jan Bosch, Len Bass, Cesare Pautasso, and Paul Rimb. A taxonomy of blockchain-based systems for architecture design. In *International Conference on Software Architecture*, pages 243–252. IEEE, 2017.

[128] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. *CoRR*, abs/1906.11078, 2019.

[129] Rui Yuan, Yu-Bin Xia, Hai-Bo Chen, Bin-Yu Zang, and Jan Xie. Shadoweth: Private smart contract on public blockchain. *Journal of Computer Science and Technology*, 33(3):542–556, 2018.

[130] Rui Zhang, Rui Xue, and Ling Liu. Security and privacy on blockchain. *ACM Comput. Surv.*, 52(3):51:1–51:34, 2019.

[131] Qiuhong Zheng, Yi Li, Ping Chen, and Xinghua Dong. An innovative ipfs-based storage model for blockchain. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2018, Santiago, Chile, December 3-6, 2018*, pages 704–708. IEEE Computer Society, 2018.