

An Efficient Customized Blockchain System for Inter-Organizational Processes

Puwei Wang^{1,2}, Zhouxing Sun¹, Rui Li³, Jinchuan Chen^{1,2*}, Ping Gong⁴, Xiaoyong Du^{1,2}

¹School of Information, Renmin University of China

²Key Laboratory of Data Engineering and Knowledge Engineering (Ministry of Education), Renmin University of China

³School of Mathematics, Renmin University of China

⁴College of Computer and Cyber Security, Fujian Normal University

*Corresponding Author: jcchen@ruc.edu.cn

Abstract—Blockchain technologies pave a promising way for implementing the inter-organizational processes. Most of the current research works translate the execution logic in the process models into the smart contracts, which can run independently on the blockchain without the outside process engine. However, the works usually suffer from the execution and storage costs, since the translation needs to be done when the processes are deployed. In this paper, we customize a process engine for executing the inter-organizational business processes via a blockchain-style procedure, i.e., checking the validity of transactions, adding the valid transactions into the blockchain through the consensus mechanism, and then updating the process states according to the committed transactions. And then, we build a blockchain system by embedding the customized process engine into the blockchain nodes. Moreover, in order to realize the interactions between the inter-organizational processes running on blockchain and the services outside blockchain, we propose a blockchain-based approach for service registration, binding and invocation, and design a lease-based concurrency control protocol to logically isolate transactions from each other when invoking the services simultaneously. Finally, we implement a prototype system based on a permissioned blockchain platform *Hyperledger Fabric* and a process engine *Activiti*. The experimental results show the proposed blockchain system can execute the inter-organizational processes correctly and efficiently.

Index Terms—Inter-Organizational Processes, Process Engine, Inter-Organizational Services, Customized Blockchain System

I. INTRODUCTION

An inter-organizational process is a collection of activities involving two or more organizations to achieve a common objective. The emergence of blockchain technologies paves the promising way for implementing the business processes across organizational boundaries, especially involving mutually distrustful participants.

An important issue raised by leveraging the blockchain for executing inter-organizational processes is how the process engine runs on the blockchain. A process engine is a software framework that enables the execution and management of business processes. A straight idea is to encapsulate the process engine into an oracle [1], which is considered as a trusted service outside the blockchain. That is, the process engine runs outside the blockchain, and the blockchain only provides the shared ledgers for storing transactions and process states. However, it introduces a trusted third party, that is not favored by the blockchain-based applications. Having fully

blinded trusts on the oracles may greatly hinder the reliability. Most recent research works [2] [3] [4] [5] propose translating the execution logic in the process models into the smart contracts, which can run independently on the blockchain without the outside process engines, so that they can overcome security risks caused by the oracles. However, these works usually suffer from the execution and storage costs, since the translation needs to be done when the processes are deployed. Moreover, the Business Process Modeling and Notation (BPMN) diagrams [6] are widely used to model the inter-organizational processes. The existing process engines can often implement the BPMN 2.0 specification. But, recent research works do not make use of the existing full-featured process engines.

In this paper, we build a permissioned blockchain system for executing the inter-organizational processes by embedding a full-featured process engine into each blockchain node. For example, *Activiti*¹ is a lightweight and open-source BPMN 2.0 process-engine framework. *Hyperledger Fabric*² is a modular and high-performing permissioned blockchain platform. We can embed the Activiti process engine into the Hyperledger Fabric's nodes to build a blockchain system for executing the inter-organizational processes.

There are two challenges for building the blockchain system. One is how to ensure the process engines run correctly on the blockchain? The process engines embedded in the malicious nodes may falsify the execution results. In our approach, we customize a process engine to execute the inter-organizational processes via a blockchain-style procedure, i.e., the *endorsement-consensus-committing* procedure. Endorsement is a way for permissioned blockchains to verify whether a transaction is valid or not. In the endorsement phase, the process engines simulate executing processes and provide the simulation results as endorsements. Only transactions with enough endorsements can be submitted to the consensus mechanism. In the consensus phase, the blockchain nodes check the endorsements of transactions, and put valid transactions into the blocks. The consensus mechanism can enable the blockchain nodes to maintain a consistent copy of transactions.

¹Activiti, <https://www.activiti.org>

²Hyperledger Fabric, <https://hyperledger-fabric.readthedocs.io>

And then, in the committing phase, the process engines update the local copies of process states according to the committed transactions. In this way, we make use of the existing full-featured process engines, and ensure them run correctly.

The other challenge is how to realize the interactions between the inter-organizational processes running on the blockchain and the services outside the blockchain? Organizations usually have their own information systems and they can expose the external interface of their information systems in the form of services, which are platform-independent computational elements, and can be discovered and invoked using standard protocols. The services provided by the organizations sometimes need to fulfill the responsibilities proposed by the inter-organizational processes. In order to reliably interact with the services, we propose a blockchain-based approach for service registration, binding and invocation. Moreover, there may be conflicts between the transactions when invoking the services simultaneously. To address this issue, we design a lease-based concurrency control protocol. It will hold up the conflicted transactions in the endorsement phase so that the transactions put into the blockchain are logically isolated. Moreover, it permits transactions to be processed in parallel and then improves overall performance.

Specifically, we make the following contributions.

- We customize a process engine to execute the inter-organizational business processes via a blockchain-style procedure, i.e., checking the validity of transactions, then adding the transactions into the blockchain through the consensus mechanism, and then updating the process states according to the committed transactions;
- We realize the reliable interactions between the inter-organizational processes running on the blockchain and the services outside the blockchain, and design a lease-based concurrency control protocol to ensure the necessary isolation among transactions when invoking the services simultaneously.
- We implement a prototype system based on the permissioned blockchain platform *Hyperledger Fabric* and the process engine *Activiti*. Extensive experimental results show that the proposed blockchain system can run correctly and efficiently.

The paper is structured as follows. Section II reviews the related works. Section III introduces the motivating example used to guide our work. Section IV sketches the customized blockchain system for executing the inter-organizational processes. Section V introduces a prototype system and provides the experimental analysis. Finally, Section VI concludes this paper and discusses the future work.

II. RELATED WORK

A. Process Modeling

Blockchain technologies are considered as a promising technical solution for revitalizing research area [7]. Current research works on the inter-organizational business processes mainly start with modeling the processes by specification languages, and then focus on interactions and constraints across

organizational boundaries. Normally, current modeling works can be divided into process-centric modeling and declarative modeling.

The process-centric works mainly adopt BPMN diagrams, i.e., collaboration or choreography diagrams [3] [8] [9] [10] [11], since they are standardized with formal semantics. Weber et al. [3] adopts choreography diagram to model choreography tasks with message exchange notations in the inter-organizational environment. As an open source blockchain-enabled framework for business processes, Caterpillar [8] uses the BPMN collaboration diagrams to model the processes. For the declarative modeling, the domain-specific languages like DCR (Dynamic Condition Response)-graphs [12] or Business Artifacts [13], [14] are applied to model flexible interactions. Besides considering the interactions from control-flow perspectives, recent research works make various extensions by introducing other perspectives into the process modeling, e.g., time [15], resource [16] [17] and complex decision [18], etc.

In process models, there is usually a kind of service task that can be automatically completed by using external services. The service tasks are necessary and feasible for the blockchain-enabled implementation of inter-organizational processes [19]. However, current research works do not have enough discussions on how to implement the interactions between inter-organizational processes running on the blockchain and the services outside blockchain.

B. Blockchain-based Implementation

For the implementation of blockchain-based process models, it is common to utilize the smart contracts, since they are programmable and accessible to on-chain transactional data. Current research works translate the process models (including process-centric and declarative ones) into the execution-oriented smart contracts (e.g., Solidity language of Ethereum) according to the process execution semantics. Basically, the execution-oriented smart contracts include the functional modules of execution, mediator (between on/off chain), storage, and logging, etc.

Weber et al [3] directly translate BPMN models into the smart contracts, which act as coordinator for the interactions between on-chain and off-chain. But, the work suffers from expensive cost of execution and storage, since the translation could not reuse the process engine modules for each time of process deployment. Such works are termed as compilation-based approach [20]. Similarly, the works [8] [9] [21] [22] [23] adopt the compilation-based translation. To ensure the correctness of translations, the works [9] [22] use the state machine and CSP (Communicating Sequential Programs) to capture the process semantics. Caterpillar [8] uses the Petri-net execution semantics, and then makes further optimizations to the translation by reducing Petri-net spaces [24]. To further reduce the cost, the work [20] recently proposes an interpretation-based method for executing the inter-organizational processes. The work presents an interpreter that encodes the semantics of the BPMN language in a single smart contract. That design reduces the costs of deployment since the smart contract

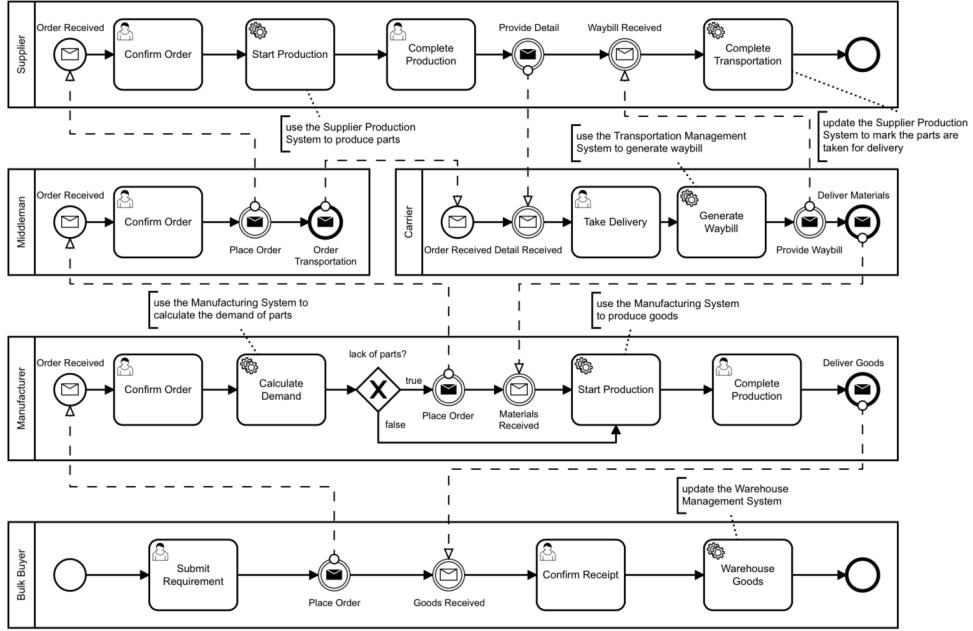


Fig. 1. BPMN collaboration diagram of supply chain scenario

encoding the interpreter only needs to be deployed once, but can support the instantiation of multiple process models.

Although the above works have also more or less discussed the performance issues, there is still a gap for the practical use of the blockchain-based process execution [19]. It is desirable to build a high-performing customized blockchain system for enacting the inter-organizational processes.

III. MOTIVATION EXAMPLE

Fig.1 depicts a BPMN collaboration diagram for representing a supply chain scenario [3]. We will use this scenario as a running example throughout the paper. This scenario involves five organizations: *Bulk Buyer*, *Manufacturer*, *Middleman*, *Carrier* and *Supplier*. These organizations have their own information systems, which have different data schemas and internal logic. For example, the Manufacturer runs a manufacturing system for producing goods. The information systems of organizations sometimes need to fulfill the responsibilities proposed by the inter-organizational processes. An inter-organizational process instance begins with the Bulk Buyer placing an order with the Manufacturer. The Manufacturer then uses its manufacturing system to calculate the demand of parts. If there are missing parts, it orders the parts via the Middleman. The Middleman forwards the order to the Supplier and arranges transportation by the Carrier. The Supplier uses its production system to produce parts. Once the parts are produced, the Carrier uses its transportation management system to generate waybills and deliver the parts to the Manufacturer. The Manufacturer produces the goods, and then delivers them to the Bulk Buyer.

We observe that the introduction of blockchain technologies brings performance issues, such as throughput and latency,

etc. In real world scenarios, a considerable number of process instances may run simultaneously on blockchain. In order to obtain good user experiences, there are performance requirements for the process engine. For example, in the supply chain scenario, the Bulk Buyers may demand that the tasks can be completed within 1 second on average. But, for the public blockchain based approaches, such as Caterpillar [8], the throughput is usually less than 100 and the latency may be more than ten seconds. The process engine built on the top of blockchain still needs to be optimized [25]. The permissioned blockchains are usually high-performing [26]. We aim to build an efficient permissioned blockchain system by embedding a full-featured process engine into the blockchain nodes.

We also observe that the information systems of organizations may be complementary to the inter-organizational processes. For example, the Supplier runs its information system to produce parts, and then the parts are delivered to the Manufacturer by the Carrier through the inter-organizational process. After the parts are taken by the Carrier, the Supplier updates its information system to mark the parts taken for delivery. The inter-organizational processes run on the blockchain, but the information systems of organizations run outside the blockchain. Recent works [27] introduce the oracles for implementing the on-chain and off-chain collaboration. But, blindly trusting the oracles may greatly hinder reliability. We need to realize the interaction between inter-organizational processes running on the blockchain and the information systems of organizations outside the blockchain.

IV. CUSTOMIZED BLOCKCHAIN SYSTEM

An inter-organizational process is composed of a collection of activities involving more than one organization, which collectively fulfill a common business objective. In the traditional

process engines, the trusted third-party nodes take responsibility to manage and execute the business processes. But, it is probably difficult to find the trusted third-party nodes for the inter-organizational cooperation, since the organizations collaborate in the peer-to-peer mode and there are no master-slave relationships between them. To this end, we build a blockchain system for the inter-organizational cooperation.

A. Architecture

Fig.2 depicts the system architecture that consists of a customized permissioned blockchain and a set of mutually untrusted organizations.

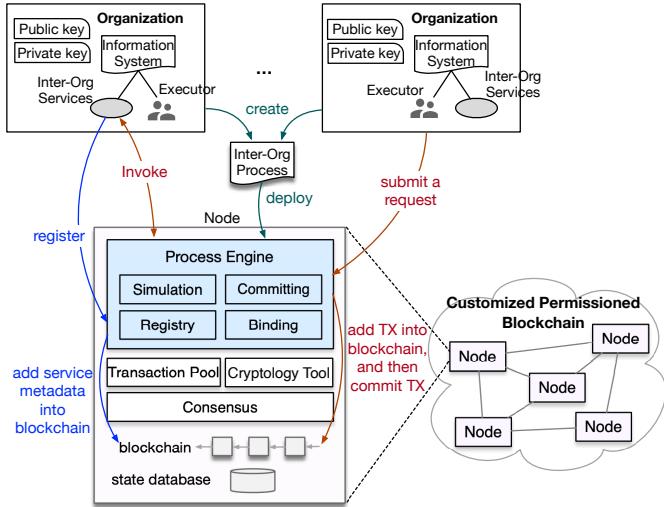


Fig. 2. The architecture of blockchain system

In permissioned blockchain network, a node usually has the modules: 1) Transaction pool. It is the place that contains the unconfirmed transactions; 2) Cryptology tool. It uses digital signatures for verification purposes; 3) Consensus module. It collaborates with the consensus modules of other nodes in the peer-to-peer mode to reach agreement that the transactions in a block are legitimate before the block is added to blockchain; 4) Blockchain. It is a growing list of blocks that are securely linked together via cryptographic hashes. Each block contains a set of transactions; 5) State database. State refers to the execution result of all committed transactions on blockchain. The state database holds the latest states and it is updated once new transactions are added into blockchain. In our system, the process instances and the transactions are stored in the blockchain, and the states of the process instances are stored in the state database.

We embed a full-featured process engine into the blockchain nodes. The process engine collaborates with the consensus module to check the validity of the transactions, and updates the state database by executing the committed transactions. For the process engine, we divide the execution of process instance into simulation and committing. Simulation execution will not change the states of process instances, but the simulation results will be submitted as evidence to the consensus

mechanism. If a transaction provides enough evidence, the transaction can be added into the blockchain and can be used to update the state of process instance. Moreover, the organizations expose the APIs of their information systems in the form of services. We call the services involved in the organizational cooperation the *inter-organizational services*. In order to ensure reliable interactions between the process engine and the services, the organizations should register the services for validation before invoking the services. The process engine consists of the following modules.

- *Simulation*. The module simulates executing the process instances and provides the simulation results as evidence. None of the effects of the simulation persist in the process instances;
- *Committing*. If the consensus mechanism confirms the validity of a transaction and adds it into blockchain, the module applies the changes made by the transaction to the local copies of the process states.
- *Registry*. The organizations submit their service metadata (such as name and APIs, etc) with digital signatures to the module, and then the module interacts with the consensus mechanism for adding them into blockchain;
- *Binding*. A process instance has a kind of service task that runs service to automatically complete the task. The module binds the inter-organizational services to the service tasks.

The overall procedure of inter-organizational cooperation is described as following: *First*, the organizations register their inter-organizational services into the blockchain. The blockchain network enables the service metadata to spread across organizations while ensuring they are not tampered with; *Second*, the organizations collectively create an inter-organizational process along with the signature that is generated collectively by the involved organizations, and submit the processes with the signature to the blockchain; *Third*, the organizations bind their services, which have been registered in the blockchain, to the inter-organizational process; *Fourth*, after receiving a request, the process engines simulate executing the request and provide the simulation results as the evidence. And then, the transaction is added into the blockchain, and the process engine embedded in each blockchain node applies the changes made by the transaction to the local copies of the process state.

B. Interaction with Inter-Organizational Services

In order to achieve the interactions between the inter-organizational processes running on the blockchain and the interactions between the services running outside the blockchain, we propose a blockchain-based approach for service registration, binding and invocation.

1) *Service Registration*: Service metadata is data describing services such as name, organization, address, port and APIs, etc. For example, in the supply chain scenario, the Manufacturer packages the external interfaces of manufacturing system as a service. The service metadata is shown below.

```
{
  service: "manufacturing",
  organization: "Manufacturer",
  address: "10.77.110.222", port: "8088",
  api: [{URI: "/calcuuate", method: "POST",
    description: "calculate the demand of parts",
    input: [{variable: "product", type: "string"}, {variable: "quantity", type: "integer"}],
    output:[{variable: "parts", type: "Array"}}],
  {URI: "/start", method: "GET",
    description: "start to produce goods",
    input:[{variable: "product", type: "string"}]
    output:[{variable: "started", type: "boolean"}]}]
}
```

That is, the service provides two APIs. One is used to calculate the demand of parts and the other is used to start to produce goods. Their URIs are `/calculate` and `/start`, respectively.

Algorithm 1: Blockchain-enabled Service Registration

Receive: a blockchain node n_0 receives a request $\text{REQUEST}(\text{metadata}, \text{sig})$, where metadata is the service metadata and sig is the signature of the service provider

- 1 node n_0 : the registry module broadcasts the request REQUEST to other blockchain nodes N ;
- 2 **for** $\forall n_i \in N$ **do**
- 3 | checkSig(sig); //the signature is valid
- 4 | healthy $_i \leftarrow \text{checkHealth}(\text{REQUEST})$; //checks whether the service works properly or not
- 5 | forward healthy $_i$ back to the node n_0 ;
- 6 **end**
- 7 node n_0 : healthy $\leftarrow \text{aggregate}(\{\text{healthy}_i | n_i \in N\})$;
- 8 node n_0 : the registry module puts the transaction $\text{REGISTER}(\text{REQUEST}, \text{healthy})$ into the transaction pool.

Result: $\text{REGISTER}(\text{REQUEST}, \text{healthy})$ is added into the blockchain through the consensus mechanism

The process of service registration is shown in Algorithm 1. An organization submits its registration request to a blockchain node. The registry module in the node broadcasts this request to other blockchain nodes (Line 1). The registry module in each node independently checks the service health and returns the inspection result to the node that initially receives the registration request (Lines 2-6). The registry module in the node integrates the inspection results to obtain the health status (Line 7), and puts the registration request along with the health status into the transaction pool (Line 8). Finally, the consensus mechanism adds the registration request along with the health status into the blockchain. The blockchain enables the service metadata to spread across organizations while ensuring they are not tampered with.

2) *Service Binding:* In the supply chain scenario, after the Manufacturer receives an order from the Bulk Buyer, it starts its manufacturing system to produce goods. The Manufacturer packages the external interfaces of its manufacturing system as a service and registers it into the blockchain. How to bind the service to the supply chain process? BPMN provides a kind of service task that uses a service to automatically complete the task. As shown in Fig.1, the service task *Calculate Demand*

is used by the Manufacturer to calculate the demand of parts. We bind the service *manufacturing* to the service task by using the binding information below.

```
{
  process: "Supply Chain",
  organization: "Manufacturer",
  servicetask: "Calculate Demand",
  service: "manufacturing",
  URI: "/calculate",
  input:[{"product": "Bulk Buyer"."product", "quantity": "Bulk Buyer"."quantity"}]
}
```

That is, the service *manufacturing*'s URI `/calculate` is bound to the service task *Calculate Demand*. The service has two input parameters: *product* and *quantity*. As shown in Fig.1, the Bulk Buyer has placed an order (including product and quantity) with the Manufacturer and the order has been added into the blockchain. When running the service task, it will read the data of product and quantity of the Bulk Buyer from the blockchain and then will take them as the service input.

Algorithm 2: Blockchain-enabled Service Binding

Receive: a blockchain node n_0 receives a request $\text{REQUEST}(\text{binfo}, \text{sig})$, where binfo is the binding information and sig is the threshold signature

- 1 node n_0 : the binding module broadcasts the request REQUEST to other blockchain nodes N ;
- 2 **for** $\forall n_i \in N$ **do**
- 3 | checkSig(sig); //the threshold signature is valid
- 4 | checkTask(binfo); //the service task is valid
- 5 | checkService(binfo); //the service is registered
- 6 | forward the inspection result result $_i$ back to the node n_0 ;
- 7 **end**
- 8 node n_0 : result $\leftarrow \text{aggregate}(\{\text{result}_i | n_i \in N\})$;
- 9 **if** result is valid **then**
- 10 | node n_0 : the binding module puts the transaction $\text{BIND}(\text{REQUEST})$ into the transaction pool.

Result: $\text{BIND}(\text{REQUEST})$ is added into the blockchain through the consensus mechanism and the process state is accordingly updated

In order to ensure authenticity, only the services which have been registered into the blockchain can be bound to the service tasks and the service binding request should be attached with a threshold signature, which is generated collectively by the involved organizations of the process instance. That is, the binding policy is that the involved organizations need to agree with the binding request. For example, the process *supply chain* involves five organizations. The binding request should be attached with a threshold signature of these organizations. Our approach can also apply to other binding policies, e.g., the dynamic binding policy discussed in [28]. The process of service binding is shown in Algorithm 2. The blockchain nodes independently check the validity of the binding request (Lines 1-7). The node that initially receives the request inte-

grates the inspection results. If the request is valid, it is put into the transaction pool (Lines 8-11). Finally, the consensus mechanism adds the binding request into the blockchain.

3) *Service Invocation*: For a service task, we can obtain the metadata, such as IP address and port, of the service bound to the service task from the blockchain. For example, the service *manufacturing* has been registered in the blockchain. By querying the blockchain, its IP address and port are 10.77.110.222 and 8088, respectively. That is, the service can be accessed from 10.77.110.222:8088. Moreover, the order data, which the organizations have submitted to the blockchain, can be used as the service input. Suppose the Bulk Buyer have placed an order `{product:"drone", quantity:10, address:"RUC"}`. According to the binding information, `{product:"drone", quantity:10}` will be sent to the service at the endpoint `http://10.77.110.222:8088/calculate`, and the service will return the demand of parts with its digital signature.

C. Execution of Inter-Organizational Process

BPMN collaboration diagram is widely used to model the inter-organizational processes. It consists of tasks, events and gateways. The tasks represent atomic activities. There are mainly two kinds of tasks: user task and service task. A user task is performed by a user. A service task does not require any interaction with the users. It is completed automatically, by using some sort of external service. The events represent something that happens during the course of process. They affect the flow of the process. The gateways are decision points that adjust the paths of process flows.

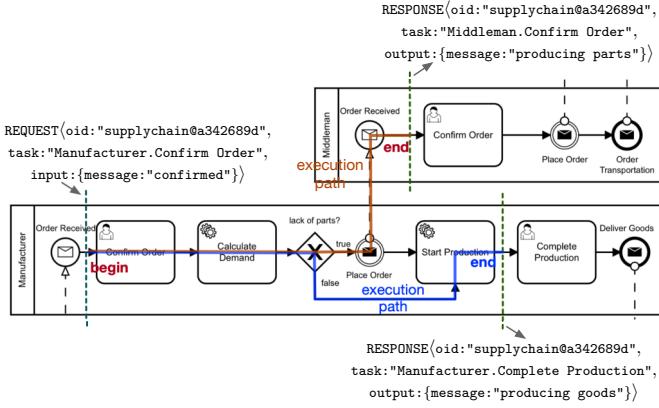


Fig. 3. The consecutive path of an execution

In the blockchain system, each blockchain node maintains a copy of instances of the inter-organizational processes. For example, as shown in Fig.3, suppose there is a process instance *supplychain@a342689d*. The manufacturing service has been bound to the service tasks *Manufacturer.Calculate Demand* and *Manufacturer.Start Production*, as discussed in Section IV-B. That is, these service tasks will be automatically executed by the manufacturing service. The current task is *Manufacturer.Confirm Order*. During executing the process instance, the Manufacturer completes the task, and then the service task *Manufacturer.Calculate Demand* is automatically

executed to calculate the demand of parts. A gateway decides the execution path according to the result of the service task. If there is a lack of parts, a message event occurs to send the order of missing parts to the Middleman, and then the current task changes to *Middleman.Confirm Order*; Otherwise, the service task *Manufacturer.Start Production* is automatically executed to start producing goods, and then the current task changes to *Manufacturer.Complete Production*. We find an execution of process instance has a consecutive execution path, starting from a user task and ending at the next user task. The execution path is determined by the process flow, the gateways and the execution results of the tasks within the path.

A request to execute process instance is given below.

`REQUEST<oid, t, ν>`

where `oid` is the ID of process instance, `t` is the current task of process instance and `ν` is the input related to the task. By executing the request, the process engine generates a transaction below.

`EXECUTE<REQUEST, nt, o, {(s, γ, ω)i|i ∈ [1, K]}>`

where `REQUEST` is the request, `nt` is the user task that the execution path ends at, `o` is the output, $\{(s, \gamma, \omega)_i | i \in [1, K]\}$ is a set of K service tasks within the execution path and the data they will read from (write into) the blockchain (s is a service task, γ is the data the service task reads from the blockchain and ω is the data the service task will write into the blockchain).

For example, as shown in Fig.3, the Manufacturer submits a request below.

`REQUEST<oid:"supplychain@a342689d", task:"Manufacturer.Confirm Order", input:{status:"confirmed"}>`

According to the request, the process engine generates a transaction (the red execution path in Fig.3) below.

`EXECUTE<REQUEST, nextTask:"Middleman.Confirm Order", output:{message:"ordering parts"}, {(serviceTask:"Manufacturer.Calculate Demand", readSet:{product:"drone", quantity:10}, writeSet:{parts:"propeller", quantity:10})}>`

That is, if executing the request, the next user task is *Middleman.Confirm Order* and the output is `{message:"ordering parts"}`. In the execution path, there is a service task *Manufacturer.Calculate Demand*. It reads the order of the Bulk Buyer and will write the demand of parts into the blockchain. Once the transaction is added into the blockchain, the process engine embedded in each blockchain node accordingly updates the local copy of the process state.

However, the process engines embedded in the malicious blockchain nodes may falsify the transactions. For example, they may falsify the next task to *Manufacturer.Complete Production*, instead of *Middleman.Confirm Order*. How to make the process engines generate valid transactions? We customize a process engine to execute the inter-organizational processes via a blockchain-style procedure, as shown in Fig.4.

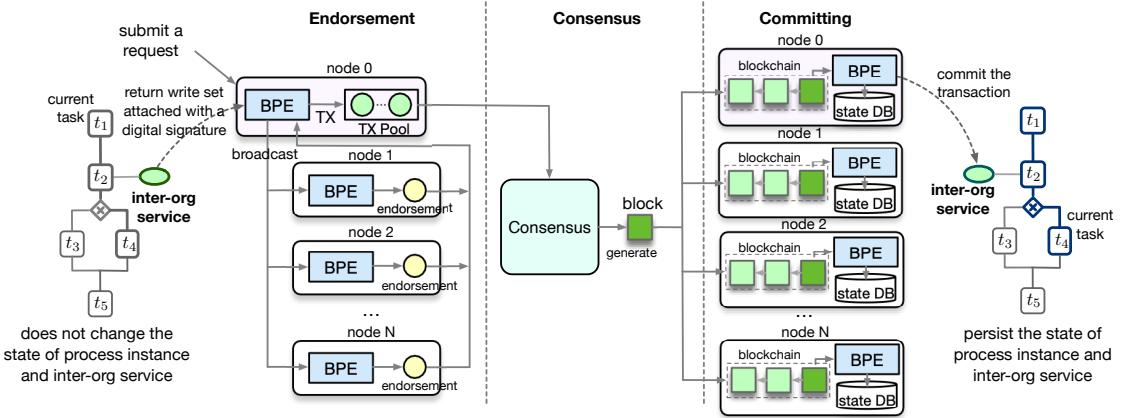


Fig. 4. The business process engines (BPEs) execute the inter-organizational processes via an *endorsement-consensus-committing* procedure

1) *Endorsement*: Endorsement means declaring support for something with its own credibility. It is a common way for permissioned blockchain to verify a transaction and declare the transaction is valid or not. The endorsement policy defines which nodes need to agree on the result of a transaction before it can be added into the blockchain [29]. Suppose the number of malicious nodes is f . We verify a transaction is valid if at least $f + 1$ blockchain nodes provide the endorsements for the transaction. Our approach can also apply for other kinds of endorsement policy.

As shown in Fig.4, the current task of process instance is t_1 . Suppose a request $\text{REQUEST}\langle\text{oid}, t_1, \nu\rangle$ is submitted to a blockchain node n_0 , where ν is the input related to the task t_1 . The process engine embedded in the node simulates executing the request. In the execution path, there is a service task t_2 that is bound to an inter-organizational service. The process engine obtains the write set attached with the service signature by invoking the service, as discussed in Section IV-B. According to the service result, a gateway makes the execution path end at the user task t_4 . In this way, the node generates a transaction as $\text{EXECUTE}\langle\text{REQUEST}, t_4, o, \{(t_2, \gamma, \omega)\}\rangle$, where o is the output, γ is the data that the service task t_2 will read from the blockchain and ω is the data that the service task t_2 will write into the blockchain. The node broadcasts the transaction to the endorsement nodes. For each endorsement node i , if there is no error generating the transaction according to the request REQUEST , it provides the hash of the transaction as the endorsement and returns it to the node n_0 .

$$\text{ENDORSE}\langle\text{hash}(\text{EXECUTE})\rangle\delta_i$$

where $\text{hash}(\text{EXECUTE})$ is the hash of the transaction EXECUTE and δ_i is the digital signature of the blockchain node i .

For a transaction, if having received at least $f + 1$ endorsements, it is proved to be valid according to the endorsement policy. The blockchain node n_0 then puts the transaction with the endorsements ($\text{EXECUTE}, \{\text{Endorse}_i\}$) into the transaction pool. In this phase, the blockchain nodes will not update the local copies of the process instance. The services within the execution path will also not change their states.

2) *Consensus*: The consensus mechanism takes the transactions from the transaction pool, encapsulating them into a block, checking whether they are endorsed correctly and adding the block in the blockchain. For example, Hyperledger Fabric, a classical permissioned blockchain platform, provides a pluggable consensus algorithm, such as PBFT [30], etc. It enables all nodes to spontaneously maintain a consistent copy of blockchain. Before a new block can be added into the blockchain, the blockchain nodes validate the endorsements in each transaction in the block against the endorsement policy, e.g., at least $f + 1$ blockchain nodes provide endorsements for the transaction. If the endorsement policy fails to satisfy, the transaction is marked as invalid. In this phase, if the transaction EXECUTE satisfies the endorsement policy, it is added into the blockchain through the consensus mechanism.

3) *Committing*: After the transaction EXECUTE is added into the blockchain, in each blockchain node, the committing module of the process engine constructs three objects: `execution`, `task` and `event`. The `execution` object depicts the ID of process instance and the execution time, etc. The `task` object depicts the next user task of process instance after completing the transaction. The `event` object depicts the events related to the task. The process engine in each blockchain node serializes the objects of `execution`, `task` and `event` in the state database.

Finally, the process engine in the node n_0 notifies the services within the execution path that the transaction has been committed. The services check whether the transaction is committed or not and then update their states according to the committed transaction. In this way, the request is completed.

D. Lease-based Concurrency Control Protocol

When invoking the inter-organizational services simultaneously, there may be conflicts among transactions. For example, in the supply chain scenario (Fig.3), there is a service task *Manufacturer.Calculate Demand* that uses an external service to calculate the demand of parts according to the inventory records. If two transactions have the same demand of parts, but the inventory can only meet one of them, there is a conflict between them.

We can usually adopt *lock* to handle the conflicts. When a transaction tries to access a data item x , it must obtain a lock on x first. All other transactions cannot access x until the transaction releases the lock. However, lock-based protocol can not work in our blockchain system, where the lifecycle of a transaction is divided into three separate phases (i.e. endorsement, consensus and committing). In the endorsement phase, we ask a service whether a transaction can be executed and obtain the write set. Later on in the committing phase, we invoke the service again to commit the write set. Note that a transaction that has successfully obtained an endorsement may still be aborted. It may fail in obtaining enough endorsements, or not be packaged into the block. If this transaction has been granted a lock from service, this lock will not be released and the corresponding data items will be logically inaccessible.

To handle this issue, we design a *lease*-based protocol. A lease is a lock attached with a time limit [31]. A transaction should complete its task and release the lock before the time expires, otherwise the lease manager (i.e. a service) will release the lock. Blockchain is a decentralized system and it is difficult to maintain a unified time clock. We use the *block height* to control the lease period. A lease request is attached with a block height h and the granted lock will expire when the service sees a block at the height of h .

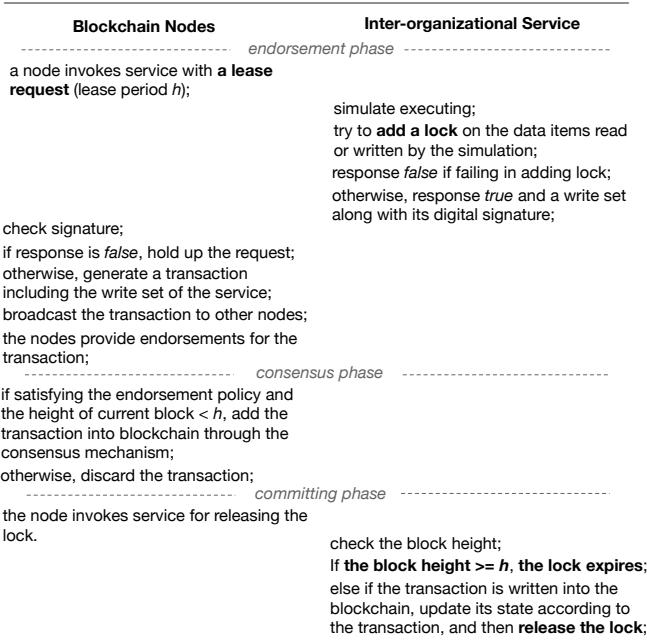


Fig. 5. Lease-based concurrency control protocol

The lease-based protocol is shown in Fig.5. In the endorsement phase, a lease request with a block height h is submitted to the service, and the service simulates executing the request to generate a transaction, and adds a lock on the data items. When other transactions try to access the data items, they will be held up. If the current block height is more than and equal to h and the transaction is still not written into the blockchain, the transaction is discarded. In the committing phase, if the

transaction has been added into the blockchain and the block height is less than h , the service updates its state according to the committed transactions. Otherwise, the lock expires.

With the concurrency control protocol, the conflicted transactions will be held up in the endorsement phase so that the transactions put into the blockchain are logically isolated. This can reduce the abort rate of transactions.

V. EVALUATION

A. Experiment Setup

Activiti is an open-source, lightweight process-engine framework. It is able to perform BPMN 2.0 functions, e.g., deploying process definitions, starting new process instances and executing tasks.

- We rewrite part of the Activiti's codes for constructing the simulation module, the committing module and the binding module;
- *Nacos*³ is an open source platform designed for dynamic service registration and discovery. We integrate Nacos with the blockchain and embed it into the Activiti process engine for constructing the registry module;
- The Activiti framework relies on the MySQL database for persisting the process state, the deployment artifacts and the process history. *Redis*⁴ is an open-source in-memory data structure store, used as an in-memory database. In order to achieve high performance, we rewrite part of the Activiti's codes to make it run on the Redis database.

Hyperledger Fabric is a permissioned blockchain platform. It mainly has two kinds of nodes: the orderer nodes (providing consensus service) and the peer nodes (maintaining a copy of blockchain data). A channel composed of a set of peer nodes actually is a permissioned blockchain. To join a channel, the peer nodes must be authenticated and authorized, and they can only propose or receive transactions on that channel.

- *IPFS* (InterPlanetary File System)⁵ is a peer-to-peer network for storing and sharing the large amounts of data. IPFS content addressing enables to store large files off-chain and put the immutable links in transactions on-chain, without having to put the data itself on-chain. We integrate Hyperledger Fabric and IPFS to manage the large amount of data in the inter-organizational processes.

We develop a customized blockchain system by embedding the revised Activiti process engine into the Hyperledger Fabric's peer nodes and managing the data of inter-organizational processes by the IPFS network. We deploy the blockchain system on the Renda Xing Cloud, which is a cloud computing platform managed by Renmin University of China. We have 17 nodes, each of which has 8 vCPU cores, 8G memory and 500G hard disk, running on CentOS 7.9. Among them, 4 nodes are the orderer nodes, 10 nodes are the peer nodes and 3 nodes run the inter-organizational services.

³Nacos, <https://github.com/alibaba/nacos>

⁴Redis, <https://redis.io>

⁵IPFS, <https://ipfs.tech>

TABLE I
REST API

Method	URI	Description
POST	/nacos/v1/ns/instance	Registers an inter-organizational service
GET	/nacos/v1/ns/instance/list	Queries the registered services
POST	/bind	Binds an inter-organizational service to a service task
POST	/wfRequest	Deploys a process; Instantiates a process instance; Executes a process instance
GET	/query	Queries the current state of a process instance

The functionality of the blockchain system is exposed via the REST API, summarized in Table I. The organizations register their services on the URI `/nacos/v1/ns/instance`, and bind the services to the service tasks on the URI `/bind`. The organizations deploy the processes, instantiate the process instances and execute the process instances on the URI `/wfRequest`, and query the process states on the URI `/query`.

B. Correctness Evaluation

We consider four kinds of inter-organizational processes, i.e., *supply chain*, *VIP channel*, *travel* and *order to cash*, derived from the literature [3] [8]. We derive a set of permissible execution traces for each process, called the *conforming traces*. Moreover, we obtain a set of *not conforming traces*, generated through random manipulation from the conforming ones. The manipulation operators include: adding an activity, removing an activity and switching the order of two activities. These make the modified traces different from the correct traces. For example, the supply chain scenario has 14 tasks and 1 gateway. We randomly generate 9,699 conforming traces and 2,416 not conforming traces for the supply chain process.

In the first set of experiments, we assume that the process engines embedded in malicious blockchain nodes may deliberately falsify transactions or simply not work. Table II shows the experiment results. The blockchain system identifies all the not-conforming traces in 100% of the cases tested. No task is incorrectly started within the not conforming traces. The system also successfully finishes all the conforming traces in 100% of the cases tested.

TABLE II
CONFORMANCE CHECKING RESULTS

Process	Tasks	Gates-ways	Trace Type	Traces	Correct
Supply Chain	14	1	Conforming	9,699	100%
			Not conforming	2,416	100%
VIP Channel	19	3	Conforming	2428	100%
			Not conforming	871	100%
Travel	18	4	Conforming	2,790	100%
			Not conforming	918	100%
Order to Cash	13	4	Conforming	2,499	100%
			Not conforming	843	100%

C. Performance Evaluation

Apache JMeter⁶ is a performance testing framework and has been widely used as a testing tool for Web applications. We use JMeter to simulate heavy loads on the blockchain system and use the following metrics to analyze the performance.

- *Throughput*: the amount of transactions the blockchain system can complete per second. A transaction represents a consecutive execution path consisting of a collection of tasks, events and gateways;
- *Latency*: the amount of time taken to complete a transaction, consisting of the endorsement time, the consensus time and the committing time.

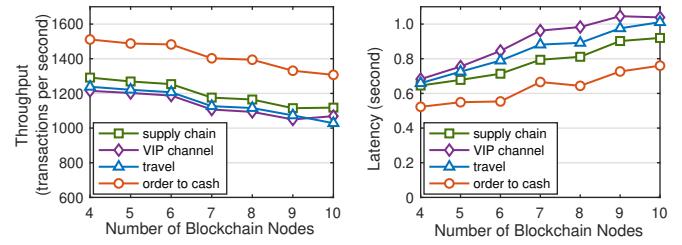


Fig. 6. Performance of different processes

In the second set of experiments, we evaluate the performance of four processes, i.e., *supply chain*, *VIP channel*, *travel* and *order to cash*. Among these processes, the average transaction execution path of the process *order to cash* is the shortest, and that of the process *VIP channel* is the longest. The experiment results can be seen in Fig.6. The throughput of *order to cash* is 1,416tps on average, and that of *VIP channel* is 1,132tps on average. The latency of *order to cash* is 0.632s and that of *VIP channel* is 0.9s on average. Different processes have different performance. The reason is that the performance is also influenced by the transaction execution paths. For these processes, with increasing the number of blockchain nodes, the throughput decreases from 1,314tps to 1,141tps on average, and the latency increases from 0.627s to 0.932s on average.

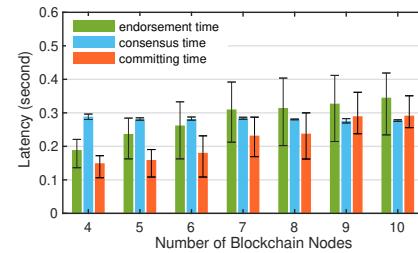


Fig. 7. Breakdown analysis of latency (when the throughput is over 1,200tps)

We also take a breakdown analysis of the latency. The latency consists of the endorsement time, the consensus time and the committing time. The experiment results can be seen in Fig.7. For these processes, there is little difference in the consensus time, but there is a relatively large difference in

⁶Apache JMeter, <https://jmeter.apache.org>

TABLE III
LATENCY ANALYSIS FOR THE SUPPLY CHAIN SCENARIO (SECOND)

Transaction	Average	Min	10th pct	50th pct	90th pct	95th pct	Max
start→Bulk Buyer.Submit Requirement	0.604	0.08	0.376	0.579	0.8	0.938	3.062
Bulk Buyer.Submit Requirement→Manufacturer.Confirm Order	0.653	0.112	0.392	0.639	0.865	1.03	2.603
Manufacturer.Confirm Order→Middleman.Confirm Order	0.674	0.113	0.401	0.675	0.88	1.356	2.614
Middleman.Confirm Order →Supplier.Confirm Order	0.737	0.119	0.435	0.749	0.923	1.138	3.292
Supplier.Confirm Order→Supplier.Complete Production	0.633	0.133	0.384	0.596	0.85	0.999	3.432
Supplier.Complete Production →Carrier.Take Delivery	0.697	0.113	0.411	0.706	0.89	1.075	3.575
Carrier.Take Delivery→Manufacturer.Complete Production	0.861	0.118	0.63	0.812	1.127	1.271	3.755
Manufacturer.Complete Production →Bulk Buyer.Confirm Receipt	0.729	0.127	0.432	0.745	0.913	1.105	3.574
Bulk Buyer.Confirm Receipt →end	0.664	0.106	0.398	0.668	0.865	0.997	3.488

the endorsement time and the committing time. The reason is that the endorsement time and committing time are also greatly influenced by the transaction execution paths, but the consensus time is almost not. For a transaction, on average, the endorsement time, the consensus time and the committing time are 0.282s, 0.283s and 0.22s, respectively.

We analyze the latency of each transaction in the supply chain scenario. A transaction represents a consecutive execution path, starting from a user task (or start event) and ending at another user task (or end event). In the experiments, we denote the transaction by the starting user task (or start event) and the ending user task (or end event). For example, as shown in Fig.3, we denote a transaction by "Manufacturer.Confirm Order→Middleman.Confirm Order". It represents an execution path starting from the user task "Manufacturer.Confirm Order" and ending at the user task "Middleman.Confirm Order". In the supply chain scenario, there are 9 transactions for executing a process instance completely. We set the number of blockchain nodes as 4. The experiment results can be seen in Table III. For the transactions, the 50th percentile latency is between 0.579s and 0.812s. The 95th percentile latency is between 0.938s and 1.271s. The maximum latency is less than 3.8s. Moreover, the transaction "Carrier.Take Delivery → Manufacturer.Complete Production" has the higher latency than other transactions. The reason is that it has the longer execution path so that it takes more time in the endorsement phase and the committing phase. The transaction "start→ Bulk Buyer.Submit Requirement" has

the lower latency, because it has the shorter execution path than other transactions.

In the third set of experiments, we compare our approach with translation-based approach. We translate the process *supply chain* into chaincodes (i.e., the smart contracts of Hyperledger Fabric), and then executing this process by running the chaincodes on Hyperledger Fabric. The experiment results can be seen in Table IV. Our approach has better performance than translation-based approach. The reason is that we customize an efficient process engine running on an in-memory database. In contrast, the translation-based approach does not specifically optimize the process execution performance.

In summary, the throughput is over 1,200tps and the latency is less than 0.9s on average. More than 50% transactions are completed within 0.8s. More than 90% transactions are completed within 1s. Our approach can correctly and efficiently execute inter-organizational processes.

VI. CONCLUSION AND FUTURE WORK

In the paper, we customize a process engine to execute inter-organizational processes via a blockchain-style procedure, and then build a permissioned blockchain system by embedding the customized process engine into the blockchain nodes. Moreover, in order to achieve the interaction with the services outside the blockchain, we propose a blockchain-based approach for the service registration, binding and invocation, and then design a lease-based concurrency control protocol to reduce the transaction conflicts. Finally, we implement a prototype system based on the permissioned blockchain platform *Hyperledger Fabric* and the process engine *Activiti*. Extensive experimental results show that the proposed blockchain system can execute the inter-organizational processes correctly and efficiently.

In future work, we plan to adopt the advanced blockchain technologies, such as sharding and DAG. We also plan to study the privacy issues of inter-organizational processes.

ACKNOWLEDGMENT

The work was supported by the National Key Research and Development Program of China (2020YFB2104101) and the National Natural Science Foundation of China (U1911203, 62072459).

TABLE IV
COMPARISON WITH TRANSLATION-BASED APPROACH

Nodes	Throughput (tps)		Latency (s)	
	Customized Process Engine	Translation -based	Customized Process Engine	Translation -based
4	1,291.1	472.6	0.65	2.08
5	1,268.8	442.8	0.68	2.24
6	1,253.6	429.7	0.71	2.3
7	1,175.8	373.1	0.79	2.65
8	1,164.8	364.8	0.81	2.73
9	1,148.3	367.8	0.9	2.68
10	1,178.9	349.9	0.92	2.82

REFERENCES

- [1] Q. Lu, A. Binh Tran, I. Weber *et al.*, “Integrated Model-driven Engineering of Blockchain Applications for Business Processes and Asset Management,” *Software: Practice and Experience*, vol. 51, no. 5, pp. 1059–1079, 2021.
- [2] P. Klinger and F. Bodendorf, “Blockchain-based Cross-Organizational Execution Framework for Dynamic Integration of Process Collaborations,” in *WI Zentrale Tracks*, 2020, pp. 893–908.
- [3] I. Weber, X. Xu, R. Riveret *et al.*, “Untrusted Business Process Monitoring and Execution Using Blockchain,” in *International Conference on Business Process Management*, 2016, pp. 329–347.
- [4] C. Di Cicco, A. Cecconi, J. Mendling, and D. Felix, “Blockchain-Based Traceability of Inter-organizational Business Processes,” in *Business Modeling and Software Design*, 2018, pp. 56–68.
- [5] L. Mercenne, K.-L. Brousseau, and E. B. Hamida, “Blockchain Studio: A Role-Based Business Workflows Management System,” in *Annual Information Technology, Electronics and Mobile Communication Conference*, 2018, pp. 1215–1220.
- [6] OMG, “Business Process Model and Notation (BPMN), Version 2.0.2 (Dec 2013).”
- [7] J. Mendling, I. Weber, W. V. der Aalst *et al.*, “Blockchains for Business Process Management-Challenges and Opportunities,” *ACM Transactions on Management Information Systems*, vol. 9, no. 1, pp. 1–16, 2018.
- [8] O. López-Pintado, L. García-Bañuelos, M. Dumas *et al.*, “Caterpillar: A Business Process Execution Engine on the Ethereum Blockchain,” *Software: Practice and Experience*, vol. 49, no. 7, pp. 1162–1193, 2019.
- [9] H. Nakamura, K. Miyamoto, and M. Kudo, “Inter-organizational Business Processes Managed by Blockchain,” in *International Conference on Web Information Systems Engineering*, 2018, pp. 3–17.
- [10] D. Silva, S. Guerreiro, and P. Sousa, “Decentralized Enforcement of Business Process Control Using Blockchain,” in *Enterprise Engineering Working Conference*, 2019, p. 69–87.
- [11] T. Henry, A. Brahem, N. Laga *et al.*, “Trustworthy Cross-organizational Collaborations with Hybrid On/Off-Chain Declarative Choreographies,” in *Service-Oriented Computing*. Springer International Publishing, 2021, pp. 81–96.
- [12] M. F. Madsen, M. Gaub, T. Høgnason *et al.*, “Collaboration Among Adversaries: Distributed Workflow Execution on a Blockchain,” in *Symposium on Foundations and Applications of Blockchain*, 2018.
- [13] R. Hull, V. S. Batra, Y.-M. Chen *et al.*, “Towards a Shared Ledger Business Collaboration Language Based on Data-Aware Processes,” in *International Conference on Service-Oriented Computing*, 2016, pp. 18–36.
- [14] V. Amaral de Sousa, C. Burnay, and M. Snoeck, “B-MERODE: A Model-Driven Engineering and Artifact-Centric Approach to Generate Blockchain-Based Information Systems,” in *International Conference on Advanced Information Systems Engineering*, 2020, pp. 117–133.
- [15] A. Abid, S. Cheikhrouhou, and M. Jmaiel, “Modelling and Executing Time-Aware Processes in Trustless Blockchain Environment,” in *International Conference on Risks and Security of Internet and Systems*, 2020, pp. 325–341.
- [16] O. López-Pintado, M. Dumas, L. García-Bañuelos *et al.*, “Dynamic Role Binding in Blockchain-Based Collaborative Business Processes,” in *International Conference on Advanced Information Systems Engineering*, 2019, pp. 399–414.
- [17] C. Sturm, J. Scalanczi, S. Schönig *et al.*, “A Blockchain-based and Resource-aware Process Execution Engine,” *Future Generation Computer Systems*, vol. 100, pp. 19–34, 2019.
- [18] S. Haarmann, K. Batoulis, A. Nikaj *et al.*, “DMN Decision Execution on the Ethereum Blockchain,” in *International Conference on Advanced Information Systems Engineering*, 2018, pp. 327–341.
- [19] J. Ladleif, M. Weske, and I. Weber, “Modeling and Enforcing Blockchain-Based Choreographies,” in *International Conference on Business Process Management*, 2019, pp. 69–85.
- [20] O. López-Pintado, M. Dumas, L. García-Bañuelos *et al.*, “Interpreted Execution of Business Process Models on Blockchain,” in *International Enterprise Distributed Object Computing Conference*, 2019, pp. 206–215.
- [21] F. Corradini, A. Marcelletti, A. Morichetta *et al.*, “Engineering Trustable and Auditable Choreography-Based Systems Using Blockchain,” *ACM Transactions on Management Information Systems*, vol. 13, no. 3, Feb. 2022. [Online]. Available: <https://doi.org/10.1145/3505225>
- [22] X. Tianhong, F. Shangqing, M. Pan *et al.*, “Smart Contract Generation for Inter-Organizational Process Collaboration,” in <https://arxiv.org/abs/2303.09257>, 2023.
- [23] M. Edoardo, D. C. Claudio, and W. Ingo, “Fine-Grained Data Access Control for Collaborative Process Execution on Blockchain,” in *International Conference on Business Process Management*. Springer International Publishing, 2022, pp. 51–67.
- [24] L. García-Bañuelos, A. Ponomarev, M. Dumas *et al.*, “Optimized Execution of Business Processes on Blockchain,” in *International Conference on Business Process Management*, 2017, pp. 130–146.
- [25] H. Nakamura, K. Miyamoto, and M. Kudo, “Inter-organizational Business Processes Managed by Blockchain,” in *International Conference on Web Information Systems Engineering*, 2018, pp. 3–17.
- [26] M. Dabbagh, K.-K. R. Choo, A. Beheshti *et al.*, “A Survey of Empirical Performance Evaluation of Permissioned Blockchain Platforms: Challenges and Opportunities,” *Computers & Security*, vol. 100, 2021.
- [27] D. Basile, V. Goretti, C. Di Cicco *et al.*, “Enhancing Blockchain-based Processes with Decentralized Oracles,” in *International Conference on Business Process Management*, 2021, pp. 102–118.
- [28] O. López-Pintado, M. Dumas, L. García-Bañuelos *et al.*, “Controlled Flexibility in Blockchain-based Collaborative Business Processes,” *Information Systems*, vol. 104, 2022.
- [29] E. Androulaki, A. Barger, V. Bortnikov *et al.*, “Hyperledger Fabric: a Distributed Operating System for Permissioned Blockchains,” in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.
- [30] M. Castro, B. Liskov *et al.*, “Practical Byzantine Fault Tolerance,” in *USENIX Symposium on Operating Systems Design and Implementation*, 1999, pp. 173–186.
- [31] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts (7th Edition)*. McGraw-Hill, 2019.