Future Generation Computer Systems 101 (2019) 564–575

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

uBaaS: A unified blockchain as a service platform

Qinghua Lu [a], Xiwei Xu [a,*], Yue Liu [b], Ingo Weber [a], Liming Zhu [a], Weishan Zhang [b]

[a] Data61, CSIRO, Sydney, Australia
[b] College of Computer and Communication Engineering, China University of Petroleum (East China), Qingdao, China

ARTICLE INFO

Article history:
Received 30 August 2018
Received in revised form 11 March 2019
Accepted 19 May 2019
Available online 3 June 2019

Keywords:
Blockchain
Architecture
Design patterns
Deployment
Blockchain as a service

ABSTRACT

Blockchain is an innovative distributed ledger technology which has attracted a wide range of interests for building the next generation of applications to address lack-of-trust issues in business. Blockchain as a service (BaaS) is a promising solution to improve the productivity of blockchain application development. However, existing BaaS deployment solutions are mostly vendor-locked: they are either bound to a cloud provider or a blockchain platform. In addition to deployment, design and implementation of blockchain-based applications is a hard task requiring deep expertise. Therefore, this paper presents a unified blockchain as a service platform (uBaaS) to support both design and deployment of blockchain-based applications. The services in uBaaS include *deployment as a service*, *design pattern as a service* and *auxiliary services*. In uBaaS, *deployment as a service* is platform agnostic, which can avoid lock-in to specific cloud platforms, while *design pattern as a service* applies design patterns for data management and smart contract design to address the scalability and security issues of blockchain. The proposed solutions are evaluated using a real-world quality tracing use case in terms of feasibility and scalability.

1. Introduction

Blockchain technology has attracted a wide range of interests as a distributed ledger technology for establishing digital trust in business. Many start-ups, enterprises, and governments [1,2] are currently exploring how to leverage blockchain technology to achieve trust and decentralization in the next generation of applications. Blockchain-based application areas are diverse, including physical or digital asset ownership management, tokens, currency, identity management, supply chain, electronic health records, voting, energy supply, and more.

Blockchain as a service (BaaS) is a promising solution to improve the productivity of blockchain application development. Easier deployment is the primary service offered by the existing BaaS platform providers, e.g. Microsoft Azure,[1] IBM,[2] and Amazon.[3] However, the existing BaaS deployment solutions are usually vendor locked, which are bound to either a cloud vendor (e.g. Microsoft Azure[1]) or a blockchain platform (e.g. IBM Hyperledger[2]).

In addition to deployment, the design and implementation of blockchain-based applications are challenging to developers. First, blockchain is a new technology with limited tooling and documentation, so there can be a steep learning curve for developers. According to a survey by Gartner [3], "23 percent of [relevant surveyed] CIOs said that blockchain requires the most new skills to implement any technology area, while 18 percent said that blockchain skills are the most difficult to find". Second, blockchain is by design an immutable data store, so updating deployed blockchain smart contracts can be hard. This makes it difficult to fix bugs by releasing new versions of smart contracts. Mistakes in smart contracts have led to massive economic loss such as the DAO exploit on the Ethereum blockchain [4].

A software design pattern is defined as a solution to a problem that commonly occurs within a given context during software design [5]. Design patterns for blockchain-based applications are best practices from industry and can be encapsulated services to ease the burden of developers and improve the quality of blockchain-based applications. One motivating example is the on-chain and off-chain design pattern, which provides a solution for storing data of blockchain-based applications. The purpose of separately storing data on-chain and off-chain is to ensure the integrity of on-chain data and privacy of the off-chain data. This pattern can be implemented as a service to generate an on-chain data registry smart contract and an off-chain data table in the conventional database based on the data model built by the developers. For example, in a quality tracing system, the on-chain attributes could be the traceability identifier and traceability result, while the off-chain attributes could be product price, buyer name, etc.

* Corresponding author.
E-mail addresses: qinghua.lu@data61.csiro.au (Q. Lu), xiwei.xu@data61.csiro.au (X. Xu).

[1] https://azure.microsoft.com/en-gb/solutions/blockchain/.
[2] https://www.ibm.com/blockchain/platform/.
[3] https://aws.amazon.com/blockchain/.

https://doi.org/10.1016/j.future.2019.05.051
0167-739X/

This paper presents a unified blockchain as a service platform named uBaaS, which facilitates the design and deployment of blockchain-based applications. The contributions of this paper are as follows.

- A set of design patterns for data management and smart contract design of blockchain-based applications to better take advantage of blockchain technology in practice. The design patterns include *on-chain and off-chain*, *hash integrity*, *data encryption*, *multiple authorities*, *dynamic binding*, and *embedded permission*.
- The uBaaS platform approach:
  - *Deployment as a service* which includes a blockchain deployment service and a smart contract deployment service. Deployment as a service can ease blockchain network deployment and smart contract deployment. The proposed blockchain deployment service is platform agnostic, which can avoid lock-in to specific cloud platforms.
  - *Design pattern as a service* which consists of *data management services* and *smart contract design services*. Each service is designed based on a design pattern to better leverage the unique properties of blockchain (i.e. immutability and data integrity, transparency) and address the limitations (i.e. privacy and scalability).
- Feasibility and scalability of the proposed solutions are evaluated using a real-world quality tracing use case. The evaluation results show that our solutions are feasible and have good scalability.

The remainder of this paper is organized as follows. Section 2 discusses the background and related work. Section 3 summarizes the design patterns for data management and smart contract design. Section 4 presents the overall architecture of uBaaS platform and design of each service provided by uBaaS. Section 5 introduces the implementation details. Section 6 evaluates the proposed solutions in terms of feasibility and scalability. Section 6 concludes the paper and outlines the future work.

2. Related work

This section introduces the related work of our research. Blockchain technology and its classification are presented first, as it is the basis of this research. Further, how to apply blockchain technology to software systems is demonstrated, after which there is a brief introduction of the current obstacles to develop blockchain-based applications. Lastly design patterns and blockchain as a service are discussed.

2.1. Blockchain technology

Blockchain is the technology behind Bitcoin [6], which is a decentralized data store that maintains all historical transactions of the Bitcoin network. The concepts of blockchain have been generalized to distributed ledger systems that verify and store transactions without coins or tokens [7], without relying on any central trusted authority, e.g. traditional banking systems. Instead, all participants in the blockchain network can reach agreement on the states of transactional data to achieve trust.

The data structure of blockchain is an ordered list of identifiable blocks, each of which is connected to the previous block in the chain. Blocks are containers aggregating transactions while transactions are identifiable data packages that store parameters (such as monetary value) and function calls to smart contracts. A smart contract is a user-defined program that is deployed and executed on the blockchain network [8], which can express triggers, conditions and business logic [9] to enable more complex programmable transactions. Smart contracts can be implemented as part of transactions, and are executed across the blockchain network by all connected nodes. The blockchain platform Ethereum provides a built-in Turing-complete scripting language for writing smart contracts, called Solidity. The Ethereum Virtual Machine (EVM) is the execution environment for Ethereum, which comprises all full nodes on the network and executes bytecode compiled from Solidity scripts.

Table 1
Types of blockchain (⊕: Less favourable, ⊕⊕: Neutral, ⊕⊕⊕: More favourable).

Type	Impact			
	Fundamental properties	Cost efficiency	Performance	Flexibility
Public blockchain	⊕⊕⊕	⊕	⊕	⊕
Consortium blockchain	⊕⊕	⊕⊕	⊕⊕	⊕⊕
Private blockchain	⊕	⊕⊕⊕	⊕⊕⊕	⊕⊕⊕

2.2. Types of blockchain

As shown in Table 1, there are three types of blockchain in terms of deployment, including public blockchain, consortium blockchain, and private blockchain. Public blockchains, which are used by most digital currencies, can be accessed by anyone on the Internet. Using a public blockchain achieves better data transparency and auditability, but sacrifices performance and has a different cost model. A consortium blockchain is typically used across multiple organizations and has pre-authorized nodes to control the consensus process. In a private blockchain network, write permissions are often kept within one organization, although this may include multiple divisions of a single organization. The right to read the consortium or private blockchain may be public or may be restricted to a specific group of participants. When using a consortium or private blockchain, a permission management component is required to authorize participants within the blockchain network. Private blockchains are the most flexible for configuration because the network is governed and hosted by a single organization. Our work is mainly focused on consortium blockchains and private blockchains.

2.3. Blockchain as a component of software application systems

Researchers in academia and developers in industry are investigating and exploring how to build next generation applications using blockchain technology [1,2]. Application areas in industry include but are not limited to digital currency, international payments, registries, government identity and taxation management, Internet of Things (IoT) identify and security management, and supply chain [10–13]. Furthermore, there is various academic work that exploits blockchain to address issues in different domains. Qu et al. [14] propose spatio-temporal blockchain technology that supports fast query processing, which is proved to be applicable and effective. Zyskind et al. use blockchain to build a personal data management system that ensures users own and control their data in a decentralized way [15]. Bulat Nasrulin et al. proposed a mobility analytics application that is built on top of blockchain [16] and renovated blockchain with distributed database [17]. Namecoin [18] and PKI [19] are two public key platforms built on blockchain. ProvChain enables data Provenance for cloud-based data analytics [20].

Software components are the fundamental building blocks for software architecture, and blockchain can be a software component offering computational capabilities. Blockchains are complex, network-based software components, which can provide data storage, computation services, and communication services.

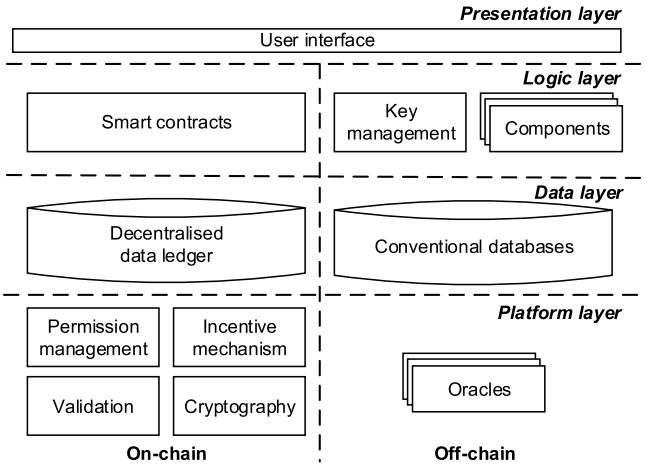


Fig. 1. Blockchain as a component of software application systems.

Fig. 1 illustrates the architecture of a blockchain-based application system, which consists of four horizontal layers and two vertical layers. The horizontal layers include a presentation layer, a logic layer, a data layer, and a platform layer, while the vertical layers are on-chain and off-chain.

The users interact with blockchain smart contracts and off-chain components (including the key management component) via the user interface. The blockchain-based application system can implement business logic through different off-chain components and on-chain smart contracts. Key management is an essential off-chain component in such blockchain-based system. Every participant in a blockchain network has one or more private keys, which are used to digitally sign the transactions. The security of these private keys is important: if the private key is stolen, assets held by the respective account can be accessed and protected functions of smart contracts can be invoked.

At the data layer, blockchain acts as a decentralized data ledger to store data which requires integrity and/or transparency. Off-chain conventional databases are often needed to store private or large-size data due to the scalability and privacy.

At the platform layer, the fundamental blockchain features can include permission management, incentive mechanisms, transaction validation, and cryptographically-secure payment. The oracles supply information about the external world to the blockchain, usually by adding that information to the blockchain as data in a transaction.

2.4. Challenges of blockchain application development

Blockchain is an emerging distributed ledger technology which requires developers to learn new skills and have a deep understanding of the technology in order to build blockchain-based applications. We summarize the challenges of blockchain application development as follows.

- Deployment. The current blockchain deployment solutions (e.g. Microsoft Azure[1], IBM Hyperledger[2], Amazon[3]) lock customers in to specific cloud and/or blockchain platforms. However, many enterprises or governments require building blockchain-based applications on their own on-premise private cloud, which are not met by the existing blockchain solutions. Besides, the deployment process of blockchain is error-prone, time-consuming, and requires frequent updating.
- Scalability. Blockchain has limited storage capability since it contains a full history of all the transactions across all participants of the blockchain network. Thus, the size of blockchain continues to grow. The ever-growing size of blockchain is a challenge for storing data on blockchain. Also, storing large amounts of data or deploying large smart contracts within a transaction may be impossible due to the limited size of the blocks of the blockchain, which is under control of the network [21].
- Data privacy. Blockchain-based applications might have sensitive data which should be only available to some certain blockchain participants. However, the information on blockchain is designed to be accessible to all the participants. There is no privileged user within the blockchain network, whether the blockchain is public, consortium or private.
- Key management. Authentication on blockchain is achieved by digital signatures. However, blockchain does not offer any mechanism to recover a lost or a compromised private key. Losing a key results in permanent loss of control over an account, and potentially smart contracts that refer to it.
- Permission control. All the smart contracts deployed on blockchain can be accessed and called by all the blockchain participants by default. A permission-less function might be triggered by unauthorized users accidentally, which becomes a vulnerability of blockchain-based application.

2.5. Design patterns and blockchain as a service

A design pattern is a reusable solution to a problem that commonly occurs within a given context during software design [22]. There are a few works on design patterns for blockchain-based applications. J. Eberhardt and S. Tai present a group of patterns which mainly focus on on-chain and off-chain data and computation [23]. Zhang et al. applies four existing object-oriented software patterns to smart contract design in the context of a blockchain-based health care application [24]. Liu et al. summarized eight smart contract design patterns and classified them into four categories [25]. Bartoletti and Pompianu conduct an empirical analysis of smart contracts, in which they collected hundreds of smart contracts and divided them into several categories: token, authorization, oracle, randomness, poll, time constraint, termination, math and fork check [26]. In recent work, some of the authors of this paper proposed an extensive pattern collection [27]. However, the summarized design patterns for blockchain-based applications are mostly at conceptual level, requiring users to implement the solutions.

Blockchain as a service (BaaS) provides an offering that allows developers to develop blockchain-based applications efficiently. Most of the current BaaS platforms are designed to help developers create, deploy, and manage blockchain network, e.g. Microsoft Azure[1], IBM[2], and Amazon[3]. However, the existing BaaS deployment solutions are usually locked into a specific public cloud or blockchain network provider (e.g. Microsoft Azure[1], IBM Hyperledger[2]). Many governments or enterprises deploy their applications in private clouds and may have different blockchain network preferences. Besides, in addition to deployment services, development services are also required for BaaS platforms. Architecture design of blockchain application is a challenge to developers since they need to learn blockchain programming languages and have a deep understanding of blockchain technology.

3. Design patterns for blockchain-based applications

In software engineering, a design pattern is a reusable solution to a problem that commonly occurs within a given context during

software design [5]. In this section, we summarize and categorize six design patterns which can be applied to the architecture design of a blockchain-based application system. Those patterns are divided into *data management design patterns* and *smart contract design patterns* according to their characteristics and effects.

3.1. Data management design patterns

As discussed in Section 2.3, in a blockchain-based application system, blockchain can act as a decentralized data ledger and work with conventional databases to store data. Data management of blockchain-based application is challenging due to the fundamental properties (e.g. data transparency) and limitations of blockchain (e.g. blockchain scalability). Here we summarize three design patterns for data management: *on-Chain and off-Chain*, *hash integrity* and *data encryption*.

3.1.1. On-chain and off-chain

Summary: The on-chain and off-chain pattern separately stores data on blockchain and off blockchain to ensure the data integrity while addressing the storage capability issue of blockchain.

Context: Some applications consider leveraging blockchain to ensure data integrity since all the data on blockchain are transparent and immutable.

Problem: It may be impossible to store sensitive and large amounts of data on blockchain since all the information on blockchain is accessible to all participants and blockchain has limited storage capacity.

Solution: Rather than placing all the data on-chain, only the sensitive and small data is stored on-chain. For example, a food quality tracing system can store traceability information that is required by the traceability regulation (e.g. traceability number and results) on-chain, while places the factory production process photos off-chain. The benefits of separately storing data on blockchain and off blockchain is to better leverage the properties of blockchain and avoid the limitations of blockchain. Blockchain can guarantee the integrity and immutability of the critical data on-chain. The non-tiny files are stored off-chain so that the size of the blockchain would not grow so fast. Storing the hashes of off-chain files can further ensure the integrity of the off-chain files.

3.1.2. Hash integrity

Summary: The hash integrity pattern uses hashing to ensure the integrity of arbitrarily large datasets which may not fit directly on the blockchain.

Context: Some blockchain-based applications consider using blockchain to ensure the integrity of large amounts of data.

Problem: It may be impossible to store large amounts of data within a transaction since the blocks of blockchain has limited size (e.g., Ethereum has a block gas limit to control the data size, computational complexity and number of transactions included in a block). There is a problem about how to store arbitrary size data on blockchain to guarantee data integrity.

Solution: For data of large size (essentially data that is bigger than its hash value), rather than storing the raw data directly on blockchain, a hash value of the raw data is stored on blockchain. The hash value is produced by a hash function which maps data of arbitrary size to data of fixed size and is non-invertible. Any change to the data will lead to a change in its corresponding hash value.

3.1.3. Data encryption

Summary: The data encryption pattern ensures confidentiality of the data stored on blockchain by encrypting it.

Context: For some blockchain-based applications, commercially sensitive data should be only accessed by specific participants. An example would be a special discount price offered by a service provider to a subset of its users. Such information might not be supposed to be accessible to the other users who do not get the discount.

Problem: The lack of data privacy is one of the main limitations of blockchain. All the information on blockchain is publicly available to the participants of the blockchain. There is no privileged user within the blockchain network, no matter the blockchain is public, consortium or private. On a public blockchain, new participants can join the blockchain network freely and access all the information recorded on blockchain. Any confidential data on public blockchain is exposed to the public.

Solution: Asymmetric(or symmetric) encryption can be used to encrypt data before storing the data on blockchain. One of the involved participants generate a key pair and distribute the decryption key to other involved participants. The involved participants can encrypt the data before placing it on blockchain using the encryption key. Only the involved participants who have the decryption key can decrypt the data.

3.2. Smart contract design patterns

Developers can define smart contracts (i.e. software programs on blockchain) to implement business logic and enable more complex programmable transactions. However, developing smart contracts usually require developers to have a deep understanding of blockchain and rich smart contract development experience. Thus, we summarize three design patterns as solutions to address the common problems (e.g. security) in the design of smart contracts.

3.2.1. Multiple authorities

Summary: A set of blockchain account addresses which can authorize a transaction is pre-defined. Only a subset of the predefined account addresses is required to authorize transactions.

Context: Some activities in blockchain-based applications might need to be authorized by multiple parties (i.e. blockchain account addresses). For example, a monetary transaction may require authorization from multiple blockchain account addresses.

Problem: The actual addresses that authorize an activity might not be able to be determined due to the availability of the authorities.

Solution: To enable more flexible binding, an M-of-N mechanism can be used to define that M out of N private keys are required to authorize the transaction. M is the threshold of authorization.

3.2.2. Dynamic binding

Summary: The dynamic binding pattern uses a hash created off-chain to dynamically bind authority for a transaction.

Context: In blockchain-based applications, some activities need to be authorized by one or more participants that are unknown when the corresponding smart contract is deployed or the transaction is submitted to blockchain.

Problem: Blockchain does not support dynamic binding with a blockchain account address which is not defined in the transaction or smart contract. All accounts that can authorize a second

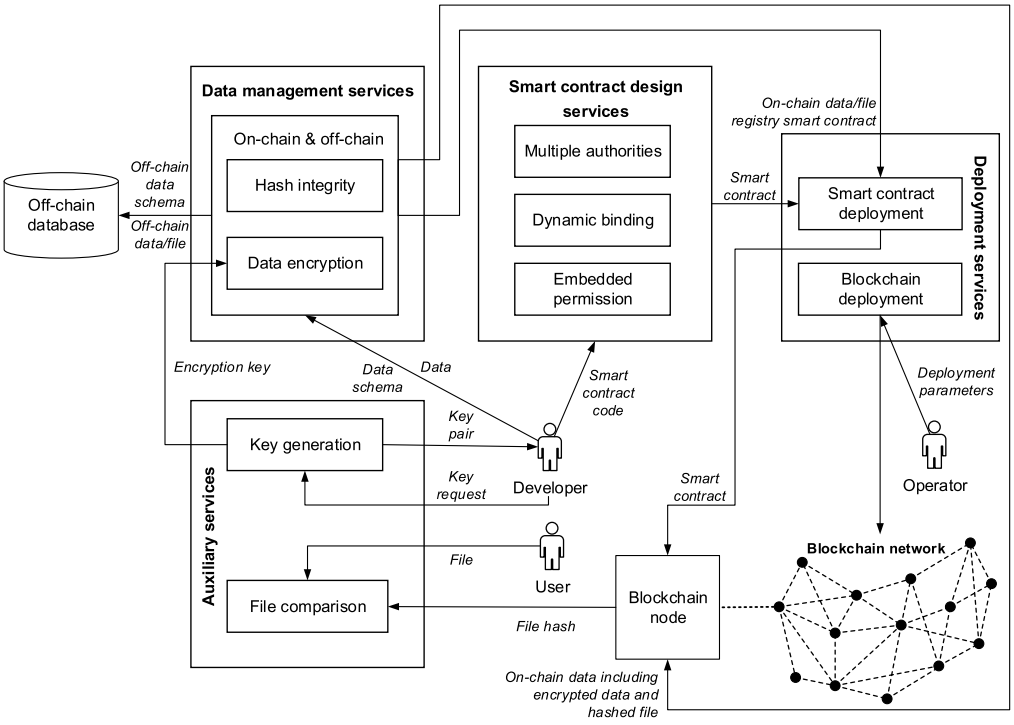


Fig. 2. Architecture of uBaaS.

transaction have to be defined in the first transaction before that transaction is added to the blockchain.

Solution: An off-chain secret can be used to enable a dynamic binding when the participant authorizing a transaction is unknown beforehand. In the context of payment, when the sender deposits money to an escrow smart contract, a hash of a secret is submitted with the money as well. The participant who receives the secret off-chain can claim the money from the escrow smart contract by revealing the secret. Thus, the receiver of the money does not need to be defined beforehand in the escrow contract.

3.2.3. Embedded permission

Summary: Smart contracts use an embedded permission control to restrict access to the invocation of the functions defined in the smart contracts.

Context: A smart contract by default has no owner, since the smart contracts running on blockchain can be accessed and called by all the blockchain participants and other smart contracts by default.

Problem: Once the smart contract is deployed, the author of the smart contract has no special privilege to invoke on the smart contract. A permission-less function can be triggered by unauthorized users accidentally, which becomes a vulnerability of blockchain-based application. For example, a permission-less function is discovered in a smart contract library used by the Parity multi-sig wallet, caused the freezing of about 500 K Ether.[4] In 2016, 7% smart contract on public Ethereum could be terminated without authority [28].

Solution: Permission control can be added to every smart contract function to check permissions for every function caller based on the blockchain address of the caller before executing the logic of the function. Calls from unauthorized blockchain addresses are rejected.

[4] https://paritytech.io/a-postmortem-on-the-parity-multi-sig-library-self-destruct/.

4. Architecture of uBaaS

In uBaaS, we propose deployment as a service for vendor-independent deployment and design pattern as a service to address the scalability and security issues of blockchain-based applications. Fig. 2 illustrates the overall architecture of uBaaS. The services proposed in uBaaS are classified into three categories, *deployment as a service*, *design pattern as a service* and *auxiliary services*. Users can build up blockchain environment and design blockchain-based applications via uBaaS front-end user interface, which interacts with the back-end services through an API gateway. The API gateway forwards API calls from the front-end user interface to the corresponding services.

The remainder of this section introduces *deployment as a service*, *design pattern as a service* and *auxiliary services* respectively. *Deployment as a Service* consists of 2 kinds of deployment services, *design pattern as a service* includes 6 design pattern services to help developers take advantage of blockchain's properties and address blockchain's limitations, while *auxiliary services* can act as assistance to better leverage design pattern as a service.

4.1. Deployment as a service

Deployment as a service includes *blockchain deployment service* and *smart contract deployment service*. Users can configure the blockchain settings (such as difficulty and participant node IP) and deploy their customized blockchain network using uBaaS. Infrastructure-as-code is applied to the blockchain deployment service which enables the automation of deployment, configuration, and task management by the developed script. Once the blockchain is set up, users can monitor the status of blockchain at real-time and deploy the developed smart contracts on the blockchain. We assume we can access the participant nodes since we focus on consortium blockchain and private blockchain. Currently, both Ethereum blockchain and Hyperledger Fabric blockchain are supported. We plan to add more blockchain platforms as deployment options in uBaaS.

The smart contract deployment service enables users to select the developed smart contract (i.e. program) file and deploy the smart contract code on the blockchain. Besides, the smart contract address and Application Binary Interface (ABI) are stored in the off-chain database for invoking the deployed smart contract.

4.2. Design pattern as a service

Design pattern as a service consists of *data management services* and *smart contract design services*. *Data management services* include *on-chain and off-chain service*, *data encryption service*, and *hash integrity service*, while *smart contract design services* comprises *multiple authorities service*, *dynamic binding service*, *embedded permission service*. Each type of services is designed based on a design pattern to improve scalability, adaptability, and security of blockchain-based applications. The *data management services* enable users to manage data directly via uBaaS front-end user interface while the *smart contract design services* integrate smart contract design patterns into the original smart contract.

4.2.1. Data management services

To address the issues of blockchain storage capability limitation and data privacy, an *on-chain and off-chain service* is proposed to store the critical data which is required to be immutable on-chain while keep all the data off-chain to enhance the data reading efficiency. Users can define the data schema and determine which attributes are stored on-chain and off-chain. Based on the data model built by users, the service builds up a data registry on blockchain by deploying the generated on-chain data registry smart contract and sets up an off-chain data table in the conventional database. The on-chain data registry and off-chain data table share the same name. The user can write/read data to/from the selected data store regardless to being stored on-chain or off-chain.

To preserve the privacy of the involved participants, uBaaS provides a *data encryption service* that encrypts on-chain data to ensure confidentiality of the data stored on blockchain. The uBaaS user first encrypts the data item using the private key (generated by the key generation service in auxiliary services) and then stores it on blockchain. The blockchain participants who have the public key are allowed to access the transaction and decrypt the information. By using the on-chain data encryption service, the sensitive data stored on blockchain are not accessible to blockchain participants who do not hold the public key.

To store the large size data (i.e. file) on blockchain, the *hash integrity service* generates the hash value of the file and stores the hash in the on-chain file registry which is associated with the on-chain data registry using the pre-defined foreign key.

4.2.2. Smart contract design services

The *smart contract design services* focus on the permission control for invocation of the smart contract functions. The input of each of those services is the smart contract code written by the user while the output is updated smart contract code by adding the template code implementing the corresponding smart contract design pattern.

The *multiple authorities service* focuses on the smart contact functions that can be invoked only when the authorized blockchain addresses approve. Users can predefine a group of blockchain addresses which can authorize a transaction (i.e. calling a function in the smart contract) and set the minimal number of authorizations for transaction approval. The users need to select a smart contract they write and the function which needs the mechanism of multiple authorities. Then uBaaS modifies the code of the selected function and generate an updated smart contract with code for multiple authorities.

The *dynamic binding service* uses an off-chain secret to enable a dynamic authorization when the participant approving a transaction (i.e. calling a function in the smart contract) is unknown beforehand. Users need to provide an secret (e.g. a random number) and select the corresponding function in the smart contract. The service modifies the function code by adding the hash of the secret and generate the updated smart contract. There is no need for a special protocol to exchange the secret as it can be exchanged in any ways off-chain. Only the user who has the secret can invoke the selected function in the smart contract.

The purpose of *embedded permission service* is to restrict access to the invocation of the functions defined in the smart contracts. Users can identify the authorities for the selected function in the smart contract by providing the authority addresses. The service adds permission control code to the smart contract function to check permissions for every caller based on the blockchain addresses of the caller, which is done before executing the function logic.

4.3. Auxiliary services

The current version of uBaaS supports two auxiliary services, *key management service* and *file comparison service*.

The *key management service* is used to generate key pairs for data encryption. The developers can encrypt the data using the data encryption service and share the decryption key with other platform users (e.g. developers or application users) before store the encrypted data in the on-chain data registry. The channel for sharing the decryption key is out of the scope in this paper.

The purpose of *file comparison service* is to validate the authenticity of a file (e.g. higher education certificates). Users can select the file from local node and check the authenticity of the file by comparing its hash value with the hash value of the associated original file stored in the on-chain file registry.

5. Implementation

Fig. 3 illustrates the implementation design of uBaaS using the class diagram. *BlockchainRegistry* maintains each deployed *Blockchain* network, while *OffChainSmartContractRegistry* stores the source code and information of *SmartContract*. There are three classes that inherit from *SmartContract*: *OnChainDataRegistry*, *OnChainFileRegistry* and *DesignedContract*. Each *OnChainDataRegistry* is associated with an *OffChainDataRegistry* for data storage. User-defined data record attributes are contained in *Record*, and *OnChainDataRegistry* and *OffChainDataRegistry* are composed of *Record*. Note that the attribute values stored in *Record* must be encrypted using *KeyPair* before storing in *OnChainDataRegistry*. *File* is stored in both *OffChainFileRegistry* and its hash value is stored in *OnChainFileRegistry*. *DesignedContract* applies smart contract design patterns to the smart contract code written by users.

The platform is developed in Java 1.8 using Eclipse Java IDE 4.6.0 and released using Tomcat v7.0 server. To achieve blockchain deployment as a service, we used SSH to transfer the deployment files, including the client file (e.g. Geth for Ethereum) and genesis block file (e.g. genesis.json for Ethereum), to the participant nodes. To implement data management services, we selected MySQL 5.7.17 as the supported database to store off-chain data. Regarding smart contract design services, the smart contract design pattern templates are pre-developed and stored in the database. The platform users can select the smart contract design pattern that they want to apply. For both data management services and smart contract design services, the smart contracts are written in Solidity, compiled with Solidity compiler version 0.4.24. After compilation, a smart contract is deployed on blockchain via web3 API, returning smart contract address as result if the deployment succeeds.

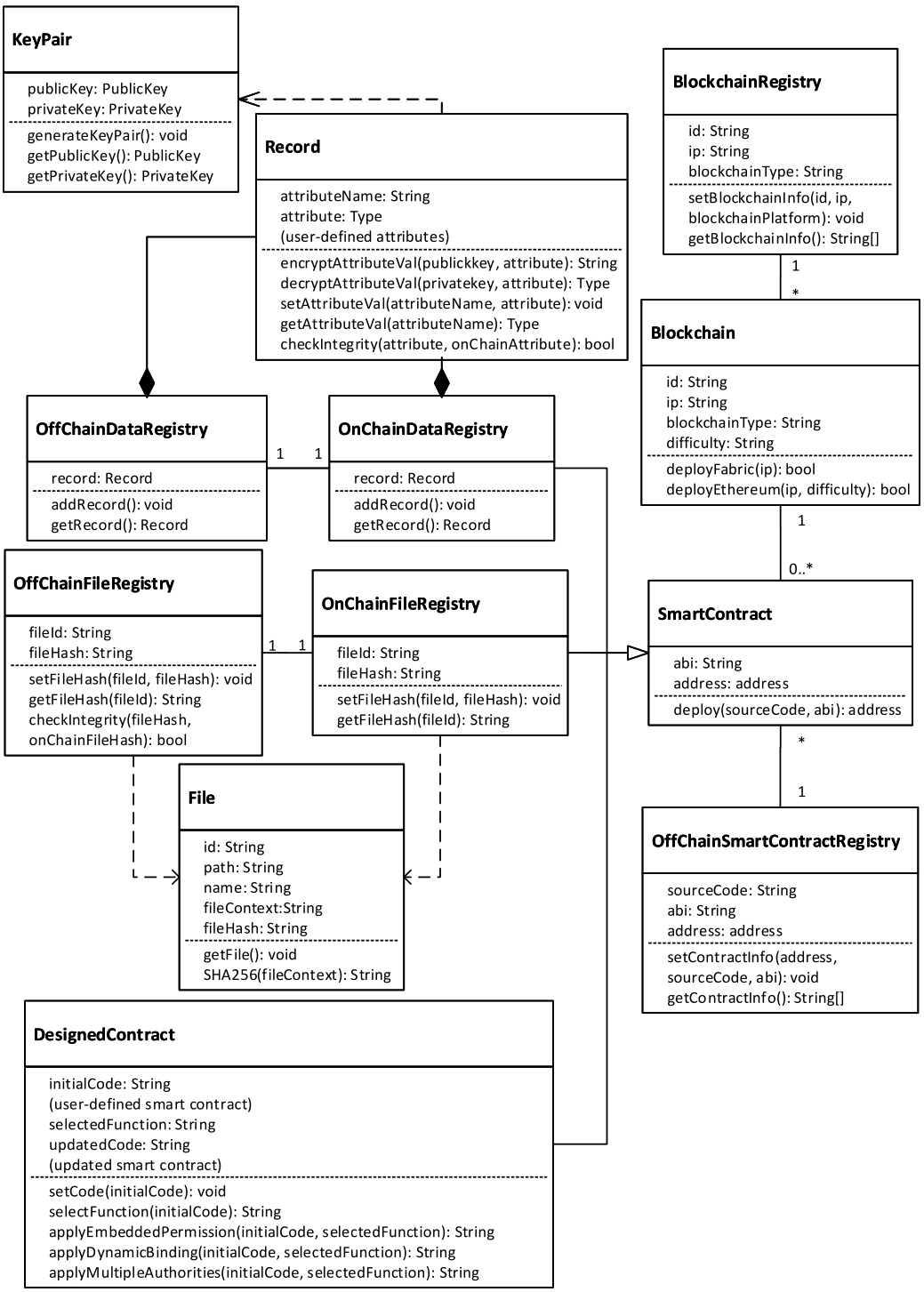


Fig. 3. The main class diagram of uBaaS implementation.

6. Evaluation

In this section, we evaluate the feasibility and scalability of uBaaS. We first introduce the experiment environment. Then we evaluate the feasibility and scalability of the *blockchain deployment service* and *data management services* in uBaaS. Finally, we evaluate the feasibility of *smart contract design services* in uBaaS using a real-world quality tracing use case.

6.1. Experiment environment

The uBaaS platform was deployed on an Alibaba Cloud[5] virtual machine (2 vCPUs, 8G RAM, 20 GB disk). The adopted blockchain is Ethereum 1.5.9-stable, in which the consensus algorithm is Proof-of-Work (PoW). The selected database for off-chain data storage is MySQL 5.7.17. We set up 100 Alibaba Cloud virtual machines (1 vCPUs, 1G RAM) as the blockchain nodes (the number of nodes varies based on the experiment requirements).

5 https://www.aliyun.com/.

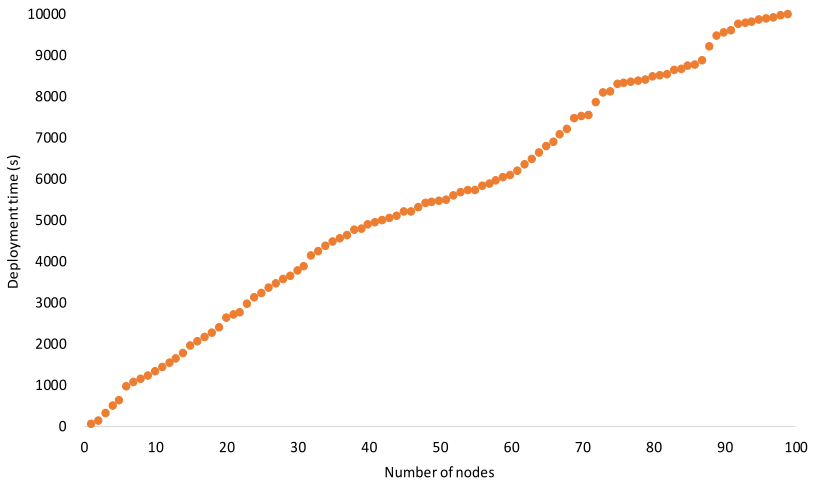


Fig. 4. Blockchain deployment time.

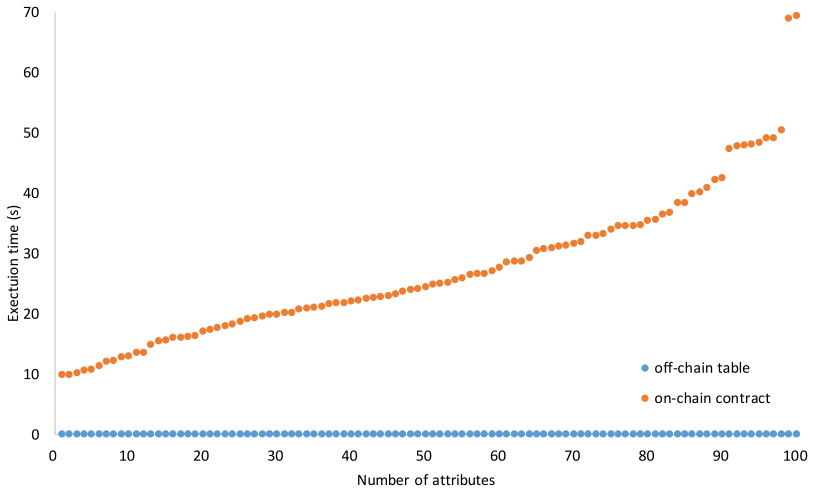


Fig. 5. Execution time of generating an off-chain data table and an on-chain data registry smart contract.

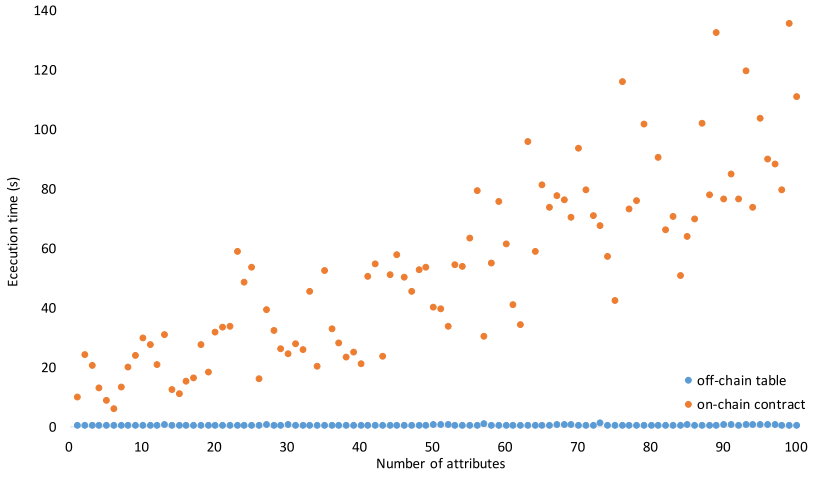


Fig. 6. Execution time of writing data to the off-chain data table and the on-chain data registry smart contract.

6.2. Evaluation of blockchain deployment service

We evaluated the scalability of blockchain deployment service by measuring the deployment time of blockchain with different number of nodes. We set the blockchain type as Ethereum and the difficulty as 0*x*4000, and filled in the IP addresses of the nodes for deployment. According to the settings, uBaaS automatically deployed the Ethereum blockchain on the target nodes.

Fig. 4 shows the measurement results for deployment time of Ethereum blockchain using uBaaS with increased number of nodes. The deployment time increased almost linearly, showing good scalability. The experiment results also show that the

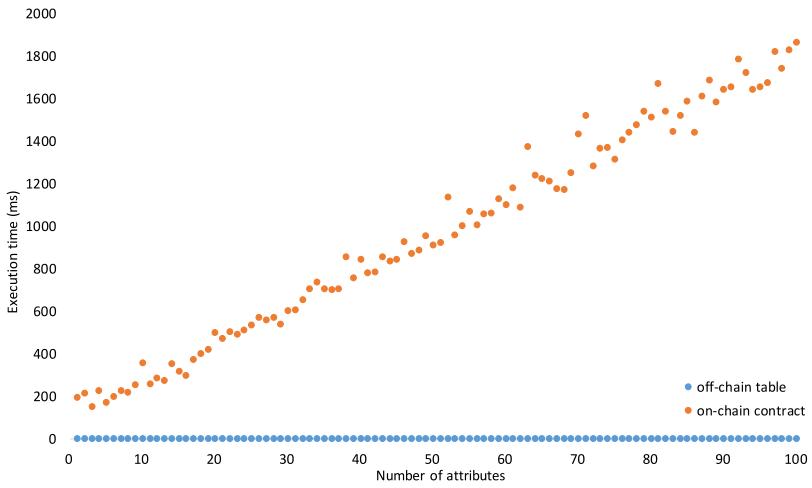


Fig. 7. Execution time of reading data from the off-chain table and the on-chain smart contract.

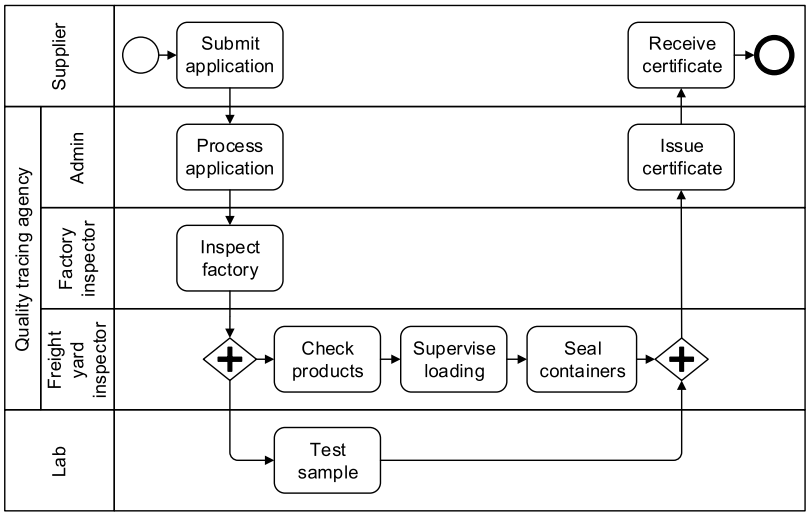


Fig. 8. Quality tracing process.

blockchain deployment service in uBaaS is feasible.

6.3. Evaluation of data management services

The main functionality provided by the data management services include: (1) generating an on-chain data registry and an off-chain data table in the conventional database based on the data model built by the developers; (2) writing/reading data to/from the on-chain data registry and off-chain data table. Therefore, we evaluated the data management services in a two-fold way: First, we measured the execution time of creating a data table off-chain in a remote conventional database and generating the corresponding data registry smart contract on-chain; Then, we tested the execution time of writing/reading data to/from the off-chain table and on-chain smart contract respectively. Note that the current version of uBaaS encrypts all the data before storing them on-chain. Thus, the time of writing data and reading data to/from on-chain data registry include data encryption and decryption.

Fig. 5 illustrates the execution time of creating an off-chain data table and generating the associated on-chain data registry with the increased number of data record attributes. The execution time of generating a smart contract on blockchain is much higher than creating a table in the conventional database, due to the Proof of Work (POW) mechanism in Ethereum. In addition, 96.7% of the on-chain data registry smart contracts are generated within 50 s. The on-chain data registry smart contract generation time is fluctuating because block generation time varies around an average value according to the difficulty setting of Ethereum.

Fig. 6 demonstrates the execution time of writing data to an off-chain data table and to the associated on-chain data registry smart contract with increased number of attributes. The execution time increased linear, which shows good scalability. Compared to writing data to the off-chain data table in the conventional database, storing data to the on-chain data registry smart contract is much slower since it is time-consuming to generate a new block and include the transactions in the block. The fluctuation of writing data to the deployed smart contract is also because the block generation time is unstable as mentioned above.

Fig. 7 shows the execution time of reading data from the on-chain data registry smart contract, which is much shorter than generating an on-chain data registry smart contract and writing data to the deployed smart contract, as reading is from the local node of blockchain. It is still higher than reading data from traditional database, as the data stored on-chain needs to be decrypted before being presented to the user in uBaaS.

The experiment results in Figs. 5–7 also show the feasibility of the data management services in uBaaS.

6.4. Evaluation of smart contract design services

We evaluated the feasibility of the smart contract design services using a real world quality tracing process. Fig. 8 illustrates the quality tracing process for import commodities in China [29]. The quality tracing agency accredited by the Chinese government provides quality tracing services and issues traceability certificates of commodity if all requirements are fulfilled. The process starts when a product supplier submits a quality tracing application to the agency. The administrator processes the application paper work (e.g. invoices) and payment. Once the application is validated, the agency assigns a factory inspector to check the factory location, production capability, quality control process, etc. After inspecting the factory, a freight yard inspector is sent to examine the products placed in the freight yard and to inspect the on-site loading process. The inspector seals the containers if the process of on-site loading complies with regulations. In the meantime, a product sample is sent to the lab for sample testing. Once the application passes the inspections and testing, the quality tracing agency issues the supplier a traceability certificate of commodity. In the quality tracing system, the quality tracing agency manages the quality tracing process and data while the lab submits testing reports through the quality tracing system. The quality inspection bureau mainly monitors the quality tracing process and does not provide any input to this system.

```
contract MultipleAuthorities{
  uint total;
  address[] authority;
  bool agreeing;
  uint agreeThreshold;
  mapping(address => bool) agreeState;
  bool agreePermission;
  address agreeRequester;
  ...
  function agreeSignature(){
    agreeState[msg.sender] = true;
    if(agreeResult())
      agreePermission = true;
  }
  function agreeResult() internal returns (bool signatureResult){
    uint k = 0;
    for(uint i = 0; i <total; i++)
      if(agreeState[authority[i]] == true)
        k++;
    if(k >= agreeThreshold)
      return true;
    else
      return false;
  }
  function initialAgree() internal{
    ...
  }
  modifier isEnoughAgreement(){
    if(agreeing == true && agreePermission == true && msg.sender ==
     agreeRequester){
      _;
    initialAgree(); }
  }
  ...
}

contract SampleTesting is MultipleAuthorities{
  string sampleID;
  bool passed;
  function sampleTest(string ID){
    sampleID = ID;
    passed = false;
  }
  function pass() isEnoughAgreement(){
    passed = true;
  }
  ...
}
```

Listing 1: Updated *SampleTesting* code after using the *multiple auhtorities* service.

According to the design of the quality tracing system, we deployed an Ethereum blockchain on three nodes, which represent a node in the quality tracing agency, a node in the lab, and a node in the quality inspection bureau. We configured the difficulty as 0*x*4000 and provided the IP addresses of three nodes in uBaaS. Then uBaaS automatically deployed an Ethereum blockchain on the three target nodes.

```
contract DynamicBinding{
  bytes32 hashKey;
  bool init;
  address owner;
  function initial(bytes32 key){
    if(init != true){
      hashKey = key;
      init = true;
      owner = msg.sender;
    }
  }
  function changeKey(string oldKey , bytes32 newKey){
    if(init == true)
      if(hashKey == sha256(oldKey))
        if(owner == msg.sender)
          hashKey = newKey;
  }
  modifier verify(string inputKey){
    if(hashKey == sha256(inputKey)){ _; }
  }
 }

contract ServiceAgreement is DynamicBinding{
  string firstParty;
  string secondParty;
  bytes32 contractHash;
  ...
  function queryAgreement(string key)
       verify(key) constant returns (string, string,
     bytes32) {
        return firstParty, secondParty, contractHash;
  }
  ...
}
```

Listing 2: Updated *ServiceAgreement* code after using the *dynamic binding* service.

As discussed in Section 4, uBaaS uses the smart contract design services to restrict access to the invocation of the functions in the smart contracts. Thus we selected three different functionalities (i.e. sample testing result approval, service agreement query and freight yard picture storage) to apply each of the proposed smart contract design services: *multiple authorities service*, *dynamic binding service*, and *embedded permission service*. We wrote three smart contracts implementing the business logic in those three scenarios and select a function in each of the three smart contract to apply each smart contract design service. Thus, the output of uBaaS is the updated smart contract code with permission control for the smart contract function invocation. List 1–3 show the generated code by uBaaS. The code in black colour represents the input smart contract code (i.e. the smart contract code written by the developers), while the code highlighted in red colour is the code generated by the corresponding smart contract design services in uBaaS.

```
contract EmbeddedPermission{
  address [] authority;
  address owner;
  function EmbeddedPermission(address [] temAuthority){
    owner = msg.sender;
    authority = temAuthority;
  }
  function changeAuthority(address [] temAuthority){
    if(msg.sender == owner){
      authority = temAuthority;
    }
```

```
  }
  modifier permission(){
    for(uint i = 0; i < authority.length; i++){
      if(msg.sender == authority[i]){
        _;
        break;}}
  }
 }

contract FreightYardPic is EmbeddedPermission{
 bytes32 [] freightYardPic;
 address [] freightYardExaminer;
 function FreightYardPic() embeddedPermission(addr){
   address [] addr;
   addr.push(/authority address/);
 }
 function setFreightYardPic(bytes32 pic, address
     uploader) permission() {
  freightYardPic.push(pic);
        freightYardExaminer.push(uploader);
 }
 function getFreightYardPic(uint i) constant returns (
     bytes32 , address){
  return (freightYardPic[i], freightYardExaminer[i]);
 }
}
```

Listing 3: Updated *FreightYardPic* code after using the *embedded permission* service.

The SampleTesting smart contract implements the logic in the sample testing activity. According to the process design, the product passes the sample test (i.e. the pass() function in the SampleTesting smart contract is invoked) only when enough number of labs agree that the product passes the sample test. Thus, SampleTesting smart contract code and the required lab addresses are provided as the input for uBaaS and the pass() function is selected to apply *multiple authorities* design pattern using the corresponding service. MutlipleAuthorities smart contract including the modifer isEnoughAgreement() attached to pass() are added after using the uBaaS *multiple authorities* service. Listing 1 presents the updated smart contract code generated by the *multiple authorities* service in uBaaS.

The quality tracing service agreement signed between the product supplier and the quality tracing service agency is implemented in the smart contract ServiceAgreement. In order to ensure the data privacy in the service agreement, uBaaS adds the modifier verfy() to the queryAgreement() function in ServiceAgreement using uBaaS. Only the party who can provide the secret key can successfully read the service agreement information stored on-chain. The smart contract code generated by the uBaaS *dynamic binding* service is presented in Listing 2. Note that the secret key can be changed on demand by invoking function changeKey().

The smart contract FreightYardPic maintains the pictures taken at the freight yard. According to the process design, only the freight yard inspectors can upload the hash value of the pictures taken at the freight yard. Thus, we use the *EmbeddedPermission* service in uBaaS to add the access control for the function setFreightYardPic() which only allows the specific agency employee addresses to store the hash value of the freight yard pictures on blockchain. The smart contract code generated by the *embedded permission* service is presented in Listing 3.

7. Conclusion and future work

In this paper, we present a unified blockchain as a service platform named uBaaS which provides deployment as a service, design pattern as a service and auxiliary services. Deployment as a service in uBaaS is not vendor locked and provides one-click deployment service by hiding deployment script from users. Design pattern as a service leverage design patterns to facilitate data management and smart contract design of blockchain-based applications. We evaluate the feasibility and scalability of uBaaS using a real-world quality tracing use case, which demonstrates it is feasible and scalable to design and deploy blockchain-based applications using uBaaS. The future work includes adding more blockchain platforms as deployment platform options, and designing self-sovereign identity as a service in uBaaS.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Distributed Ledger Technology: beyond blockchain, Technical Report, 2016. UK Government Chief Scientific Adviser.

[2] M. Staples, S. Chen, S. Falamaki, A. Ponomarev, P. Rimba, A.B.T.I. Weber, X. Xu, J. Zhu, Risks and opportunities for systems using blockchain and smart contracts, Technical Report, Sydney, 2017. Data61(CSIRO).

[3] G.P. Release, Gartner survey reveals the scarcity of current blockchain deployments, 2018. accessed: 4 May 2018.

[4] D. Siegel, Understanding the dao attack, 2016.

[5] K. Beck, W. Cunningham, Using pattern languages for object oriented programs, in: Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), ACM, Orlando, FL, USA, 1987.

[6] S. Nakamoto, Bitcoin: A Peer-to-Peer electronic cash system, 2008.

[7] F. Tschorsch, B. Scheuermann, Bitcoin and beyond: A technical survey on decentralized digital currencies, IEEE Commun. Surv. Tutor. 18 (3) (2016) 464.

[8] S. Omohundro, Cryptocurrencies, smart contracts, and artificial intelligence, AI Matters 1 (2) (2014) 19–21.

[9] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, J. Mendling, Untrusted business process monitoring and execution using blockchain, in: BPM, Springer, Rio de Janeiro, Brazil, 2016, pp. 329–347.

[10] X. Li, P. Jiang, T. Chen, X. Luo, Q. Wen, A survey on the security of blockchain systems, Future Gener. Comput. Syst. (2017).

[11] A. Reyna, C. Martn, J. Chen, E. Soler, M. Daz, On blockchain and its integration with iot. challenges and opportunities, Future Gener. Comput. Syst. 88 (2018) 173–190.

[12] Q. Lu, X. Xu, Adaptable blockchain-based systems: A case study for product traceability, IEEE Software 34 (6) (2017) 21–27.

[13] X. Xu, Q. Lu, Y. Liu, L. Zhu, H. Yao, A.V. Vasilakos, Designing blockchain-based applications a case study for imported product traceability, Future Gener. Comput. Syst. 92 (2019) 399–406.

[14] Q. Qu, I. Nurgaliev, M. Muzammal, C.S. Jensen, J. Fan, On spatio-temporal blockchain query processing, Future Gener. Comput. Syst. 98 (2019) 208–218.

[15] G. Zyskind, O. Nathan, A. Pentland, Decentralizing privacy: Using blockchain to protect personal data, in: SPW, 2015.

[16] B. Nasrulin, M. Muzammal, Q. Qu, Chainmob: Mobility analytics on blockchain, in: 19th IEEE International Conference on Mobile Data Management, MDM 2018, Aalborg, Denmark, June 25-28, 2018, 2018, pp. 292–293.

[17] M. Muzammal, Q. Qu, B. Nasrulin, Renovating blockchain with distributed databases: An open source system, Future Gener. Comput. Syst. (2018).

[18] M. Ali, J. Nelson, R. Shea, M.J. Freedman, Blockstack: A global naming and storage system secured by blockchains, in: USENIX ATC, Santa Clara, CA, 2016.

[19] S. Matsumoto, R.M. Reischuk, Ikp: Turning a pki around with decentralized automated incentives, in: IEEE SSP, San Jose, CA, US, 2017.

[20] X. Liang, S. Shetty, D.T. et al., Provchain: A blockchain-based data provenance architecture in cloud enviornment with enhanced privacy and availability, in: CCGrid, 2017.

[21] I. Weber, V. Gramoli, M. Staples, A. Ponomarev, R. Holz, A. Tran, P. Rimba, On availability for blockchain-based systems, in: SRDS'17: IEEE International Symposium on Reliable Distributed Systems, IEEE, Hong Kong, China, 2017, pp. 64–73.

[22] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-oriented Software, Pearson Education, UK, 1995.

[23] J. Eberhardt, S. Tai, On or off the blockchain? insights on off-chaining computation and data, in: ESOCC 2017(6th European Conference on Service-Oriented and Cloud Computing), Springer International Publishing, Oslo, Norway, 2017, pp. 3–15.

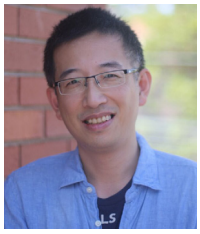
[24] P. Zhang, J. White, D.C. Schmidt, G. Lenz, Applying Software Patterns to Address Interoperability in Blockchain-based Healthcare Apps, 2017.
[25] Y. Liu, Q. Lu, X. Xu, L. Zhu, H. Yao, Applying design patterns in smart contracts, in: International Conference on Blockchain, Springer, 2018, pp. 92–106.
[26] M. Bartoletti, L. Pompianu, An empirical analysis of smart contracts: platforms, applications, and design patterns, ArXiv e-prints (2017).
[27] X. Xu, C. Pautasso, L. Zhu, Q. Lu, I. Weber, A pattern language for blockchain-based applications, in: EuroPLoP'18: European Conference on Pattern Languages of Programs, Kloster Irsee, Germany, 2018.
[28] L. Anderson, R. Holz, A. Ponomarev, P. Rimba, I. Weber, New kids on the block: an analysis of modern blockchains, CoRR abs/1606.06530 (2016).
[29] I. General Administration of Quality Supervision, Q. of the People's Republic of China, Administration of Quality Supervision, Q. of the People's Republic of China, Rules for quality tracing of import commodities, Technical Report, Beijing, 2017.

Dr. Qinghua Lu is a senior research scientist at CSIRO, Australia. She is also a conjoint senior lecturer at University of New South Wales (UNSW). Before she joined CSIRO, she was an associate professor at China University of Petroleum. She formerly worked as a researcher at NICTA (National ICT Australia). She received her Ph.D. from University of New South Wales in 2013. Her research interest includes architecture design of blockchain applications, blockchain as a service, model-driven development of blockchain applications, reliability of cloud computing, and service engineering. She has published more than 70 peer-reviewed academic papers in international journals and conferences. She has served as an editor or reviewer for many journals and as a PC member for a number of international conferences and workshops in the blockchain, cloud computing, big data and software engineering community. Email: Qinghua.Lu@data61.csiro.au

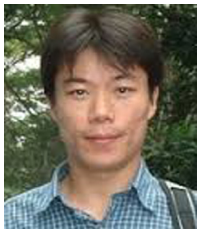
Xiwei Xu is a senior research scientist in Architecture & Analytics Platforms (AAP) team at Data61, CSIRO. She is also a Conjoint Lecturer at University of New South Wales (UNSW). She has a Ph.D. from UNSW. Her main research interest is software architecture. She also does research in the areas of service computing, business process, cloud computing and dependability. She started working on blockchain since 2015. She is doing research on blockchain from software architecture perspective, for example, trade-off analysis, decision making and evaluation framework etc.

Yue Liu is a post-graduate student in the College of Computer and Communication Engineering at China University of Petroleum (East China), Qingdao, China. His research interest includes blockchain as a service and architecture design of blockchain applications. Email: s17070790@s.upc.edu.cn

Dr. Ingo Weber is a Principal Research Scientist & Team Leader of the Architecture & Analytics Platforms (AAP) team at Data61, CSIRO in Sydney. In addition he is a Conjoint Associate Professor at UNSW Australia and an Adjunct Associate Professor at Swinburne University. He has published over 80 refereed papers and two books. His research interests include Blockchain, Systems Operations and DevOps, Business Process Management, Cloud Computing, and Artificial Intelligence/AI Planning. Ingo served as a reviewer for many prestigious journals, including various IEEE and ACM Transactions, and as PC member for WWW, BPM, ICSOC, AAAI, ICAPS, IJCAI, and many other conferences and workshops. Email: Ingo.Weber@data61.csiro.au



Dr./Prof. Liming Zhu is a Research Director at Data61, CSIRO. He is also a conjoint full professor at University of New South Wales (UNSW). He is the chairperson of Standards Australia's blockchain and distributed ledger committee. His research program has more than 200 people innovating in the area of big data platforms, computational science, blockchain, regulation technology, privacy and cybersecurity. He has published more than 150 academic papers on software architecture, secure systems and data analytics infrastructure. Email: Liming.Zhu@data61.csiro.au



Prof. Weishan Zhang is a full professor in the Department of Software Engineering at China University of Petroleum. His research interests are big-data platforms, pervasive cloud computing, and service-oriented computing. Zhang received a Ph.D. in mechanical manufacturing and automation from Northwestern Polytechnical University. Email: zhangws@upc.edu.cn