# T-DSES: A Blockchain-powered Trusted Decentralized Service Eco-System

Xing Wu[1,†], Zhenfeng Gao[2,†], Yushun Fan[1,*], Xiu Li[3,*], Liang Gu[2], Jia Zhang[4], Chang Chen[5], Hao Zhang[5] and Qiang Wang[5]

[1]*Beijing National Research Center for Information Science and Technology (BNRist), Department of Automation,Tsinghua University, Beijing, China*
[2]*Sangfor Technologies Inc., Shenzhen, China*
[3]*Graduated school at shenzhen, Tsinghua University, Shenzhen, China*
[4]*Department of Computer Science, Southern Methodist University, TX, USA*
[5]*Zhigui Technology Inc., Beijing, China*
*E-mail: wuxing17@mails.tsinghua.edu.cn; fanyus@tsinghua.edu.cn; gzf@sangfor.com.cn; guliang@sangfor.com.cn; li.xiu@sz.tsinghua.edu.cn; zhangjia@smu.edu; chenchang@zhigui.com; zhanghao@zhigui.com; wangqiang@zhigui.com*
[*]*Corresponding Author*
[†]*Co-first Authors*

Received 14 November 2020; Accepted 15 July 2021;
Publication 27 October 2021

## Abstract

Existing Web service eco-systems are typically managed in a centralized manner, which hinders their further development due to inherent disadvantages such as trust issues, interest disputes, value separation and so on. The recently emerged blockchains provide distributed ledgers that enable parties who do not fully trust each other to maintain a set of global states, which provide a natural solution. Based on the INKchain, which is an open-source permissioned blockchain mechanism extending the Hyperledger Fabric, this paper proposes Trusted Decentralized Service Eco-System (T-DSES). T-DSES achieves not only fundamental functionalities of conventional systems, but also offers mechanisms to stimulate participants to bring trustworthiness to the whole system. The trustworthiness of T-DSES is realized by three strategies: reliable information of services and mashups, reliable records of participants' rights, and reliable measurement of participants' contributions. A customized token "*SToken*" is created to act as the media of value circulation. In this paper, the overall framework and detailed design of T-DSES are presented, especially including how to utilize Kubernetes to establish a cloud-based environment. A tailored Web front-end ensures the usability of operations. Over real-world data from ProgrammableWeb.com, analyses and experiments have been conducted to verify the feasibility and effectiveness of the presented approach.

**Keywords:** Web service, service eco-system, blockchain, system decentralization, Hyperledger Fabric, INKchain.

## 1 Introduction

With the wide recognition and adoption of Service-Oriented Architecture (SOA) and Cloud Computing, the number of published Web services on the Internet has witnessed a rapid growth [1] in recent years. Service providers interconnect their offerings in unforeseen ways, which gives rise to Web service eco-systems [2]. Furthermore, by reusing existing services (i.e., Application Programming Interfaces or APIs), software developers are able to quickly create service compositions (i.e., mashups) to meet complex functional needs and offer additional business values [3]. A Web service eco-system thus becomes a logical collection of Web services as well as their compositions. For example, ProgrammableWeb.com,[1] consisting of more than 24,000 Web services and 6,400 mashups as of June 2021, represents by far the largest Web service eco-system [4]. Another example is StateOfTheDapps,[2] which has collected over 2,400 decentralized applications (Dapps) on the Ethereum since its inception in 2015 [5]. StateOfTheDappps provides users with the information of the DAPPs, including their introductions, current states, development activities, developers, labels and contract addresses on the Ethereum.

Most existing service eco-systems typically rely on some centralized service registries (e.g., ProgrammableWeb.com) as "middle people" to record

[1]https://www.programmableweb.com

[2]https://www.stateofthedapps.com

and track service behaviors (i.e., service interactions in past activities within the service eco-systems), thus to provide functionalities such as service discovery, ranking and recommendation [6,7]. Such a centralized architecture usually comes with issues like trust and scalability. On the one hand, all records (e.g., information about services, mashups and usage records between them) are stored at and managed by a centralized server. Only if users trust the centralized registry, they will participate in contributing to it (e.g., provide detailed feedback information about services and utilize services in the system to create new mashups). On the other hand, people's active participation contributes to the reliability of the service registry, based on which service discovery, ranking and recommendation could be reached with higher accuracy. In other words, the trustworthiness of the centralized service registry and people's participation exhibit a chicken-and-egg situation.

In reality, however, different developers who register services and mashups in the eco-system may not trust each other. Furthermore, existing centralized registries typically lack effective incentive mechanisms to encourage service providers and users to participate actively. That is to say, the "value" in the centralized service eco-systems is isolated. Most systems only play the role of a centralized platform gathering all kinds of information, but lack mechanisms to realize value circulation according to participants' behaviors. Such a fact would reduce participants' enthusiasm to make contributions continuously. For example, when a developer intends to apply for the access to a specific service, he could "pay" the owner for this service in exchange for the rights. If the owner does not get the value reward, he will have less motivation to provide such popular services in the future. In sum, excessive centralization increasingly becomes the bottleneck and hinders the further development of the service eco-systems nowadays.

In recent years, the blockchain technologies are taking the world by storm, largely thanks to the success of Bitcoin [8]. A blockchain, also called a distributed ledger, is essentially an append-only data structure maintained by a set of nodes that may not fully trust each other. In its original design, Bitcoin's blockchain stores the coins as the system states [9]. Since then, the technology has grown beyond crypto-currencies to support user-defined states. Taking the Ethereum [5] as an example, it provides a platform for people to develop and run any decentralized applications, known as DApps with the help of smart contracts. In such cases, "token," known as "BTC" in a Bitcoin network, or "ETH" in the Ethereum, is the key media to realize value circulation and systematic stimulation in blockchain systems. Recently, increasingly more industry organizations from different domains have made efforts to develop customized blockchain platforms where participants are authenticated. Such systems are called permissioned or consortium blockchains [10–12]. Particularly, the Hyperledger Fabric [13] realizes the identity management mechanism required under enterprise scenarios, and has become a widely-used open-source permissioned blockchain platform aimed at business uses [14]. In general, a blockchain-based system would create a trusted environment between participants who may not trust each other. It could also prevent recorded information from illegal modification by malicious attackers unless they could control most resources in the blockchain network.

Therefore, to address the aforementioned issues of conventional service eco-systems, in this paper, we migrate the idea of blockchain into service eco-system and introduce the framework of Trusted Decentralized Service Eco-System (T-DSES). Generally, T-DSES refers to a service eco-system gathering and providing trusted information of services and mashups along with value circulation through customized token, which is powered by permissioned blockchain technology and the endorsement of authoritative organizations. T-DSES keeps the records of participants' equities (or rights) about having the access to using a specific service, which brings trustworthiness to the overall eco-system. In sum, the trustworthiness of T-DSES is mainly reflected from the following three aspects: reliable information covering all aspects of services and mashups; reliable records of participants' rights of using specific services in the system; reliable measurement of participants' contributions to the system.

In terms of the selection of the underlying blockchain architecture, in order not to reinvent the wheel, T-DSES leverages the INKchain[3] for two major reasons. First, identification and authority controls are the unique features of permissioned blockchains, which are helpful to allow qualified organizations to participate in maintaining T-DSES, thus to promote the quality and reliability of the information stored in the system. Second, the systematic characteristic of the asset component in the INKchain makes it possible to issue our customized token "*SToken*" as the media of value circulation and realize the incentive mechanisms in T-DSES. To be more specific, function blocks are implemented as smart contracts (i.e., chaincodes) on a supervisor level. Whenever an activity occurs in T-DSES, all involved parties will individually create a detailed record through the designed interfaces in the chaincode. Illegal operations will be denied and the agreed-upon records

[3]https://github.com/inklabsfoundation/inkchain

will be permanently stored and maintained at the local database of each involved node in the network. Such a distributed-database-oriented solution will enable services who do not fully trust each other to maintain a set of global states. In this way, service discovery and recommendation with higher scalability and maintainability can be realized.

To the best of our knowledge, this paper is the first attempt that designs and implements a cloud-based "trusted" decentralized service platform. The contributions of this paper are summarized in three-fold:

(1) A new concept of "Trusted Decentralized Service Eco-System" (T-DSES) is created, as the first attempt to utilize the blockchain mechanism to bring trustworthiness and address the problems encountered by conventional centralized service eco-systems.
(2) As a proof of concept, the overall framework of T-DSES is implemented based on the INKchain. We present how to build a cloud-based service eco-system, including the design of the overall logic, details of the chaincode and the method to establish and manage a cloud-based network environment. A tailored Web front-end is also provided. The code of T-DSES has been open sourced in the GitHub repository.[4]
(3) Over the real-world data from ProgrammableWeb.com, case studies and tests are conducted to prove the design of T-DSES focusing on its functionality, performance and robustness.

The rest of this paper is organized as follows: Section 2 introduces background and motivation. Detailed design of T-DSES is provided in Section 3. In Section 4, the experimental results based on a cloud-based T-DSES environment are presented. Section 5 discusses related work and finally Section 6 concludes the paper.

## 2 Background and Motivation

### 2.1 Service Eco-System

Service eco-systems are logical collections of Web services (APIs) as well as their compositions (mashups) [2]. A typical architecture around a service eco-system is shown as Figure 1, with the legend on the right.

Generally, a typical service eco-system is a platform that gathers information of all kinds of services and mashups. There exist three kinds of major roles in service eco-system: **Developer**, **User**, and **System Administrator**.

[4]https://github.com/wuxing610/T-DSES

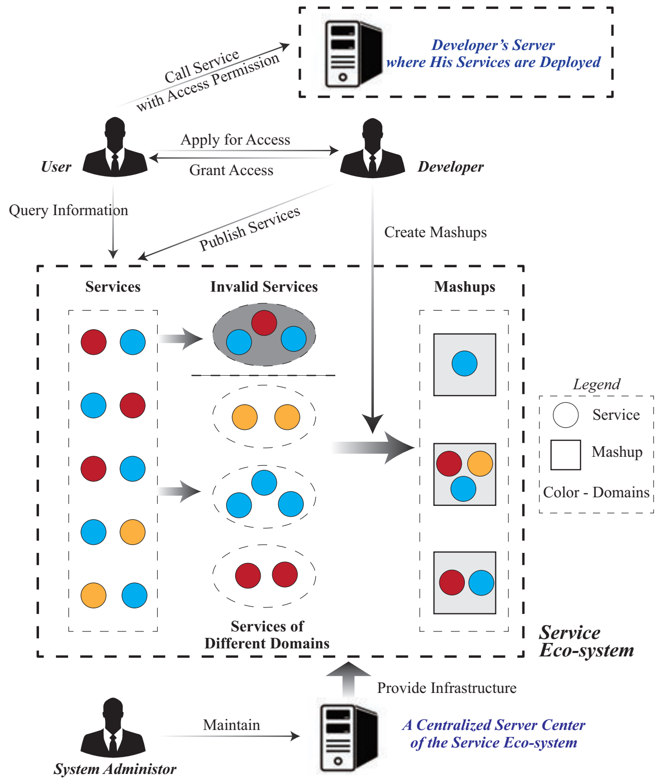


**Figure 1** Framework of conventional centralized service eco-system.

**Developers** (i.e., service providers) register and publish their original software services (i.e., APIs), which are the basic components of the service eco-system. Various services may provide different functions and work in different application scenarios, which are reflected by different colors in Figure 1. With the evolution of the service eco-system, some services may become invalid, while others gradually form different application domains according to their functions. To meet complex functional needs and offer additional business values in a faster development cycle, developers reuse existing services to create mashups (i.e., service compositions). A mashup may invoke one or more services to implement specific function. The second role of a service eco-system is **User**. Users may propose inquiries to search for services or mashups in the eco-system and then directly utilize them under different circumstances. In a centralized service eco-system, the third role, **System Administrators** also play an important role in maintaining the stability, security and prosperity of the service eco-system. Without their efforts, the service eco-system could become more vulnerable, inconspicuous and inactive.

The bottom of Figure 1 illustrates that existing service eco-systems are usually deployed on centralized servers maintained by system administrators. The service eco-system only collects information of services and mashups, while the real available services are usually deployed on the developers' own servers. A user could query information in the service eco-system and select a specific service. However, if he intends to call the service in real scenarios, the user has to submit an application to this service's developer for access to call the service. After being authorized, the user can communicate with the developer's server and call the service. In other words, these behaviors are performed outside the service eco-system. That is to say, most existing service eco-systems only present related information, but are isolated from the real value circulation between the participants.

To be more specific, let us take ProgrammableWeb.com as an example. Since its inception in 2005, ProgrammableWeb.com has aggregated more than 24,000 Web services and 6,400 mashups up to June 2021. It is a centralized service eco-system maintained by the official team of the platform. ProgrammableWeb.com, as the official journal and directory of the API Economy, aims at building a news and information source as well as a community of all kinds of API economy stakeholders. Its operating team has established the website to provide a portal to people to publish and search for various services and mashups. The real available URLs are provided in the profile of each service and mashup. For example, a user will find service ***Google Map*** in ProgrammableWeb.com, and intends to call the service when he needs to query geographical locations. He might need to pay Google for the access, though. These relevant behaviors are outside of the ProgrammableWeb.com.

## 2.2 Blockchain Technology

A typical blockchain system usually consists of multiple nodes which do not fully trust each other [9]. Together, the nodes maintain a set of shared, global states and perform transactions modifying these states. Figure 2 shows the typical structure of a blockchain, where each block is linked to its predecessor via a cryptographic pointer, all the way to the first (i.e., genesis) block. Due to this structure, blockchain is also called distributed ledger.

A specific transaction in a blockchain is a sequence of operations applied to global states. The core difference is that blockchain is decentralized,

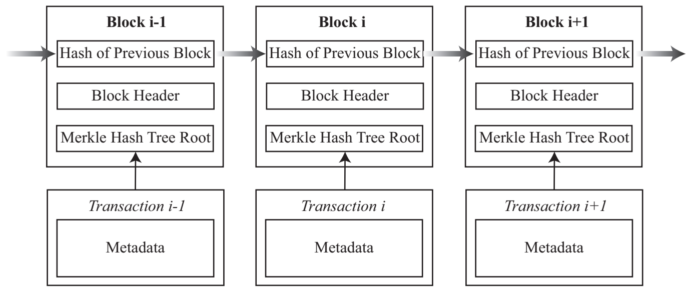


**Figure 2** Typical blockchain structure.

and each node has its own ledger while having to reach a consensus in the whole system. Blockchain systems have advantages in decentralization, security, anonymity and untamperability. Avoiding the trust issues of centralized systems, blockchain has enabled a new class of decentralized applications [18].

Blockchain usually makes a trade-off between performance and trust. Generally, blockchain systems can be categorized as *public*, *private* or *permissioned* ones, which are described in details as follows.

**(1) Public Blockchain:** Public blockchains are maintained across peer-to-peer networks in a totally decentralized and anonymous manner [5, 8]. Anyone can join the network freely. Bitcoin [8] is the most well-known example of public blockchains. In order to determine which block to be appended to the ledger next, peers have to execute Proof-of-Work (PoW) consensus [19]. Ethereum [5] has grown beyond crypto-currencies to support user-defined states and Turing Complete state machine models. It enables any decentralized applications (DApps) with the help of smart contracts. Public blockchains have advantages of enabling the ledger to be created anonymously, and promoting peers' willingness to hold a copy of the ledger and try to create new blocks.

**(2) Private Blockchain:** In private blockchains, only specific organizations are allowed to join the networks. They are widely used in building blockchain-based underlying systems among multiple specific organizations. For example, two banks might negotiate to establish a private blockchain between them to assist with the reconciliation.

**(3) Permissioned Blockchain:** The definition of permissioned blockchains falls in between public blockchains and private ones. Only authorized organizations can join in permissioned systems. Permissioned blockchains require a set of trusted nodes tasked with creating new blocks. Moreover, permissioned blockchains make it possible to control the set of participants tasked with maintaining the ledger. This feature increases its popularity among industrial communities. Hyperledger Fabric is an open-source project under the Hyperledger umbrella project,[5] which targets at business applications [20, 21]. In real application scenarios of permissioned blockchain, the maintenance tasks for nodes in the network can be assigned to authorized organizations according to the business logic.

## 2.3 Motivation to Bring Blockchain

ProgrammableWeb.com has successfully established a community of all kinds of API economy stakeholders. Developers publish their new services or mashups on it, and users could find out information related to these services or mashups. Moreover, ProgrammableWeb.com updates news about APIs frequently, and sets up an “API University” to provide valuable information and guidance for users and developers.

However, there exist disadvantages due to the centralization of the system.

**(1) Trust Issues:** Excessive centralization could result in trust problems. In a centralized service registry, services provided by different developers may not trust each other. A developer would play an active part in the service eco-system (i.e., provides accurate and comprehensive information about services, and utilizes services to create mashups frequently) only if he trusts the authority of the centralized service registry. For centralized system, the security of user’s privacy and the authenticity of the data are guaranteed by the reputation of the platform. However, this guarantee is limited because the platform may disclose the user’s data for its own benefit. There have been some actual cases of personal data leakage and abuse. US cell carriers (including AT&T, T-Mobile and Sprint) soled access to customers’ real-time phone location data to a little known company called Securus.[6] The personal data of 533 million Facebook users was leaked to the Internet, including phone numbers, email addresses, hometowns, full names and birth dates.[7] Not all centralized platforms have experienced trust crisis, but they do face the risk of losing people’s trust. Without a service eco-system containing credible information about all kinds of behaviors, it will be difficult to realize service

[5]https://www.hyperledger.org

[6]https://www.zdnet.com/article/us-cell-carriers-selling-access-to-real-time-location-data/

[7]https://www.technologyreview.com/2021/04/07/1021892/facebook-data-leak/

management, discovery and recommendation with scalability and maintainability. Furthermore, because users may not utilize a distrustful platform to search information of services, the process of applying for services' access between users and developers may not happen in the service eco-system.

**(2) Security Problems:** Centralized service platforms are usually susceptible to attack, making the massive data of services easy to tamper with. That is to say, in a centralized system, if the main servers are invaded, the information of all the services and mashups may get falsified, which will bring massive economic losses.

**(3) Privacy Control:** In a centralized system, it is usually difficult to realize fine-grained authorization control on a systematic level, especially when the system is not fully trusted by all the participants.

**(4) Lack of Incentive Mechanisms:** In existing conventional service eco-systems, developers publish their services and mashups voluntarily. There are no explicit incentive mechanisms in the existing systems. As a result, on the one hand, some service providers (i.e., developers) might have no motivation to develop new services or mashups actively. On the other hand, the developers could not get any feedback from the system about the quality of their published services and mashups, which will have negative impacts on the stability and prosperity of the service eco-system.

**(5) High Cost of Maintenance:** The founder of Ethereum, Vitalik Buterin, said in Ethereum Industry Support that the centralized platforms usually involved high cost.[8] Centralized platforms usually need administrator roles, as well as high infrastructure and human cost. [15] and [16] refer to the high cost in centralized banking and electricity energy system. Offering a community-oriented service platform that many community members rely on will cost billions of dollars in innovation research every year [17].

**(6) Separation of Value:** As shown in Figure 1 and discussed in Section 2.1, most existing centralized service eco-systems only gather related information, but they are isolated from the real value transfer between participants. Here, the term "**value**" refers to a developer/user's equities in a service eco-system. On the one hand, when he makes contribution to the system, he could get equity reward from the system or other developers. On the other hand, his equities could be transferred when he asks for resources in the system (e.g., intends to call an service). In short, the value flows between

[8]https://www.youtube.com/watch?v=mf10IXK-WM0

different participants to maintain the smooth and healthy operation of the service eco-system.

Blockchain is an integrated system, including many outstanding technologies, such as decentralization, cryptography, consensus, and so on. Blockchain could protect the data from being tampered with by using chain storage structure, support identification and authority controls, ensure the automatic execution of transaction flow without the interference of any third-party with the help of well-designed smart contracts, realize the value transmission through tokens. And blockchain has the potential to solve trust issues. As for one of the centralization problems, e.g. privacy control, we have many alternatives to handle with. However, the objective of our work is more than providing privacy control, but establishing a trusted service eco-system. Therefore, we incorporate blockchain technology to build an integrated system.

## 2.4 Bring Trustworthiness in Three Aspects

Among the problems mentioned above, the trust problem is the most fundamental one. Only if the service eco-system is trusted by participants, could the prosperity and value circulation of the system be guaranteed.

Generally speaking, the trustworthiness of information in an ideal service eco-system is mainly reflected from the following three aspects.

**(1) Information of Services and Mashups:** First, a trusted service eco-system should provide reliable information of services and mashups, which will have a positive effect on service management, discovery and recommendation. Based on it, participants will take a more active part in making contributions to the service eco-system.

**(2) Records of Participants' Rights:** Second, reliable records of participants' rights of using specific services should also be provided by the system. Users can pay a service's developer for the access permission. Then he can call the service deployed on the developer's server. After receiving the user's access request, the server checks the user's rights (e.g., how many invocation times left) and makes response. Note that the developer/server can make changes to user's rights according to the invocation behavior.

**(3) Evaluation of Participants' Contribution:** Last but not least, related participants' contributions to the system should be provided in a trusted way, which records the credibility degree of different participants. If a user makes more contributions to the service eco-system (i.e., publishes popular services or mashups), services developed by him should become more valuable and reliable.

## 2.5 Blockchain Selection in T-DSES

There are two technical routes to deploy underlying blockchain architecture: Ethereum and Hyperledger Fabric. Taking into account the needs of service eco-system, we make a detailed comparison of Ethereum and Hyperledger Fabric, and finally choose Hyperledger Fabric as our technical solution. The main reasons are as follows.

(1) Compared with Ethereum, Hyperledger Fabric requires less sources and is able to reach smaller transaction latency and higher throughput, since resource-consuming consensus algorithms are not needed.

(2) In T-DSES, complicated smart contracts are designed to support the complex transaction logic, and smart contracts need to be called frequently to record a lot of information, such as the published service information. In Ethereum, the execution of smart contracts requires Gas, which increases the cost of transactions and restricts the function of smart contracts. These worries don't occur to Hyperledger Fabric.

(3) Hyperledger Fabric supports identification and authority controls. With Hyperledger Fabric, it is possible to allow only qualified developers or users to participate in T-DSES and avoid hostile behaviors. In realistic scenarios, the maintenance tasks for nodes in the network can be assigned to specific authorized organizations. As a result, T-DSES could focus more on information maintenance instead of worrying about malicious behaviors.

(4) In Ethereum, "*SToken*" can be generated only in Genesis block or by mining. But in Hyperledger Fabric, it is a concept defined by smart contracts, where we could design the "*SToken*" related rules (like how users obtain "*SToken*") flexibly as service eco-system requires.

Having choosing Hyperledger Fabric as the underlying blockchain, we further choose the INK consortium blockchain (INKchain), which extends the Hyperledger Fabric by providing asset accounts, as well as supporting asset transfer and other enhancements.

The systematic characteristic of asset component in INKchain makes it possible to issue customized token "*SToken*" in T-DSES as the media of value transfer. Furthermore, business logic can be coded in the chaincode. With systematic interfaces provided by INKchain, incentive mechanisms and the circulation of "value" in T-DSES can be realized.

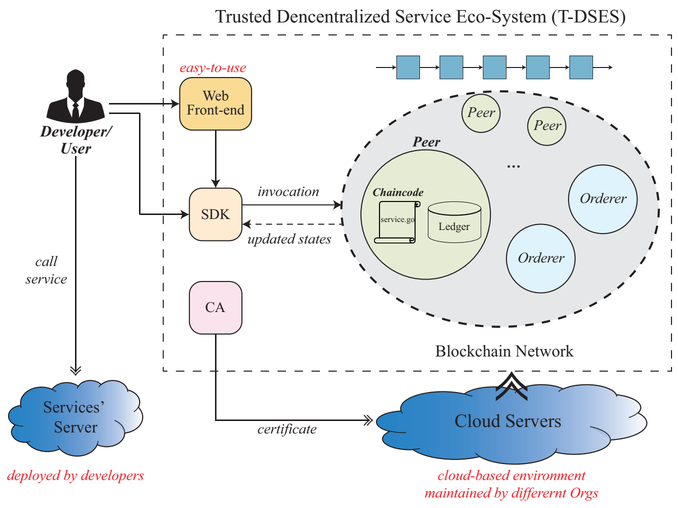


**Figure 3** Overall framework of T-DSES.

Based on its unique and expected features, we select INKchain as the underlying technique to build the blockchain network and implement business logic of service eco-systems on top of it.

## 3 Trusted Decentralized Service Eco-System (T-DSES)

### 3.1 Overall Framework

Figure 3 depicts the overall framework of T-DSES in a cloud environment, which realizes trustworthiness and the value circulation among different participants who may not fully trust each other.

Simply speaking, in this T-DSES-centered architecture, **Developer/User** is involved in the related operations of the system, and the key component of this cloud-based service eco-system is maintained by several **Organizations**. A **Developer/User** can utilize its Software Development Kit (**SDK**) to make invocation to update or query states through chaincodes. Meanwhile, a **Web Front-end** is provided, making a user/developer interact with T-DSES more conveniently and user-friendly.

In actual application scenarios, after reaching agreements on the business logic between entities in the same blockchain network, it is possible to code the business logic into chaincodes to make sure that everyone obeys it. **Peers** and **Orderers** form the Blockchain Network. The chaincode is deployed at each peer in the blockchain network, which is open and transparent to everyone. Deployed chaincode defines interfaces for participants to interact with the network, realizing corresponding business logic and incentive mechanisms in the service eco-system. Any illegal operations (e.g., not defined in chaincodes or beyond the scope of authority) will have no effect on the global states. Such a distributed database-oriented solution will enable peers who do not fully trust each other to maintain a set of global states. All the peers will hold the full history of all activities since the service eco-system is established, assuring the reliability and authority of the information. T-DSES will thus provide reliable and sound information for researches on service management, discovery and recommendation that promises high scalability and maintainability. Furthermore, T-DSES issues the basic token ***"SToken"*** to realize trusted circulation of value, realizing incentive mechanisms to encourage participants to make more contributions to the eco-system.

Detailed explanations of the main components are discussed as follows.

**Servers:** As shown in Figure 3, to maintain normal operations of T-DSES's business logic, it is necessary to deploy three kinds of servers. First, after getting authorization from Certificate Authority (CA), the organizations maintain the servers where the blockchain network of T-DSES are running. Second, the Web front-end needs a server, too. It could be provided either by a third party, or the users themselves. It is convenient for users/developers to interact with the T-DSES through an easy-to-use front-end instead of SDK. Third, since the developers only publish the information of services or mashups in T-DSES, servers are required to deploy these services and mashups to provide service and make responses to users' invocation requests.

**Developer/User:** In the business logic of T-DSES, there are only two types of participating roles (i.e., Developer and User). Developers/users could make several kinds of invocations (e.g., publish a new service, query service information, or create a new mashup), and broadcast them into the T-DSES blockchain network through SDK directly or using the Web front-end.

**"*SToken*":** The concept of "*SToken*" borrows from "BTC" in Bitcoin and "ETH" in Ethereum. However, "*SToken*" are not mined in T-DSES, since there are no miners in Hyperledger Fabric. As long as user's behaviors meet the setting of the chaincode, a certain amount of "*SToken*" would be created and added to user's account. Generally, users can obtain "*SToken*" from the following two ways: First, when users register in T-DSES, they could get a certain amount of "*SToken*" as initial assets; Second, since one of the motivations of designing "*SToken*" in our T-DSES is to promote the development of service eco-system, users can get "*SToken*" by actively participating in T-DSES (e.g. make comments to the service, contribute to the service eco-system, etc.). And for developers, they could get "*SToken*" by getting paid and awarded by users after publishing services or mashups. As typical setting of blockchain-powerd systems, we could set a large number as total amount. And as time goes by, the "*SToken*" that users can obtain from participating in T-DSES decrease. Also as typical setting, we could give out a certain proportion of "*SToken*" to the original contributor and the node maintaining organizations of T-DSES as initial distribution. The other "*SToken*" would be created during the operation of T-DSES.

**Web front-end:** The Web front-end component provides a more convenient way for developers/users to interact with the blockchain network. As shown in Figure 3, essentially, Web front-end also calls the SDK component to interact with the network part in T-DSES. While in a cloud environment, it is usually necessary to contain a Web front-end. The codes of the Web front-end are open source. The server could be provided by an organization, or any participant in the T-DSES. It should provide conventional functionalities of centralized service eco-systems (e.g., publish a new service in ProgrammableWeb.com), as well as operations about value circulation (e.g., pay a service's developer certain "*SToken*" to get the invocation access to the service). The operations in the Web front-end are similar with those in centralized service repositories, bringing no burden for developers/users. Detailed design will be introduced in the next section.

**SDK:** The SDK component implements the business logic of the service eco-system through provided interfaces defined in the chaincode. SDK links the blockchain network in T-DSES and the outside. This component could run on any blockchain node, or be deployed on a centralized server. In T-DSES, it is deployed on the servers maintained by organizations. Developers/users can directly interact with it, and operations that affect the states will generate a transaction through related interfaces. INKchain provides a matched SDK project called INK SDK.[9] INK SDK also provides means to interact with INKchain freely, operate customized tokens and invoke INK smart contracts by large amount of anonymous users using INK Accounts.

**CA:** The CA component could provide legal identity certifications to authorize qualified organizations to join the maintenance of the blockchain network in T-DSES.

[9]https://github.com/inklabsfoundation/inkchain-sdk

**T-DSES Blockchain Network:** The right side of Figure 3 illustrates the structure of T-DSES blockchain network, which is the core component of T-DSES. T-DSES blockchain network is a distributed management system of transaction records in the scheme. All the developers and users contribute blockchain operations. Similar to the Hyperledger Fabric, blockchain network based on INKchain distinguishes between two kinds of nodes: **Peers** and **Orderers** [14]. Peer is a kind of non-validating node that functions as a proxy to connect clients (issuing transactions) to validating nodes (orderers). It does not make real executions of transactions but may execute simulations and verify them. Chaincode defines basic business logic and provides interfaces for invocation, intending to record data in a distributed ledger. The data to be recorded is called "state" and stored in a key-value form. An orderer is a node on the network responsible for running consensus, validating transactions, and maintaining the ledger. In practical application scenarios, the orderers and peers could be run and maintained by different organizations. No central server managed by a third party is required. Thus there is no need for **System Administrator** role to provide maintenance service. Furthermore, in this paper, we utilize Kubernetes to build a blockchain environment on cloud servers instead of on a local PC in our previous work [22]. In sum, T-DSES is built according to the needs of industrial and real application scenarios, which can meet different participants' requirements.

## 3.2 Behaviors between Participants

In order to illustrate behaviors between different participants, we summarize two typical scenarios in T-DSES: (1) A user intends to call a specific service; And (2) a developer utilizes an existing service to publish a new mashup.

### 3.2.1 User and developer

As shown in Figure 4, the first scenario is that user A intends to call service *i* which is provided by developer B. In this case, related behavior processes are listed below.

1. Developer B creates service *i*, and deploys it on his server to provide service to others.
2. Developer B registers service *i* in T-DSES. That is to say, Developer B publishes service *i*'s information into T-DSES through the Web front-end (or SDK).
3. User A finds service *i* in T-DSES, which exactly meets his needs. item[4.] User A applies for *i*'s access. He has to pay with "*SToken*"

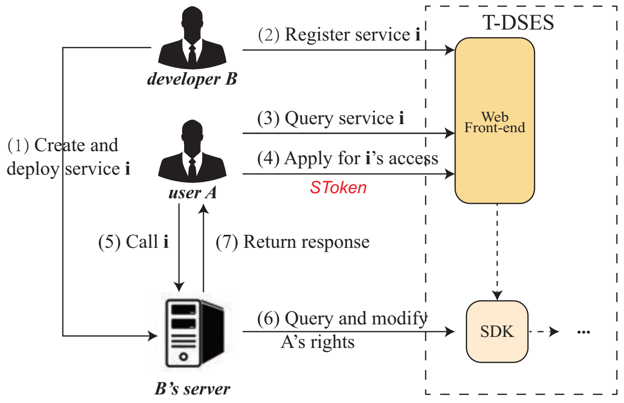

developer B
user A
B's server
T-DSES
Web Front-end
SDK
(1) Create and deploy service **i**
(2) Register service **i**
(3) Query service **i**
(4) Apply for **i**'s access
SToken
(5) Call **i**
(7) Return response
(6) Query and modify A's rights
...

**Figure 4** Behaviors between user and developer.

in order to acquire the rights to call service *i*. After verifying the transformation, A's rights on service *i* will be recorded in the global ledger.

5. User A makes an invocation request of service *i* to the B's server.
6. After receiving the request, the server checks A's rights through interfaces provided by SDK. If A is qualified, before sending the response, the server makes a transaction proposal to modify the User's rights by reducing one call-time.
7. After confirming A is authorized, the server returns response results to User A according to his invocation parameters.

In Step 4, after the transformation is successfully executed, A's rights to call service *i* is recorded in the blockchain network and can be verified by any participant in T-DSES. In T-DSES, if user A thinks service *i* is of great help to him, he can also reward developer B by transferring "*SToken*" with no strings attached.

### 3.2.2 Mashup developer and service developer

As shown in Figure 5, by searching information of services in T-DSES, developer C finds that service *i* is helpful to realize certain function more easily and credibly when creating mashups. So the second scenario is that developer C intends to create a new mashup *j* which invokes service *i* developed by B. Related behavior processes are listed below.

1. Developer B registers service *i* into T-DSES.
2. By searching information in T-DSES, developer C finds service *i* helpful for him to realize complicated functions.

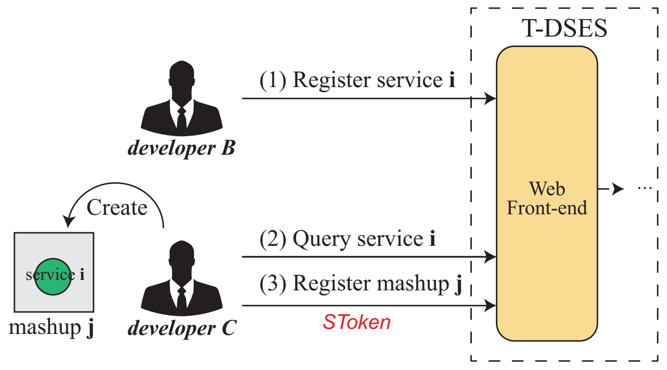


**Figure 5** Behaviors between mashup developer and service developer.

3. Developer C creates a new mashup *j* which invokes *i*. Then he tries to register mashup *j*. C has to transfer "*SToken*" to apply for the right of invoking *i* when making a new mashup. After the successful transformation of "*SToken*", C is able to publish his new mashup *j* into T-DSES.

Through the above processes, a new mashup will be added into the service eco-system. Its meta information will be recorded in the distributed ledger successfully, which is not easy to tamper with.

## 3.3 Blockchain Perspective

From a blockchain perspective, we scrutinize a transaction process. After being authorized, the user can send a transaction proposal to a peer through SDK or Web front-end. The peer checks the proposal, simulates to conduct the transaction, and endorses the results. Then the endorsed proposal will be broadcast to orderers. The orderers validate the transactions and create new legal blocks. At last, the peers update their local ledgers according to the newly-received blocks. Any operation beyond the interfaces declared in chaincode or the authority setting is considered illegal, which will have no effect on the global states. In general, the peers and orderers work together to reach a global consensus and maintain the distributed ledger together.

## 3.4 Security Requirements

T-DSES blockchain network is a decentralized system. Due to its industrial application scenario and derivation from the Hyperledger Fabric, T-DSES supports crash tolerance through an ordering service based on Apache Kafka and Zookeeper to reach a consensus.

INKchain substantially integrates current technological advancement in the fields of distributed computing and security. It takes advantage of cryptographic primitives such as Hash Function [23], Asymmetric Encryption [24] and Digital Signature [25].

## 3.5 Authority Control

Public blockchain platforms usually suffer from lacking of permission control, and the information is completely exposed to the public. Permissioned blockchain, equipped with methods to realize authority control, is designed for industrial scenarios. Based on the INKchain, T-DSES realizes it in three aspects. First, the CA component implements the PKI service that can issue identify certification in advance, and distributes it to corresponding entities. Second, the INKchain could control different entities' access level of data and resources through fine-grained policy control. Last but not least, with the help of INK Account, we design chaincode *"service.go"* to implement data access control functionally such as recording participants' rights on calling services.

## 3.6 Cloud-based environment with Kubernetes

Security of cloud-based applications is one of the key concerns. The three principles of cloud security are availability, confidentiality and integrity. One of the most efficient ways is to deploy a Container Cluster by using Docker for container packaging with Kubernetes for multi-host Docker container management [28].

The deployment of Fabric or INKchain blockchain network is based on Docker Swarms. All peers, orderers and client programs are containerized. The peers join a consensus protocol of the blockchain. In our previous work [22], we have implemented a prototype in an experimental environment on a local PC. One step further, in this paper, we establish a cloud-based blockchain network environment based on containers which are managed by Kubernetes.

In this section, we firstly introduce the Kubernetes. Then the principle of deploying the cloud-based blockchain environment of T-DSES is illustrated from aspects of Service and Namespace. Detailed parameters will be listed in the experimental part.

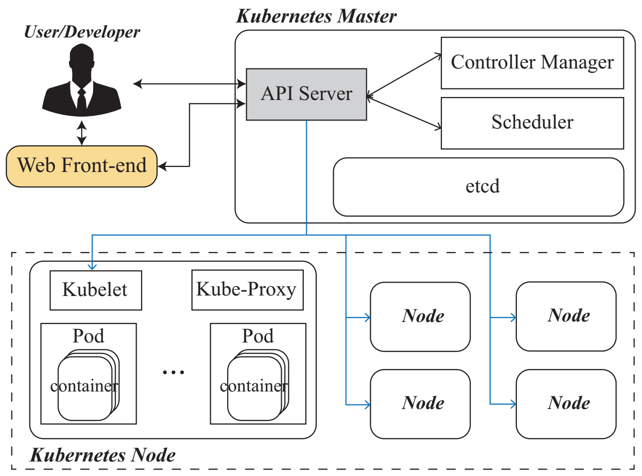


**Figure 6** Architecture of Kubernetes.

### 3.6.1 Scheme overview of kubernetes

Kubernetes (commonly stylized as k8s) [29] is an open-source container orchestration system for automating application deployment, scaling, and management across a cluster of hosts.

The architecture of Kubernetes follows the master-slave architecture [29, 31, 32]. Basically, there are two kinds of nodes (i.e., Master and Node) as shown in Figure 6. Each node may correspond to either a local physical machine or a cloud virtual machine. The number of nodes can be elastically stretched according to the requirement of the business system.

**Master:** The Master is the main controlling unit of the whole cluster, which manages the communication in the system. To support high-availability, in T-DSES, we deploy a three-node cluster of Master. The main components of Master contain API server, Controller Manager, Scheduler and etcd. API Server is a key component and serves Kubernetes API using JSON over HTTP, providing both the internal and external interfaces [31, 33]. Control Manager drives actual cluster state toward the desired cluster state [34]. Scheduler tracks the condition of resource usage on each node to ensure that workload is not scheduled in excess of available resources. Etcd [35] is a persistent, distributed, key-value data store which reliably stores the configuration data of the cluster.

**Node:** One Node, which runs a Docker environment basically, is a machine where containers (workloads) are actually deployed. Each Node mainly consists of Kubelet, Kube-proxy and Pods. Kubelet is responsible for the running state of each node, ensuring that all containers on the node are healthy. The Kube-proxy is an implementation of a network proxy and a load balancer, supporting service abstraction based on Pods. Pod is the basic scheduling unit in Kubernetes. Each Pod provides a container environment, which is the lowest level of a micro-service that holds the running application, libraries, and their dependencies.

In sum, the architectural style of Kubernetes brings various benefits such as maintainability and flexibility in scaling and aims at decreasing downtime in case of failure or upgrade [31]. These benefits will make a contribution to a high-available blockchain network environment for T-DSES.

#### 3.6.2 Services in Kubernetes

In Kubernetes, Service is a logic set of Pods that works together, making up a multi-tier application. This set of Pods is defined by a label selector, which constitutes a service with certain functionality to the outside. It is sometimes called a micro-service.

To be more specific, in T-DSES, services in Kubernetes are defined as follows.

**CA Service:** CA (Certification Authorization) is the default component to manage certifications in T-DSES. It provides legal PKI-based identity certifications to authorize organizations or users to make operations that will affect the blockchain network.

**Peer Service:** Peer Service maintains the ledgers and runs chaincode containers to make read-write operations on ledgers.

**Orderer Service:** Orderer Service is able to rank all the transactions in the network, create new blocks according to predefined configurations (i.e., fixed number of transactions or fixed separation of time), and then broadcast them.

**Kafka Service & Zookeeper Service:** Kafka and Zookeeper are both projects provided by Apache Software Foundation. Kafka aims to build a unified, high-throughput, low-latency distributed platform for handling real-time data feeds [39, 40]. Zookeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services [41]. Here in the cloud-based environment of T-DSES, Kafka Service and Zookeeper Service usually work together as a message-oriented middle-ware to generate message queue and help Orderer Service to reach the global consensus.

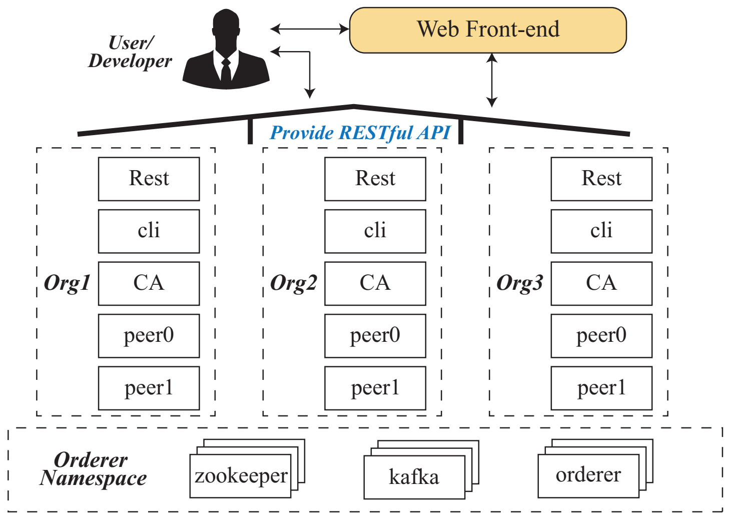


**Figure 7** Internal logic in cloud-based environment of T-DSES.

**Cli Service:** Cli (Command Line Interface) provides an environment with tools like peer, configtxlator, cryptogen and configtxgen, which makes it more convenient for peer-administrators to check status and debug with the blockchain network.

**Rest Service:** SDK component of T-DSES is deployed in Rest Service, providing interfaces for the outside to interact with the blockchain network.

### 3.6.3 Internal logic organized by Namespace

Kubernetes utilize Namespaces to partition the resources in the system into non-overlapping sets. Namespaces help people to build environments with many users from multiple organizations.

As shown in Figure 7, in T-DSES, Namespaces are adopted to manage resources among different organizations. To be more specific, in the cloud-level environment of T-DSES, there is a corresponding relationship between namespaces and organizations except the orderer namespace. Each organization deploys its Cli Service, CA Service, Rest Service and several Peer Services. In the Orderer Namespace, Orderer Service, Kafka Service and Zookeeper Service are deployed as clusters to provide high availability and reach the overall consensus in T-DSES. Here the clusters could be maintained by several authoritative and recognized companies or organizations to bring more trustworthiness into the system.

Generally, the T-DSES environment is of high availability (HA) due to four reasons:

(1) The internal mechanisms of Kubernetes. For example, the Scheduler in Master Node tracks the condition of resource usage on each node to ensure that workload is not scheduled in excess of available resources. And each service in Kubernetes usually contains redundant Pods to ensure the high availability;
(2) Cluster deployment of Peer Services in each organization (or namespace);
(3) Cluster deployment of Orderer Services in the Orderer Namespace;
(4) The internal mechanisms of permissioned blockchain. The network has to reach an overall consensus to change the global states.

## 3.7 Design of Chaincode

In permissioned blockchain architectures such as Fabric and INKchain, "chaincode" is the implementation of smart contract, spearheaded by Ethereum [5]. After the investigation on the functionalities of service eco-systems, in T-DSES, a new chaincode *"service.go"* is developed to implement related business logic. Moreover, incentive mechanisms have been provided to stimulate participants to make more contributions to the service eco-system.

### 3.7.1 Data structure

The data structure of T-DSES is designed as shown in Figure 8 with the following core elements.

**INK Account:** INKchain has designed and implemented an account system called "INK Account", which can cater to a large number of anonymous users to manage digital assets and interact directly with the blockchain. As shown in Figure 8, the **Address** field uniquely identifies a record of INK Account. The **Balance** field records all kinds of tokens in an account. **Counter** is used for validation. In T-DSES, with the help of INK Account, "*SToken*" are issued as the basic media to realize value circulation and incentive mechanisms.

**User:** In the design of data structure, we do not distinguish developers from users and store their information with User structure non-distinctively. The **Name** field is the keyword. A user also needs to provide his brief introduction. Each user is the one-to-one correspondence of an INK Account through the

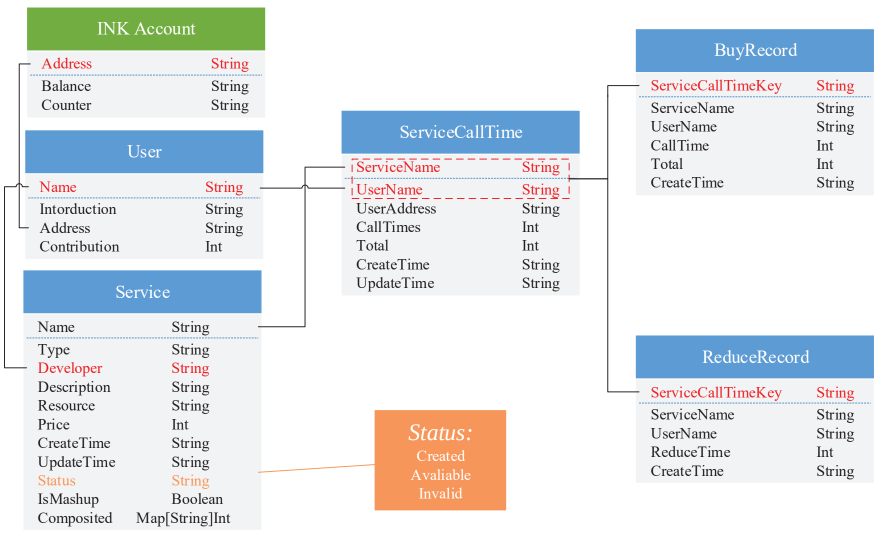


**Figure 8** Design of Data Structure in *service.go*.

**Address** field. The **Contribution** field is designed for recording the degree of a user's contribution to the service eco-system, which will be illustrated later.

**Service:** Structurally, we design the Service to record information of services as well as mashups. The **Name** field is the keyword of Service structure. There exists a one-to-one correspondence between a service and a specific user's name through the **Developer** field. We utilize a boolean field **IsMashup** to distinguish whether the service is a mashup or not. If it is a mashup, the **Composited** field would record the services (i.e., APIs) invoked by the mashup. Specifically, a string field **Status** is adopt to record the current status of a service. Furthermore, the field **Price** defines how many "*SToken*" it costs to request permission for one-time call (i.e., invoke the service in a new mashup, or call the service's server once). Price is provided by the developer of this service or mashup.

**ServiceCallTime:** As explained earlier, if a user intends to call the servers where the service is deployed, he has to transfer a certain amount of "*SToken*" to the developer to get the permission. In *service.go*, ServiceCallTime records the rights of users' accesses to various services. ServiceName and UserName constitute the composite key of ServiceCallTime. The **CallTimes** field records the number of times that the user could call the service. **Total**

field represents for the total amount of "*SToken*" the user has paid, which equals service's Price multiplies CallTimes.

**BuyRecord & ReduceRecord:** BuyRecord and ReduceRecord are utilized to record the change of users' rights on the ledger. And each BuyRecord/ReduceRecord points to a ServiceCallTime through its **ServiceCallTimeKey** field.

### 3.7.2 Functional implementation

Based on the design of data structures, we implement the invocation functions as listed in Table 1.

Generally, the invocation functions are divided into four categories: user-related functions, service-related functions, right-related functions and reward-related functions. Leveraging INK Account, we are able to recognize an invoker's identity with the help of chaincode stub interfaces such as **Transfer**, **GetAccount** and **GetSender**. Moreover, based on them, T-DSES has realized incentive mechanisms. To make better explanation, we will explain how we design the functions in the chaincode through the logic of services' status change, and how the invocation functions are utilized in typical scenarios.

**Table 1** Implemented functions in chaincode *service.go*

| Function | Description |
|---|---|
| *RegisterUser* | register a new user |
| *RemoveUser* | remove an existing user |
| *QueryUser* | query info about a specific user |
| *RegisterService* | register a new service (not mashup) |
| *PublishService* | publish a newly created service |
| *InvalidateService* | invalidate a specific service |
| *CreateMashup* | create a new mashup, compositing services |
| *QueryService* | query info about a specific service/mashup |
| *EditService* | edit info of a specific service or mashup |
| *QueryServiceByUser* | query all the services developed by one specific user |
| *QueryServiceByRange* | query services by a name range |
| *CallService* | request for the access of calling services |
| *GetSpecCallTime* | query a user's rights on a specific service |
| *GetCallTimesOfService* | query all the users' rights on a service |
| *ReduceCallTime* | reduce a users' rights of call times on a specific service |
| *RewardService* | reward a service's developer |

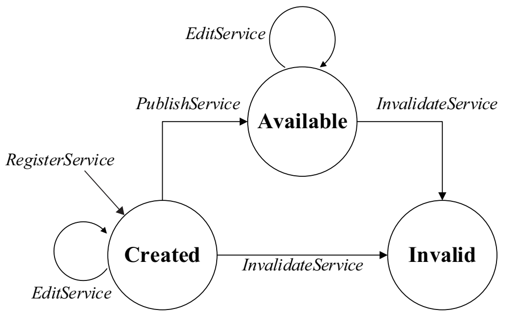


**Figure 9** Status change logic of a service in T-DSES.

### 3.7.3 The logic of services' status change

As mentioned before, a service in T-DSES has three different statuses. Figure 9 illustrates how a service's statuses change with different service-related functions.

### 3.7.4 Token circulation in typical behaviors

Two typical scenarios about the interaction of different participants are summarized in Section 3.2. In this subsection, based on the implemented functions in service.go, we will discuss the "*SToken*" circulation in different scenarios.

In the first scenario between users and developers as shown in Figure 4, in Step 4, to get the rights to call service *i*, user A's operations on Web front-end will propose an invocation of CallService interface through SDK. After reaching consensus, in the distributed ledger provided by blockchain network, certain amount of "*SToken*" will be transferred into developer B's account from A's account. The amount equals *i*'s price multiplies A's expected number of call times. Specially, Steps 5 and 7 are outside T-DSES, so they do not use any interfaces in *service.go*.

In the second scenario between mashup developers and service developers, in Steps 1 and 3, developers B and C utilize PublishService interface to register a new mashup or service in T-DSES. In the dealing process of PublishService proposed by developer C, he has to pay developer B "*SToken*" for the permission to integrate A's function, after which mashup *j* could be registered legally.

### 3.7.5 Introduction of CreateMashup and CallService

CreateMashup is an important invocation function, with which developers can register new services or mashups. CallService is also important to record or update users' rights on different services. Users can apply for outside access to developers' servers through it. The overall processes of these two functions are summarized as the tables in Example Function 1 and Example Function 2.

**Example Function 1:** CreateMashup

```
Input: Mashup information and list of invoked services
Output: Transaction record on the ledger
Procedure:
01.  Get the mashup's creator's address through GetSender
02.  Create a mashup record according to Service structure, where:
03.      a) Field IsMashup is set true
04.      b) Field Composited records all the invoked services
05.  For every invoked service
06.      Pay to its developer for providing related functionalities
         through Transfer(to, "SToken", amount) [1]
07.  End
08.  Store the newly-created service into the ledger
```

[1] *to* is the address of the service's developer, *amount* determines how many "*SToken*" are paid to the service's developer. In T-DSES, *amount* equals the price defined by developer.

Note that both CreatMashup and CallService utilize inherent interface **Transfer** provided by INKchain to make transformation of "*SToken*" between different participants, realizing value circulation in T-DSES.

### 3.7.6 Incentive mechanisms

Incentive mechanisms are designed from four aspects.

*(a) Contribution Record*

In the User data structure, **Contribution** field is adopted to record the degree of a user's contribution to the service eco-system, which is calculated as follows.

$$contribution_i = \ln(N_{Ti}+1) + \lambda \cdot \frac{N_{Ii}}{N_{Ti}} + \gamma \cdot \frac{N_{Pi}}{N_{Ti}} \tag{1}$$

where $\ln(N_{Ti}+1)$ represents the evaluation of quantity. $N_{Ti}$ is the total number of available services and mashups the developer has registered. Meanwhile, $\lambda \cdot \frac{N_{Ii}}{N_{Ti}}$ represents the evaluation of invocation quality (i.e.,

**Example Function 2:** CallService

```
Input: Service name and the number of intended call times
Output: Transaction record on the ledger
Procedure:
01.  Get the user's address through GetSender
02.  Check the legality of the existence and status of this service
03.  Create a ServiceCallTime record
04.  Pay for the rights to call the service
         Make "SToken" transfer through Transfer(to, "SToken", amount) [1]
05.  Store the newly-created record into the ledger
06.  Create and store a BuyRecord record into the ledger
```

[1] *to* is the address of the service's developer, *amount* equals price multiplies call times.

whether a service is invoked more frequently by other developers when creating new mashups), and $N_{Ii}$ is the total times that this user's services have been invoked by all the mashups in T-DSES. What's more, $\gamma \cdot \frac{N_{Pi}}{N_{Ti}}$ stands for the evaluation of practical quality (i.e., whether a service's server will get more call request by users outside T-DSES). $N_{Pi}$ is the total number of paid call times of all the users.

Here we introduce coefficient $\lambda$ & $\gamma$ to adjust the preference between the quantity index and quality index. Generally, we pay more attention to the quality of services and mashups registered in T-DSES.

*(b) Token Transfer when Creating Mashups*

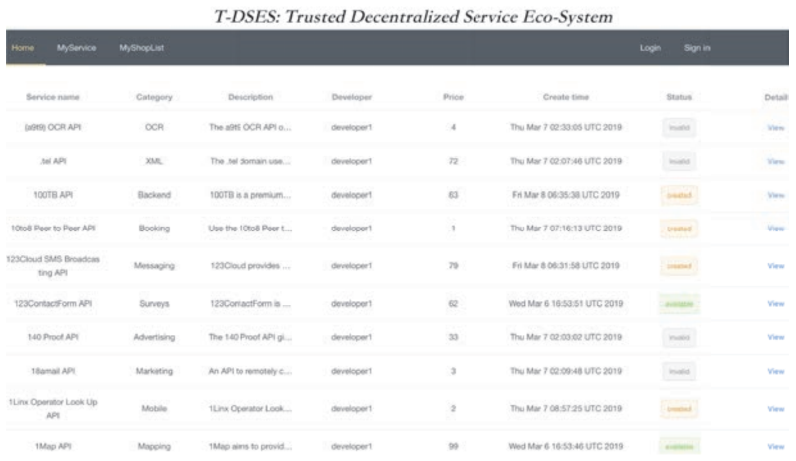
As mentioned in CreateMashup function, when creating a new mashup, the developer has to transfer a fixed amount of "*SToken*" to each invoked service's original developer to get the permission. Thus developers are motivated to create more general and more valuable services.

*(c) Token Transfer when Applying for Call Times*

As mentioned in CallTime function, when applying for call times of a service, the user has to transfer "*SToken*" to the service's developer. When the server outside T-DSES comes with a call request, it will check the user's rights to see if he is allowed to get a response. As a result, developers are motivated to register more practical and easy-to-use services to T-DSES.

*(d) Reward from Users*

The invocation function *RewardService* is also part of the designed incentive system in T-DSES. A user can reward the developer of a service or a



| Service name | Category | Description | Developer | Price | Create time | Status | Detail |
|---|---|---|---|---|---|---|---|
| (a9t9) OCR API | OCR | The a9t9 OCR API o... | developer1 | 4 | Thu Mar 7 02:33:05 UTC 2019 | invalid | View |
| .tel API | XML | The .tel domain use... | developer1 | 72 | Thu Mar 7 02:07:46 UTC 2019 | invalid | View |
| 100TB API | Backend | 100TB is a premium... | developer1 | 63 | Fri Mar 8 06:35:38 UTC 2019 | created | View |
| 10to8 Peer to Peer API | Booking | Use the 10to8 Peer t... | developer1 | 1 | Thu Mar 7 07:16:13 UTC 2019 | created | View |
| 123Cloud SMS Broadcasting API | Messaging | 123Cloud provides ... | developer1 | 79 | Fri Mar 8 06:31:58 UTC 2019 | created | View |
| 123ContactForm API | Surveys | 123ContactForm is ... | developer1 | 62 | Wed Mar 6 16:53:51 UTC 2019 | available | View |
| 140 Proof API | Advertising | The 140 Proof API gi... | developer1 | 33 | Thu Mar 7 02:03:02 UTC 2019 | invalid | View |
| 18amail API | Marketing | An API to remotely c... | developer1 | 3 | Thu Mar 7 02:09:48 UTC 2019 | invalid | View |
| 1Linx Operator Look Up API | Mobile | 1Linx Operator Look... | developer1 | 2 | Thu Mar 7 08:57:25 UTC 2019 | created | View |
| 1Map API | Mapping | 1Map aims to provid... | developer1 | 99 | Wed Mar 6 16:53:46 UTC 2019 | available | View |

**Figure 10** Home page of T-DSES web front-end.

mashup with any amount of "*SToken*" as he likes to thank the developer for the unremitting efforts. Developers would be encouraged, too.

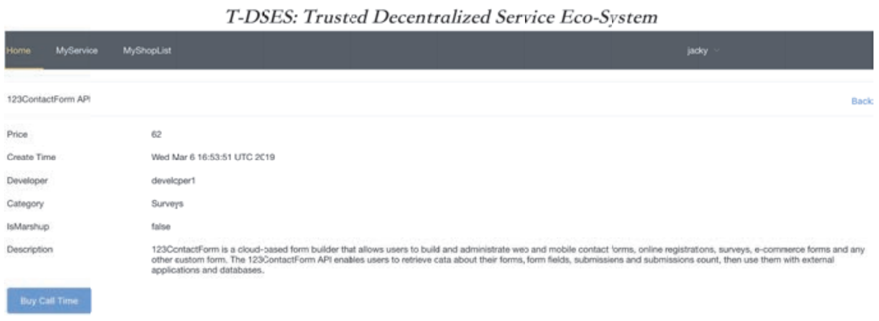
## 3.8 Design of Web Front-end

The whole front-end project is open source in the GitHub repository. Home page and service information page will be presented in this section as an example.

Figure 10 shows the website of **Home Page**. On the top is the navigation bar, providing links to home page, owned-service page and user-login page. There is a service list in the home page, containing profile information of each service. The price here refers to the amount of "*SToken*" needed to pay when invoking a service to create mashups, or requesting for the call permission outside.

Through the "View" button on the right of each service item, **Service Information Page** can be reached, as shown in Figure 11. Detailed information about a service is provided, such as the name, price, create time, developer, resource, description and so on. Resource refers to the real URL linked to servers where the service is deployed. In order to acquire the rights of calling the service, users should pay "*SToken*" through "Buy" button after logging in.

What's more, in **Owned-service Page**, a user can publish a new service or mashup. The balance of "*SToken*" and the value of contribution are also demonstrated in this page. Other pages are available in the GitHub repository.



123ContactForm API

|  |  |
|---|---|
| Price | 62 |
| Create Time | Wed Mar 6 16:53:51 UTC 2019 |
| Developer | developer1 |
| Category | Surveys |
| IsMarshup | false |
| Description | 123ContactForm is a cloud-based form builder that allows users to build and administrate web and mobile contact forms, online registrations, surveys, e-commerce forms and any other custom form. The 123ContactForm API enables users to retrieve data about their forms, form fields, submissions and submissions count, then use them with external applications and databases. |

**Figure 11** Service information page of T-DSES web front-end.

### 3.9 Trustworthiness of T-DSES

In summary, the underlying blockchain technology, the cloud-based network environment and the detailed design of logic coded in chaincode together contribute to the trustworthiness. T-DSES provides trusted information in three aspects as follows.

First, reliable information of services and mashups are provided by T-DSES through blockchain technology. Problems in conventional systems (e.g., security problem, authority control and lack of incentive mechanism) would be alleviated to a certain extent in T-DSES.

Second, reliable records of participants' rights of calling specific services are provided. Users can pay a service's developer with "*SToken*" for its access. Then he can call the service deployed on the developer's server. It solves the problem of value separation in conventional centralized systems.

Third, related participants' contributions to the system are provided in a trusted way. The value of contribution records the credibility degree of different participants.

## 4 Cloud-based System, Experiments and Discussions

In this section, we describe the implementation of our production-ready prototype of T-DSES in a case study, along with analyses on performance and robustness.

### 4.1 Data Set

ProgrammableWeb.com is a typical Web centralized service eco-system, which has been accumulating a variety of services and mashups since established in 2005. We crawled the data from its inception to June 2021, including over 24,000 services and 6,400 mashups, as the test data set in T-DSES.

## 4.2 Cloud-based Implementation

We use INKchain Version 0.13.0 as the basic blockchain library and implement our design of T-DSES. The versions of software and mechanisms in T-DSES are listed in Table 2.

We implement a T-DSES cloud-based environment in this paper. Detailed settings are listed in Table 3. In the scenario, there are three different organizations, who together maintain the blockchain network. Each organization has 2 peer nodes. And there are 3 nodes in the orderer cluster defined as Figure 7. All peers, orderers and client programs are containerized in Docker. The peers join a consensus protocol of the blockchain. A tailored Web front-end is also provided in T-DSES. The Kubernetes masters and nodes are deployed in standard cloud machines with 2c4g.

**Table 2** Versions of software and mechanisms

| **Item** | **Version** |
|---|---|
| INKchain | 0.13.0 |
| INKchain SDK | 0.17.4 |
| OS of cloud machines | Ubuntu 16.04 |
| Language of chaincode | Go 1.9.2 linux/amd64 |
| Containerization | Docker 1.35 |
| Container Management | Kubernetes 1.10.11 |
| Digital Signature Alg. | ECDSA (256 bit key) |
| Hash Function | SHA-256 |
| Framework of Web Front-end | Element UI 2.5.2 |

**Table 3** Cloud-based network environment settings

| **Item** | **Configuration** |
|---|---|
| K8S Master | 2c4g * 3 in Public Cloud |
| K8S Node | 2c4g * 5 in Public Cloud |
| # of Orgs | 3 |
| # of Peers in each Org | 2 |
| # of nodes in Orderer Cluster | 3 |
| Web Front-end Server | 2c4g * 1 in Public Cloud |

## 4.3 Performance Analysis

The goal of T-DSES is to build a well-organized and well-functioning decentralized service eco-system. In T-DSES, the operation requests waited to be executed can be divided into two types, queries and transactions. Because data are stored in the form of key-value pairs in the ledger, query operations refer to checking the value by key. Query operations have no effect on the recorded data, and the corresponding functions include *QueryUser*, *QueryService*, *QueryServiceByUser*, etc. Transaction operations refer to the update of ledger, such as adding, deleting, and modifying key-value pairs. The functions related to transaction operations include *PublishService*, *CreateMashup*, *CallService*, etc. Therefore, when considering the performance of the blockchain network, we conduct experiments towards both query operations and transaction operations.

### 4.3.1 Performance analysis of query operations

The performance of query operations is related to two key variables: the stored data size and the concurrency query numbers. Firstly, we tested the influence of data size on query performance. We made the number of stored transaction data vary from 4,000 to 10,000 with separation 2000. The number of concurrent query requests automatically increased to 200 at a fixed interval within 20 minutes in each case.

The experimental results are presented in Table 4. Based on the cloud-based environment, the success rate of request is almost 100% under all the circumstances. Average response time is approximately 523 ms with 4000 stored data size.

Secondly, we tested the influence of concurrency query numbers on query performance. We set a fixed data size to be 3000. The response time is recorded in Figures 12, and 13 records the throughput per second ($tps$).

From Figure 12, we can see that as the concurrency number increases, so does the response time. And Figure 13 reveals that after reaching stability,

**Table 4** Experimental results on query performance

| # of Trans. Data | 4,000 | 6,000 | 8,000 | 10,000 |
|---|---|---|---|---|
| # of Total Request | 123,149 | 126,479 | 126,066 | 125,989 |
| # of Failed Request | 2 | 3 | 2 | 4 |
| Request Success Rate (%) | 100 | 100 | 100 | 100 |
| Avg. Response Time (ms) | 525.63 | 1027.07 | 1021.54 | 1030.83 |

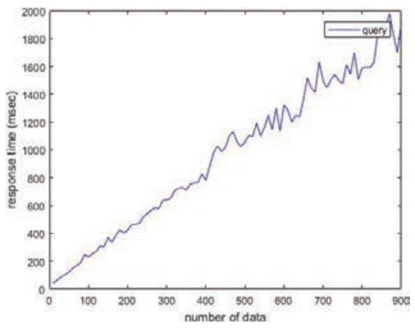


**Figure 12** Query response time.

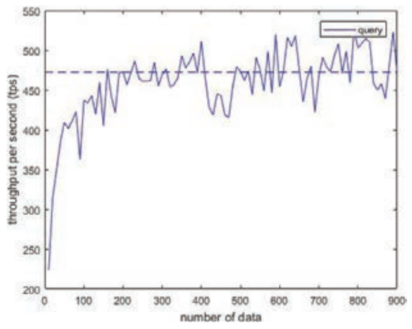


**Figure 13** Query $tps$.

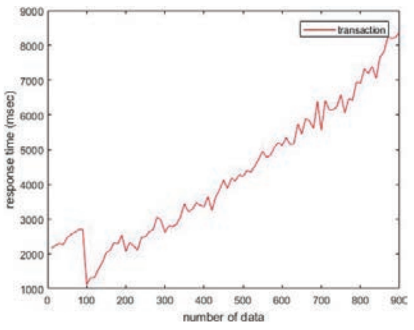


**Figure 14** Transaction response time.

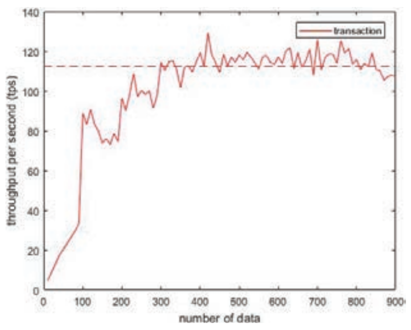


**Figure 15** Transaction $tps$.

$tps$ stays unchanged even the number of concurrency increases. In our experimental network, the average $tps$ of query is 473.

In summary, the network query performance of T-DSES is feasible for practical use.

### 4.3.2 Performance analysis of transaction operations

In this experiment, we tested the performance of transaction operations. Through making transaction requests, we can publish new services, call existing mashups, etc. The response time and $tps$ are recorded in Figures 14 and 15 respectively.

We can draw a similar conclusion from Figure 14 that as the number of concurrent requests increases, the overall response time of transactions also increases. Compared with Figure 12, what's different is that the average response time of a transaction is significantly higher than that of a query. Because transaction operation is a complicated process: it includes SDK making a request, peer node executing chaincodes, orderer node sorting transaction, etc. Besides, Figure 14 shows that when the number of concurrency reaches 100, the response time drops significantly. This is because the maximum transaction number in a block is set to 100 and the block wait time is set to 2s when configuring the network in our experiment. When the number of concurrency is less than 100, the transactions will wait for two seconds to be packaged into a block. And when number = 100, without waiting, there is an obvious inflection point in Figure 14. The same phenomenon also occurs when the number is an integer multiple of 100. But due to the increase in average response time and the noise caused by network fluctuations, this phenomenon is not easily observed when the number of concurrency is high.

Figure 15 shows that the average $tps$ of transaction operations is 112, which is also lower than that of query operations.

#### 4.3.3 The requirement of service eco-system

Take Programmable.com as an example. Since established in 2005, the average daily service-publishing rate is about 4.10. The peak value of daily service-publishing is about 110, and the peak value of daily mashup-publishing is around 40. Above all, operations around a service eco-system is usually not of high-frequency. And according to our experiment, the $tps$ index is about 110, which is enough to meet the needs of Programmable.com. Even in larger business application scenarios, replacing the basic framework we construct with a larger node cluster can greatly improve the efficiency of transmission.

### 4.4 Robustness Analysis

A cloud-based environment should also be fault tolerant to some extent. To test the robustness of T-DSES, we make faults artificially and then make a transaction proposal and query the results to see whether the system works regularly.

Based on the environment in Table 3, the consensus is based on Kafka. We have set the endorsement policy for *service.go* as necessary for Org 1. We shutdown the cloud machines to simulate system failure. As long as one peer server in Org. 1 and the orderer cluster are under normal operation, T-DSES could provide service normally.

One step further, we implement another experimental environment based on BFT consensus. In this scenario, as long as the number of failed peer nodes is less than half, T-DSES could provide service normally.

## 5 Related Work

A service eco-system is a logical collection of Web services as well as their compositions [1–3]. Existing works mainly focus on systemic management or studying related data to make analyses and recommendation [26]. However, existing service eco-systems typically rely on some centralized service registries as "middle people" to record service behaviors, which could result in problems such as trust issues, security, authority control, and high cost of maintenance [22].

Bitcoin [8] is the first successful attempt to construct a commercialized decentralized system. Since then, blockchain technology has grown beyond crypto-currencies to support real applications [36, 37]. Ethereum [5], the first one realizing chaincode, enables people to develop all kinds of DApps. In industry scenarios, Hyperledger Fabric [13, 20, 21] is the most popular solution among companies. As an extension of Fabric, INKchain is a newly open-source permissioned blockchain platform, enabling account systems and customized token issuance. Permissioned blockchains have advantages in authority control and higher performance [20]. Thus they have been widely used to develop decentralized applications [14, 27, 38]. In this paper, as the first attempt to design a decentralized service eco-system, we have built the overall T-DSES framework on top of INKchain.

Both the network component of Fabric and INKchain are containerized. Generally, cloud-based environments are deployed in public cloud or a hybrid architecture, where security problems are the main concerns (e.g, availability, confidentiality and integrity) [28]. Kubernetes [29] is a state-of-the-art and open-source container orchestration system for automating application deployment, scaling, and management across a cluster of hosts. In this paper, we utilize Kubernetes to build a cloud-level environment of T-DSES, which provides high availability for the system.

## 6 Conclusions

In this paper, as the first attempt to bring decentralization and trustworthiness to service eco-systems, we have described our design and development of the Trusted Decentralized Service Eco-System (T-DSES) based on a newly open-source permissioned blockchain called INKchain. We present how to build a cloud-based T-DSES system along with a Web front-end. To demonstrate the feasibility and effectiveness of our design, analyses and experiments on performance as well as robustness have also been conducted based on a real-world data set. The original codes are open source.

In our future work, we will utilize more technologies (e.g., Decentralized Identifier, or DID) to perfect the design of T-DSES. Furthermore, we will apply the concept of T-DSES in more practical application scenarios such as data sharing to verify the effectiveness of our prototype.

## Acknowledgements

This research has been partially supported by the National Key Research and Development Program of China (No. 2018YFB1402500).

## References

[1] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou. On the Evolution of Services. *Transactions on Software Engineering*, Vol. 38, no. 3, pp. 609–628, 2012.

[2] D. M. Barros A. The Rise of Web Service Ecosystems. *It Professional*, Vol. 8, no. 5, pp. 31–37, 2006.

[3] X. Liu, Y. Hui, W. Sun, and H. Liang. Towards Service Composition based on Mashup. *Proceedings of IEEE World Conference on Services (SERVICES)*, pp. 332–339, 2007.

[4] K. Huang, Y. Fan, and W. Tan. An Empirical Study of Programmable Web: A Network Analysis on a Service-Mashup System. *Proceedings of IEEE International Conference on Web Services (ICWS)*, pp. 552–559, 2012.

[5] Ethereum. *Ethereum*. In https://www.ethereum.org/.

[6] K. C. Bhardwaj and R. K. Sharma. Machine Learning in Efficient and Effective Web Service Discovery. *Journal of Web Engineering*, Vol. 14, pp. 196–214, 2015.

[7] S. Kamath and AV S. Semantic Similarity based Context-aware Web Service Discovery Using NLP Techniques. *Journal of Web Engineering*, Vol. 15, pp. 110–139, 2016.

[8] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. *Consulted*, 2008.

[9] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang. Untangling Blockchain: A Data Processing View of Blockchain Systems. *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, 2018.

[10] Ripple, *Ripple*. In https://ripple.com.

[11] Melonport. Blockchain Software for Asset Management. In http://melonport.com.

[12] J. Morgan and O. Wyman. Unlocking Economic Advantage with Blockchain. 2016.

[13] Hyperledger. *Hyperledger Fabric*. In https://www.hyperledger.org/projects/fabric.

[14] S. A. Blockchain Based Distributed Control System for Edge Computing. *Proceedings of International Conference on Control Systems and Computer Science*, pp. 667–671, 2017.

[15] L. Cocco, A. Pinna, M. Marchesi. Banking on Blockchain: Costs Savings Thanks to the Blockchain Technology. *Future internet*, vol. 9, no. 3, pp. 25, 2017.

[16] X. Tai, H. Sun, Q. Guo. Electricity Transactions and Congestion Management based on Blockchain in Energy Internet. *Power Syst. Technol*, pp. 3630–3638, 2016.

[17] AnantJhingran. How and Why to Transform Your Business into a Digital Ecosystem. In https://www.programmableweb.com/news/how-and-why-to-transform-your-business-digital-ecosystem/analysis/2018/01/11.

[18] M. Ali, J. Nelson, R. Shea, and M. J. Freedman. Blockstack: A Global Naming and Storage System Secured by Blockchains. *Proceedings of USENIX Annual Technical Conference (USENIX ATC)*, pp. 181–194, 2016.

[19] J. Garay, A. Kiayias, and N. Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. *Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pp. 281–310, 2015.

[20] M. Vukolić. Rethinking Permissioned Blockchains. *Proceedings of ACM Workshop on Blockchain, Cryptocurrencies and Contracts (ACM)*, pp. 3–7, 2017.

[21] C. Cachin. Architecture of the Hyperledger Blockchain Fabric. In https://www.zurich.ibm.com/dccl/papers.

[22] Z. Gao, Y. Fan and C. Wu, J. Zhang and C. Chen. DSES: A Blockchain-Powered Decentralized Service Eco-System. *Proceedings of IEEE International Conference on Cloud Computing*, pp. 25–32, 2018.

[23] X. Yi. Hash Function based on Chaotic Tent Maps. *IEEE Transactions on Circuits & Systems II Express Briefs*, vol. 52, no. 6, pp. 354–357, 2005.

[24] M. Bellare, and P. Rogaway. Optimal Asymmetric Encryption. *Proceedings of The Workshop on the Theory and Application of of*

*Cryptographic Techniques*, pp. 92–111. Springer, Berlin, Heidelberg, 1994.

[25] R. C. Merkle. A Certified Digital Signature. *Proceedings of Advances in Cryptology – CRYPTO '89, International Cryptology Conference*, pp. 218–238. Santa Barbara, California, USA, 2007.

[26] Z. Gao, Y. Fan and C. Wu and W. Tan and J. Zhang. Service Recommendation From the Evolution of Composition Patterns. *Proceedings of IEEE International Conference on Services Computing*, pp. 108–115, 2017.

[27] S. Kiyomoto, M. S. Rahman, and A. Basu. On Blockchain-based Anonymized Dataset Distribution Platform. *Software Practice and Experience*, pp. 85–92, 2017.

[28] A. Modak, S. Chaudhary, P. Paygude, and S. Ldate. Techniques to Secure Data on Cloud: Docker Swarm or Kubernetes? *Proceedings of Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, pp. 7–12, 2018.

[29] H. V. Netto, L. C. Lung, M. Correia, A. F. Luiz, and L. M. S. de Souza. State Machine Replication in Containers Managed by Kubernetes. *Journal of Systems Architecture*, vol. 73, pp. 53–59, 2017.

[30] K. Goarany, G. Kulczycki, and M. B. Blake. Research on Kubernetes' Resource Scheduling Scheme. *Proceedings of ACM International Workshop on Search and Mining User-generated Contents (SMUC)*, pp. 71–78, 2010.

[31] J. Ellingwood. An Introduction to Kubernetes. *Retrieved April*, 2017.

[32] K. Hightower, B. Burns, and J. Beda. Kubernetes: Up and Running: Dive Into the Future of Infrastructure. *O'Reilly Media, Inc*, 2017.

[33] G. Sayfan. Mastering Kubernetes. *Packt Publishing Ltd*, 2017.

[34] E. Truyen, D. Van Landuyt, V. Reniers, A. Rafique, B. Lagaisse, and W. Joosen. Towards a Container-based Architecture for Multi-tenant SaaS Applications. *Proceedings of the 15th International Workshop on Adaptive and Reflective Middleware*, p. 6, 2016.

[35] C. Pahl and B. Lee. Containers and Clusters for Edge Cloud Architectures–A Technology Review. *2015 3rd international conference on future internet of things and cloud*, pp. 379–386, 2015.

[36] R. Li, T. Song, B. Mei, H. Li, X. Cheng, and L. Sun. Blockchain for Large-scale Internet of Things Data Storage and Protection. *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 762–771, 2018.

[37] W. Viriyasitavat, L. Da Xu, Z. Bi, and A. Sapsomboon. Blockchain-based Business Process Management (BPM) Framework for Service

Composition in Industry 4.0. *Journal of Intelligent Manufacturing*, pp. 1–12. 2018.

[38] S. Rouhani, V. Pourheidari, and R. Deters. Physical Access Control Management System Based on Permissioned Blockchain. *Proceedings of IEEE International Congress on Cybermatics*, pp. 1078–1083, 2018.

[39] J. Kreps, N. Narkhede, and J. Rao. Kafka: A Distributed Messaging System for Log Processing. *Proceedings of the NetDB*, pp. 1–7, 2011.

[40] Apache Software Foundation. Kafka. In https://kafka.apache.org, pp. 17–24.

[41] Apache Software Foundation. Zookeeper. In https://zookeeper.apache.org.

## Biographies

**Xing Wu** received the BS degree in control theory and application from Tsinghua University, China, in 2017. He is currently working toward the PhD degree in the Department of Automation, Tsinghua University. His research interests include services computing, service recommendation, federated learning and blockchain.

**Zhenfeng Gao** received the PhD degree in control theory and application in 2018 from Tsinghua University, China. He is currently working as the postdoctor at the Graduated school at shenzhen, Tsinghua University as well as the postdoctoral research center at Sangfor Technologies Inc. His research interests include services computing, service recommendation, big data and blockchain technology.

**Yushun Fan** received the PhD degree in control theory and application from Tsinghua University, China, in 1990. He is currently a professor with the Department of Automation, Director of the System Integration Institute, and Director of the Networking Manufacturing Laboratory, Tsinghua University. From September 1993 to 1995, he was a visiting scientist, supported by Alexander von Humboldt Stiftung, with the Fraunhofer Institute for Production System and Design Technology (FHG/IPK), Germany. He has authored 10 books and published more than 300 research papers in journals and conferences. His research interests include enterprise modeling methods and optimization analysis, business process reengineering, workflow management, system integration, object-oriented technologies and flexible software systems, petri nets modeling and analysis, and workshop management and control.

**Xiu Li** received the PhD degree in mechanical manufacturing and automation from Nanjing University of Aeronautics and Astronautics in 2000. She was once a visiting scientist at University of Hong Kong, the Hong Kong Polytechnic University and Georgia institute of technology. She is currently a professor with the Department of Information, Shenzhen Graduate School, Tsinghua University. She has published more than 100 papers in international transactions and conferences. Her research interests include intelligent systems, data mining and pattern recognition.

**Liang Gu** received the PhD degree in Computer Software and Theory from Peking University in 2010. He worked as an associate research fellow at Yale university from 2010 to 2015. He is currently the chief scientist and the director of Sangfor Research Institute at Sangfor Technology Inc. As the person in charge of r&d technology at Sangfor, he is responsible for the technical framework improvement of a series of core products, including NGAF, AC, a Cloud HCI, aSAN and so on. These products have gained a leading market share in China and have been recognized by users and the market.

**Jia Zhang**. received her PhD degree in computer science from the University of Illinois at Chicago. She is currently the Cruse C. and Marjorie F. Calahan Centennial Chair in Engineering, Professor of Department of Computer Science at Southern Methodist University. Her research interests emphasize the application of machine learning and information retrieval methods to tackle data science infrastructure problems, with a recent focus on scientific workflows, provenance mining, software discovery, knowledge graph, and their interdisciplinary applications. Dr. Zhang has co-authored one textbook ”Services Computing” and has published over 170 refereed journal papers, book chapters, and conference papers. Dr. Zhang has served as an associated editor of the IEEE TSC since 2008. She served as Program Committee Chair for IEEE SCC (2020), ICWS (2019), CLOUD (2018), and BigData Congress (2017). She is a senior member of the IEEE.

**Chang Chen**. received the BS degree in control theory and application from Tsinghua University. He once worked at IBM-CRL as a senior researcher. He is now the CTO of Zhigui Technology Inc, as well as director of blockchain finance research center, School of Economics and Management, Tsinghua University. He was an early researcher of blockchain technology. He is a contributor of the Hyperledger open-source project. His research interests include cloud computing and blockchain technology.

**Hao Zhang**. received the BS degree from Software Engineering, San-Jiang University, China in 2014. He is working in Zhigui Technology Inc. His research interests include blockchain development.

**Qiang Wang**. received the BS degree from Electronic and Information Engineering,Shandong Technology and Business University, china in 2012. He once worked at IBM-CRL, and his main job was to assist researchers with the realization of blockchain projects. He is now working at Zhigui Technology Inc. His research interests include DevOps and blockchain applications.