ONMCA: One-Network-Multi-Chain Architecture for Customizable Asset-Oriented Blockchain Systems

Liang Wang
Hebei University
Wenying Zhou
zwy@stumail.hbu.edu.cn
Hebei University
Lina Zuo
Hebei University
Haibo Liu
Hebei University

Research Article

Keywords: Digital asset management, Blockchain, Software architecture, Customizable design

Posted Date: October 6th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3386787/v1

Additional Declarations: No competing interests reported.

Version of Record: A version of this preprint was published at Peer-to-Peer Networking and Applications on April 11th, 2024. See the published version at https://doi.org/10.1007/s12083-024-01698-8.

ONMCA: One-Network-Multi-Chain Architecture for Customizable Asset-Oriented Blockchain Systems

Liang Wang[1,2], Wenying Zhou[1*†], Lina Zuo[1,2†], Haibo Liu[1,2†]

[1*]School of Cyber Security and Computer, Hebei University, Qiyidong Road, Baoding, 071000, Hebei, China.
[2]Hebei Key Laboratory of High Trusted Information System, Hebei University, Qiyidong Road, Baoding, 071000, Hebei, China.

*Corresponding author(s). E-mail(s): zwy@stumail.hbu.edu.cn;
Contributing authors: wangl@hbu.edu.cn; zuolina@hbu.edu.cn; liuhaibo@hbu.edu.cn;
†These authors contributed equally to this work.

Abstract

The development of modern digital economies requires trusted digital asset management (DAM), for which blockchain technology is increasingly being adopted. However, the current architecture for constructing blockchain-based DAM systems (BDAMSs) is inadequate. Existing BDAMSs adopt a modified layered architecture, which enriches the database layer by adding a blockchain platform that acts as a third party to process DAM business logic with pre-written smart contracts. This architecture faces four issues that make it non-credible and non-customizable to DAM demands: 1) pseudo decentralization, 2) not asset-oriented, 3) contract dependency, and 4) high load of chains. To overcome these issues, we propose the One-Network-Multi-Chain Architecture (ONMCA), which allows multiple heterogeneous chains to be established within a same network. ONMCA enables diverse digital assets to be managed in a customizable way through the following features: 1) asset stakeholders are allowed to join the blockchain network, eliminating third parties; 2) transactions are designed to portray the changes in asset states, making the system asset-oriented; and 3) a control layer is added to take over business logic, and smart contracts are forced to regulate asset transactions full-time. We formalize ONMCA and analyze it comprehensively, and the results show that ONMCA meets the requirements of DAM and is qualified to build credible and adaptive BDAMSs.

Keywords: Digital asset management, Blockchain, Software architecture, Customizable design

1 Introduction

1.1 Background

Digital asset management (DAM) underpins the innovation of today's digital society and involves the basic benefits of everyone. The driving information technologies fundamentally shape all aspects of modern economies and industries [1–3]. What ensues is the generation, creation and accumulation of massive digital assets. They will be mined for immeasurable value if we can manage them well, and DAM systems (DAMSs) are necessary for achieving this goal. On the other hand, those digital assets are also in the interests of individuals and organizations. An improper DAMS will be bound to undermine people's control over

the ownership, circulation and application of their digital assets [4, 5]. It is therefore significant to ensure the reliability of DAMS design.

Blockchain is increasingly being adopted to build trust for DAM because of its remarkable qualities, such as network decentralization, records tamper-resistance, and histories traceability [6–8]. Blockchain-based DAMSs (BDAMSs) are expected to provide enhanced reliability for stakeholders to operate their digital assets. They need to meet several basic requirements. Firstly, stakeholders should be allowed to participate in DAM by joining the blockchain network, and supervise each other's behavior by reaching consensus. Secondly, data models of the blockchain should be customizable for diverse digital assets, and ownership and usage rights of those digital assets should be easily traced back through these data models. Finally, smart contracts should be designed to regulate the trading of digital assets rather than handle complex business logic.

However, existing BDAMSs do not work well as intended in meeting the above DAM requirements. They follow a modified layered architecture, which enriches the database layer by adding a blockchain platform that acts as a third party to process business logic with pre-written smart contracts. Four issues are hard to solve in this architecture:

1. Pseudo decentralization. The blockchain platform is usually controlled by a third party, like Ethereum[1]. Normal asset holders will have no opportunity to become a node of the blockchain platform, so nothing can be done once the platform suddenly fails. The credibility of BDAMSs relies on the reputation of blockchain service providers.
2. Not asset-oriented. The on-chain transactions only record the operations on smart contracts, but not directly on digital assets. For example, the deployment, update, and function calls of a smart contract will be recorded as transactions, but these transactions may not be directly related to the state changes of assets.
3. Contract dependency. An asset holder must program a comprehensive contract to define his/her specific digital assets and handle related DAM tasks. The definition of these assets is described simply by the contract's state variables rather than the blockchain's data models, so changes in these assets are difficult to interlink clearly. This means that assets exist by contracts. Once a contract lapses, the assets that depend on it are also unavailable.
4. High load of chains. It will be a large load for smart contracts to run complicated business logic. Especially, a DAM task may involve cooperation of multiple chains, which increases coding complexity of smart contracts.

This modified layered architecture partially results from the commercial packaging of the concepts – SMART CONTRACT. Since Ethereum launched its blockchain-based smart contract platform, the concept of smart contract has gradually been accepted as a ticket to using blockchain. Nowadays, smart contracts refer more to programming interfaces than legal agreements [9]. Many Cloud service providers have built their own smart contract platforms on their Cloud resources [10, 11]. They adopted a form of service provision similar to Ethereum, further consolidating the mode this transitional architecture is applied. We have noticed in practice that DAM tasks rely primarily on non-contract programming, and that blockchain systems become just third-party endorsers whose credibility is very hard to guarantee.

No one would believe that blockchain's potential stops there. Essentially, blockchain was born for DAM. In the case of Bitcoin[2], cryptocurrencies are internalized into the data models of the blockchain. Each node in Bitcoin network has the ability to manage the assets under its jurisdiction. In the area of DAM, this ability needs to be preserved, which of course requires the help of a more advanced architecture.

System architecture is mainly used to guide and validate the design of a class of software systems [12–14]. A good system architecture avoids catastrophic malfunctions and security issues in a specific application and ensures that the target software supports the business logic of the specific application. This paper mainly discusses the

[1] https://ethereum.org

[2] https://bitcoin.org

design of blockchain system architecture from the point of asset-oriented application.

1.2 Motivation

The analysis in Section 1.1 has shown that the current blockchain system architecture is not conducive to the development of BDAMS. Our motivation comes from the desire to address the lack of credibility and customizability of existing BDAMSs and better manage diverse types of digital assets. To this end, we conducted this research from the perspective of architecture innovation.

1.3 Contributions

We propose a new system architecture named One-Network-Multi-Chain Architecture (ONMCA), and the contributions of our work are as follows:

1. Digital asset stakeholders are allowed to join the blockchain network to jointly oversee asset transactions, keep local storage of their assets, and make the system free from third-party control. This solves the problem of "pseudo decentralization".
2. Multiple heterogeneous blockchains are allowed to coexist in the same network. Data models and functionality of each chain can be customized according to the types of digital assets. This solves the problem of "not asset-oriented".
3. A control layer is added to take over business logic, and smart contracts are forced to regulate asset transactions exclusively. This solves the problems of "contract dependency" and "high load of chains".

2 Related Work

At present, there are two main types of blockchain architectures: the chain-app-separated architecture (CASA) based on permission-less blockchain, and the chain-app-merged architecture (CAMA) based on permission-ed blockchain.

CASA considers blockchain as a data service platform, similar to a database. The difference is that the platform is provided by a third party rather than deployed locally. Therefore, when designing an application based on CASA, the upper-level apps and the underlying data service platform are separated. There are many examples that make CASA the most popular system architecture today. MedShare [15], HealthBlock [16], and BiiMED [17] are blockchain-based medical data management systems that implemented on Ethereum. Ethereum serves as the third party to provide data services for these systems. Commonly, those applications that have adopted CASA use smart contracts of the blockchain platforms to realize business logic [18]. BC-Rec [19] is a recommender system framework based on Ethereum and uses smart contracts to ease cold-start, sparsity, and shilling attack problems. AuthROS [20] is a secure data sharing method that codes smart contracts to ensure the immutability and security of private data flow between nodes. EthReview [21] aims at exploiting Ethereum smart contracts to control the number of discounted tokens to reward honesty in the system and punish fraud. However, the immutability and coding complexity of smart contracts make the above systems very hard to extend and expose them to some potential risks. Lu et al. [22] introduced a cross-business DAMS that reduced contract deployment risks to some extent, whereas it is also based on CASA. In fact, all the four issues we mentioned in Section 1.1 largely stem from this architecture.

Compared to CASA, CAMA emphasizes the native application support of blockchain. In CAMA, the servers that provide applications are also the nodes of the blockchain network. The blockchain network is maintained by nodes belonging to application users, so it is no longer a third party. Each peer node is allowed to perform business logic individually and some special nodes maintain the consistency of chain data through consensus. Hyperledger Fabric (HF)[3] is one of the representative techniques of CAMA. It mitigates the issues of "pseudo decentralization" and "not asset-oriented" we mentioned in Section 1.1 to some extent. Based on HF, Chen et al. [23] built a a personnel electronic information management system to improve the efficiency of storage and synchronization in management processes. The drawback of this system was that it did not fit well with different data types. Then, FAbAsset [24] was proposed to support multiple data types in DAM, like fungible token and non-fungible token, but it

[3] https://www.hyperledger.org/use/fabric

was inefficient when overusing channels to handle DAM business. Rodrigo et al. [25] proposed UniCon architecture, which was similar to HF, to build BDAMSs that could manage large scale digital assets. The key idea of UniCon is to allow BDAMSs to track the ownership of assets, which makes it possible to solve the problems of low flexibility, limited scalability, and high transaction cost. However, the weak adaptability of multiple data types cannot be solved unless the blockchain data models of CAMA can directly support the storage of assets.

In short, neither CASA nor CAMA completely solved the issues we mentioned in Section 1.1. Therefore, we need to design a more appropriate architecture for building BDAMSs that better meet the requirements of DAM.

3 Preliminaries

3.1 System Architecture

Three elements – components, connections, and configuration – are key considerations in system architecture as they define the structure, interactions, and behavior of the system [26].

Components refer to the individual building blocks or elements of a system architecture [27]. They can include hardware devices, software modules, databases, services, and other functional units. Components can be both physical and virtual. The selection, arrangement, and interaction of components are crucial for defining the overall structure and functionality of the system.

Connections in system architecture refer to the interactions and relationships between the components of a system [28]. They define how data, control, and communication flow between different parts of the system. Connections can involve various elements such as communication protocols, network infrastructure, application programming interfaces (APIs), and integration points. Establishing reliable and efficient connections between components is essential for the system to operate smoothly.

Configuration refers to the settings, parameters, and arrangements that determine the behavior and operations of the system [29]. It includes both the initial setup and ongoing management of the system's components. Configuration involves defining options, preferences, permissions, and rules that configure the behavior of individual components or the system as a whole. Proper configuration management ensures that the system operates according to the desired specifications and requirements.

3.2 Blockchain

Blockchain is an immutable distributed database for recording data in a chain structure, where each block contains a set of transaction records and is linked to the previous block in a cryptographic way [30]. Its striking features include decentralization, immutability, and traceability.

1. Blockchain implements decentralization by utilizing a network of nodes that collectively verify and validate transactions through consensus mechanisms, eliminating the need for a central authority. This decentralized approach ensures transparency, security, and removes single point of failure.
2. Blockchain achieves immutability by using cryptographic hash functions to create a unique hash for each block, which is based on the data within that block [31]. Any change to the data within a block would result in a different hash, making it evident that the block has been tampered with. Additionally, the block's reference to the previous block's hash creates a chronological chain, further securing the data's integrity.
3. Blockchain enables traceability by recording all transactions in a transparent and permanent manner. Each transaction is linked to previous transactions through cryptographic hashes, forming an immutable chain of records. This allows for easy tracing of the origin, movement, and ownership of assets or information stored on the blockchain, enhancing transparency and accountability.

The combination of these features makes blockchain a powerful technology with applications in various fields, including cryptocurrency, financial services, DAM, supply chain management, and more.

3.3 Formal Methods

Formal methods are a sort of techniques for modeling and verifying computer system design based on

strict mathematical foundation [32]. For designing of a system architecture, we need to know if this architecture is applicable before we can use it. So, we use formal methods to provide a framework on which the architecture can be described, designed and verified. Three aspects of formal methods will be used in this paper:

1. System modeling: Describe system S and its behavior by constructing model M.
2. Formal specification: Describe the constraints Q of system S by evaluating its key properties, such as security and flexibility.
3. Formal verification: Prove that the model M satisfies the formal specification Q (i.e. $M \vDash Q$).

3.4 Bilinear Map

Let G_S and G_T are both multiplicative cyclic groups of prime order q. A bilinear map $e : G_S \times G_S \to G_T$ must satisfy the following characteristics:

1. Bilinearity: For $\forall a, b \in Z_q^*$ and $\forall p \in G_S$, there is $e(p^a, p^b) = e(p, p)^{ab}$.
2. Non-degeneracy: $e(p, p) \neq 1$, where $p \in G_S$.
3. Computability: Exist a computationally efficient algorithm for computing $G_S \times G_S \to G_T$.

4 Architecture Modeling

In this section, we will begin by summarizing the fundamental requirements for the design of ONMCA. Next, we will overview and formalize the architecture model of ONMCA.

4.1 Requirements

It is necessary to briefly review the classification of digital assets before analyzing requirements of architecture design, because this is a priority for DAM on the one hand and closely related to the design of blockchain data models on the other. Digital assets can be classified in various ways depending on different taxonomies, as shown in Table 1.

For the sake of explanation, we synthesize the above classifications and categorize digital assets based on transaction modes of their property rights, as shown in Table 2, where d denotes a digital asset, u the usage right of d, and o the ownership of d. The transaction mode of transfer-type assets is to change their ownership, for example, a BTC (the currency unit of Bitcoin) is transferred from A to B. In contrast, the transaction mode of distribution-type assets is to distribute their usage rights, for example, A authorizes B and C to use copies of d. More specially, the transaction of value-added assets refers to changing their character through adding new elements to their bodies, and each element may has its own property rights, for example, A creates d' by adding a new idea to d, then A partially owns d'.

Table 1 Digital assets classification.

Basis	Class	Examples
By type	Test-based assets	Documents,reports, e-books
	Multimedia assets	Images,videos,audios
	Code-based assets	Software applications, scripts, APIs
	Financial assets	Cryptocurrencies, digital stocks, digital bonds
	IP assets	Patents, trademarks, copyrights
By purpose	Marketing assets	Logos, branding materials, social media graphics
	Sales assets	Product demos, brochures, pricing sheets
	Educational assets	E-learning courses, webinars, tutorials
	Operational assets	Workflow documents, process diagrams, technical manuals
	Legal assets	Contracts, agreements, policies
By value	Core assets	Proprietary software, patented technology, unique content
	Support assets	Marketing collateral, customer data, support documentation
	Discretionary assets	Nonessential software, experimental projects
By Lifecycle	Conceptual	Initial ideas, brainstorming, sketches
	Development	Prototypes, wireframes, mockups
	Finalized	Completed products, designs, content
	Retired	Obsolete software, outdated content

Table 2 Classification of digital assets based on transaction modes of property rights.

Category	Transaction mode	Typical asset
Transfer-type	$A \xrightarrow{d} B \xrightarrow{d} C^{ou}$	Cryptocurrencies
Distribution-type	$B^{u} \xleftarrow{d} A^{ou} \xrightarrow{d} C^{u}$	Data
Value-added	$d \xrightarrow{A^{ou}} d' \xrightarrow{B^{ou}} d''$	Documented ideas

Inevitably, the diversity of digital assets will bring about the coding complexity of smart contracts in developing a BDAMS. At the same time, DAM tasks also have variety depending on the assets they operate on. We focus on the DAM tasks that manage the on-chain lifecycle of different assets as shown in Figure 1. The common DAM tasks and their objectives are listed as follows:

- Register: Registering an asset on the blockchain.
- Unregister: Unregistering an asset from the blockchain.
- Transfer: Changing the ownership of an asset.
- Distribution: Dispensing the usage license of an asset.
- Minting: Initializing the features of an asset on the blockchain.
- Feeding: Adding new features or elements to an asset.
- Transformation: Solidifying a mature asset in structure.

Obviously, each of these tasks can change the state of an asset, and should be managed as a transaction and regulated with a contract.

According to the above analysis, we summarize requirements of the architecture design as follows:

1. Functionality
   User behavior must determine system states;
   Contracts must be used only to describe asset transactions;
   Business logic needs to be separated from contracts;
   ONMCA-based systems must implement basic DAM tasks.
2. Scalability
   Multiple consensus protocols should be allowed to coexist;
   No limit needs to be put on the number of chains created and operated by each user.

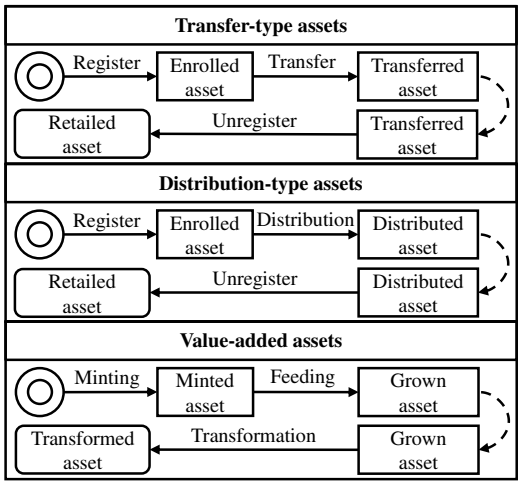

Fig. 1 On-chain lifecycle of different assets.

3. Customizability
   Blockchain data models must determine transaction modes;
   Consensus protocols must be customizable;
   Control interfaces must be customizable;
   Contract templates must be customizable;
   No limit needs to be put on the choice of programming language.
4. Security
   Each user should contribute at most one node to the blockchain network;
   ONMCA-based systems should be resistant to common attacks;
   Chains need to be isolated from each other.

4.2 Architecture Overview

4.2.1 Model Description

ONMCA aims to guide the designing of BDAMSs. Each asset holder can deploy a node on the open blockchain network of a BDAMS to manage his/her own assets in a customizable way. It is a composite architecture, which resembles a microservice-like architecture on the whole and a layered architecture on individual nodes, as shown in Figure 2.

Logically, ONMCA contains seven layers as shown in Figure 2. On the bottom is the network layer. The nodes of this layer come from users who want to join the DAM consortium. A dedicated node program will be installed on each node to run the entire BDAMS. The rest of the layers serve the design of the node program. Multi-chain layer

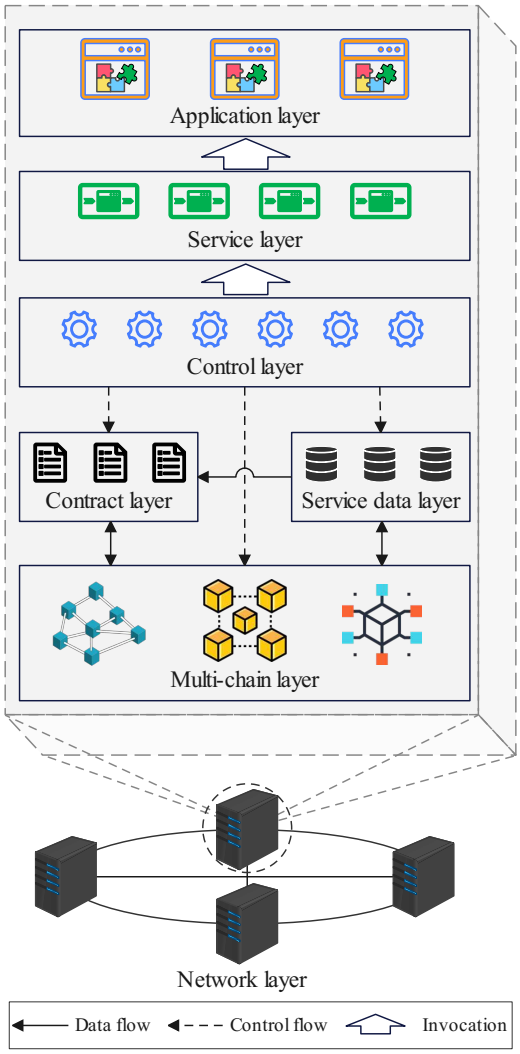

Fig. 2 Architecture model of ONMCA.

is the core of ONMCA. Multiple chains, which may be heterogeneous, are allowed to coexist in this layer. Among them, the contract chain, which is compulsory, is responsible for linking contract scripts stored in the contract layer. Other chains are called state chains and help to manage the lifecycles of particular types of digital assets. They are tailored to demand, but must be kept consistent with the data stored in the service data layer. The contract layer and service data layer are both databases. The former stores contracts about various asset transactions, and the latter stores the data generated during business logic execution. The control layer provides a group of interfaces that control the data and behavior of the lower layers, and these interfaces are invoked by the service layer to execute business logic. By invoking these interfaces reasonably, the service layer builds various services for the DApps in the application layer.

ONMCA-based BDAMSs have three types of default users, which are asset manager (AM), service provider (SP), and service consumer (SC). An AM adds a server that stores his/her digital assets to the blockchain network by running the node program. The AM may also be an SP who provides DAM services through the node program. An SC may be an AM when sharing assets with other AMs through the node program, but more commonly, he/she may just be an asset visitor who uses a personal device and works at the application layer as a temporary light node to enjoy DAM services provided by SPs.

4.2.2 Design Principles

The design of the node program is the key to developing an ONMCA-based BDAMS. Its workflow should conform to the following principles:

Principle 1: Customizable consensus. Reserve interfaces in the control layer to write or terminate a consensus protocol for each blockchain of the multi-chain layer. These protocols regulate the communication rules of nodes in the network layer.

Principle 2: Customizable data models. Reserve interfaces in the control layer to define or disable blockchain data models in the multi-chain layer. A data model contains a set of data structures, manipulations, and constraints.

Principle 3: Customizable contracts. Reserve interfaces in the control layer to create or invalidate contracts in the contract layer. A contract consists of counterparties, transaction objects, and contract terms, where the contract terms are executed in an event-triggered manner by calling interfaces of the control layer.

Principle 4: Customizable DAM services. Reserve interfaces in the control layer for programming various DAM services in the service layer. At the same time, these interfaces store the service data in the service data layer.

Principle 5: Customizable DAM tasks. Create customized applications in the application layer by invoking DAM services in the service layer. The styles of Client/Server and

Browser/Server are allowed to build applications for outside-system SCs.

Principle 6: Separation of on-chain operations and business logic. Let contracts take charge of on-chain operations and let interfaces in the control layer take over business logic.

4.2.3 State Transition

An ONMCA-based blockchain system can be defined as a quintuple $M = \{Q, R, E, q_0, F\}$, where $Q = \{q_0, q_1, ..., q_n\}$, $(n \geq 0)$ is a non-empty finite state set, $R = \{r_0, r_1, ..., r_m\}$, $(m \geq 0)$ the set of users' requests, $E = \{e \mid e : Q \times R \to Q\}$ the set of maps that executes users' requests and changes the states of the system, $q_0 \in Q$ the initial state of the system, and $F \subset Q$ the set of accepted states of the system.

A blockchain system runs by four steps: transaction initiation (TI), transaction distribution (TD), transaction verification (TV), and transaction packing (TP). Each step will push the blockchain system into a new state. Let $E_x \subset E$ be a set of maps that execute the tasks of step x, the following is an example of a system state transition maps:

$$
\begin{aligned}
E_{TI} &= \{submit\}\,; \\
E_{TD} &= \{broadcast, communicate\}\,; \\
E_{TV} &= \{submit\}\,; \\
E_{TP} &= \{approve, reject\}\,.
\end{aligned}
$$

The state transition diagram of M is shown in Figure 3.

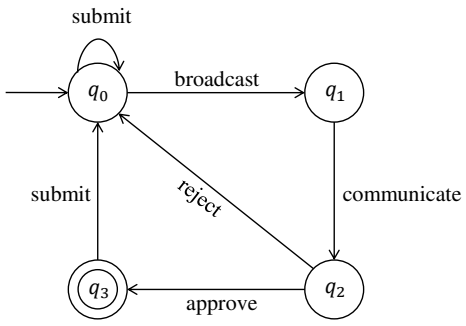

Fig. 3 State transition diagram of ONMCA

4.3 Architecture Composition

4.3.1 Components

Each layer has its own unique components as shown in Table 3. The following is an introduction to the main components of each layer.

Table 3 Components of ONMCA.

Layer	Components
Network layer	Node
Multi-chain layer	Transaction, Asset, Block, Blockchain
Contract layer	Database
Service data layer	Database
Control layer	TransControl, AssetControl, UserControl, ContrControl
Service layer	MetaService, TransService, AssetService, UserService, ContrService

1. Node
   Node components form the underlying network of a blockchain system. They can be server machines, personal computers, smartphones, tablets, and other networking devices that are capable of installing and running node programs. A node program should provide basic ability of communication and interfaces for deploying consensus protocols. A blockchain system requires a sufficient number of nodes that can store the entire blockchain data to keep working.
2. Transaction
   Transaction components are used to record the events of asset state changing. They may have different structures and manipulations depending on the types of digital assets. They are also atomic components that make up some other components, such as asset, block, and blockchain.
3. Asset
   Asset components may have different forms by designing. Generally, an asset can be composed of a group of transaction components related to this digital asset. These transactions are organized in some way, including chaining together. It is worth noting that the asset component differs from the real digital asset. The former represents the transaction history of the latter in the form of a data model.

4. Block
   Block components are the key to build a full blockchain. They may have various structures, manipulations, and different ways to organize transactions. The input of a block contains a set of transactions, and the output is a hash pointer to the block, which helps to attach the block to a blockchain.
5. Blockchain
   Blockchain components constitute the multi-chain layer. Connecting blocks together in some way will builds a blockchain. However, the connecting is not necessarily in the form of a chain. It depends on the specific DAM requirements.
6. Database
   Database components work across two layers, which are the contract layer and the service data layer. The database in contract layer serves as a repository for all contract scripts, and the database in service data layer is responsible for managing all the data generated from business logic.
7. TransControl
   TransControl components are middlewares that control the behavior of transaction components and provide interfaces for service layer to interact with the transaction components.
8. AssetControl
   AssetControl components are middlewares that control the behavior of asset components and provide interfaces for service layer to interact with the asset components.
9. UserControl
   UserControl components are middlewares that control users' information stored in the database of the service data layer. They also provide interfaces for users to interact with the system.
10. ContrControl
    ContrControl components are middlewares that manage user-defined contract scripts stored in the database of the contract layer. They also provide interfaces for users to customize their contract templates.
11. MetaService
    MetaService is a component that provides services to other components in the service layer. It offers basic libraries to other components, whereby these components can customize application programming interfaces (APIs) for the application layer.
12. TransService
    TransService components allow users to create, revoke, and verify transactions. Transaction verification is the key to keep a blockchain healthy. Function (1) verifies a transaction Tx, and if 0 is returned, Tx will be not recorded on-chain.

$$\{0,1\} \leftarrow Verify\,(Tx, paras)\,. \tag{1}$$

13. AssetService
    AssetService components are used for querying transactions of specific digital assets. Once receiving a user's request, the AssetService component will analyze the corresponding contract and then execute the services that the user requests according to the contract. Let U_i be the user, N_j the blockchain network, and C_k the contract that U_i signs. U_i can initiate a transaction Tx that operates on an asset $A_{U_i,N_j,m}$, where m is the system-assigned serial number of this asset. Function (2) is one of the terms of C_k and is used to generate Tx.

$$\begin{aligned} Tx = GenTx(paras, A_{U_i,N_j,m}, \\ pk_{from}, pk_{to}, C_k), \end{aligned} \tag{2}$$

    where $paras$ is a set of system parameters, pk_{from} the public key of the transaction initiator, and pk_{to} the public key of the transaction recipient.
14. UserService
    UserService components are responsible for user registration and authentication. They implement basic interfaces from the MetaService component, and maintain users' information stored in the database of the service data layer. They also bridge users to the system by calling interfaces provided by UserControl components. An SP will take charge of the authentication of user U_i. If the authentication passes, U_i will pick a secret key $sk_i \in Z_p^*$, and then compute the public key $pk_i = g^{sk_i}$ according to the Bilinear Map. The key pair (sk_i, pk_i) of U_i will come into play when U_i's digital assets are processed.
15. ContrService
    ContrService components provide interfaces

for users to create, modify, and revoke contracts. They have connections with UserService components and ContrControl components, and allow users to update contract scripts stored in the database of the contract layer. We predefined a series of contract templates that meet the basic DAM requirement for customization. These templates are saved as static members of ContrService. Besides, ContrService reserves specific interfaces for users to update these predefined templates or create new templates. An example of the contract template is shown in Figure 4.

Contract template ct_i

Parties:	$addr(pk_A)$, $addr(pk_B)$, …
Signatures:	sig_A, sig_B, …
Issuer:	The originator of the contract.
AssetNumber:	The system-assigned identifier of the asset.
AssetName:	The owner-assigned name of the asset.
AssetDigest:	The hash pointer of the asset.

Terms:	*Term_1*{
	deliver(
	AssetNumber,
	AssetDigest,
	$addr(pk_A)$,
	$addr(pk_B)$);
	}
	……
	Term_n{……}

Fig. 4 Contract template. pk_X, the public key of party X; sig_X, the signature of party X; $addr\,(pk_X)$, the address generated from pk_X.

Overall, the components in the multi-chain layer and the service layer should provide customization for different objects. In the multi-chain layer, the components allow for customizable data models. In the service layer, the components enable customizable business logic. For developers of ONMCA, they can develop BDAMSs with unique properties that meet specific DAM needs. At the same time, the users of ONMCA-based BDAMSs can design demand-driven business logic by combining or overriding the interfaces that the components provide.

4.3.2 Connections

Communication between components of ONMCA follows the service request framework shown in Figure 5, where $S = \{s_1, s_2, ..., s_n\}$ is a collection of services that users request, $F = \{f_1, f_2, ..., f_m\}$ the corresponding function calls of S, and $R = \{r_1, r_2, ...r_l\}$ a set of results of F.

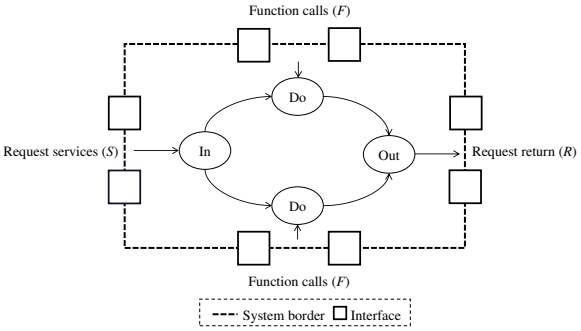

Fig. 5 Service request framework of ONMCA

This framework presents a three-step workflow: (1) a user request a service $s_i \in S\,(i \leq n)$ from the ONMCA-based system; (2) the system calls a function $f_j \in F\,(j \leq m)$ to deal with s_i; and (3) the system responds to the user with the result $r_k \in R\,(k \leq l)$ after f_j is finished.

Figure 6 illustrates the connections among the main components of ONMCA. Calling interface is the only way for components to interact with each other and the outside world. Interfaces of a component specify the services (messages, operations, and variables) that the component can provide. As shown in Figure 6, the MetaService component provides basic services to other service components, which invoke interfaces from the control layer to provide services to users. The data of the TransControl and AssetControl components comes from the multi-chain layer, and that of the ContrControl and UserControl comes from the service data layer. The provision of data is also achieved through interface calls. The asset, transaction, and database components are these service providers. They are also the building blocks of constructing a blockchain. Finally, the node components are the physical carriers of all other components.

4.3.3 Configuration

The configuration of ONMCA's components is set by parameters, which makes it easy for developers to customize a BDAMS. We use $Args_{cp}$ to

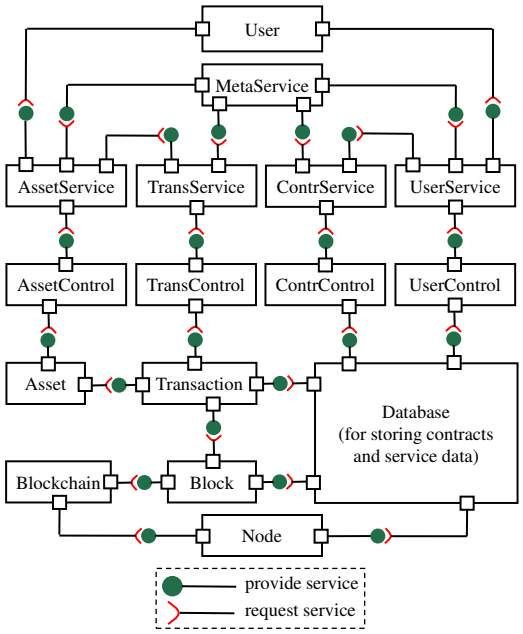

Fig. 6 Connections of ONMCA

denote the set of parameters that the configuration of component *cp* has, and describe the basic configuration of the components in each layer as follows.

1. $Args_{Node}$
   $Args_{Node}$ is used to turn a compute machine into a node of the blockchain system. The parameters in $Args_{Node}$ will be set when the node program is installed. The meanings of the parameters in $Args_{Node}$ are described as follows.
   NodeName is the unique identifier for the node.
   NodeType is the type of the node, such as light node, permanent node, etc.
   NodePosition is the geographic coordinate location of the node.
   NodeIP is the IP address of the node.
   RuntimeEnvironment is the software environment required for running the node, such as operation system, programming language, libraries, dependencies, etc.
   ComputeResources specifies the computing resources required for the node, such as CPU, memory, storage space, bandwidth, etc.
   MaxChains specifies the upper limit of the number of chains that can run simultaneously on the node.
   Concurrency represents the ability of the node to process requests concurrently, specifying the max number of requests.
   FaultTolerance specifies the fault tolerance mechanism of the node and how it handles failure scenarios.
   ConsensusProtocols specifies the consensus protocols allowed to run on the node.
2. $Args_{Transaction}$
   Parameters in $Args_{Transaction}$ are used to describe the structure and functions of a Trasanction component. Most of these parameters can be user-defined, which provides room for customization. However, we list the following types of parameters as non-optional ones to keep a transaction verifiable.
   TransParties describes the parties of a transaction. They are usually associated with uses' public keys for authenticability.
   TransContent is the transaction object that can be expressed numerically. Nevertheless, this parameter may just be a flag when the data structures in a blockchain have already been designed to express a transaction object. For example, the payment of one unit of BTC is described as an ownership transferring rather than a change in numbers.
   Signatures sets the digital signature of each party of the transaction.
   Timestamp records the exact date and time when the transaction took place.
   HashPointer is an unique identifier of the transaction. It is calculated by hashing all the key elements of the transaction.
3. $Args_{Asset}$
   Setting $Args_{Asset}$ has the similar manner as setting $Args_{Transaction}$. The followings are several distinct parameters.
   AssetNumber is the system-assigned identifier of the asset.
   RealAsset is bound to the real digital asset, and is the unique identifier of it.
   TailHash is the hash pointer of the latest transaction related to the asset. This parameter specifies the entry for asset tracing.
   AssetState represents the current state of the asset, such as available for sale, shareable, banned, revoked, etc.
4. $Args_{Block}$
   Setting $Args_{Block}$ has the similar manner as setting $Args_{Asset}$. The followings are several

distinct parameters.
BlockNumber is a number that indicates where the block is located in the current blockchain.
BlockStructure selects the data structure of the block from pre-set structures, which can be designed when the node program is developed.
BlockHash is the hash pointer of the block.

5. $Args_{Blockchain}$
The parameters in $Args_{Blockchain}$ are used to organize the blocks that have the same structure. The followings are non-optional ones of these parameters.
BlockchainID is the unique identifier of the blockchain.
BlockchainStructure selects the data structure of the blockchain from pre-set structures. The structure will determine how the blocks are organized into a blockchain.
BlockchainLength indicates the total number of blocks in the blockchain.
6. $Args_{Database}$
The parameters in $Args_{Database}$ provide constraints on database reading and writing. The following parameters are very important when the database component is accessed.
DatabasePosition indicates whether the database works at the contract layer or the service data layer.
DatabaseType indicates the type of the database. There are no restrictions on using relational or non-relational databases.
AccessMode specifies the way of accessing the database, including connection address, authentication information, and so on.
7. $Args_{*Control}$
The parameters for the components in the control layer are mainly used to normalize the way of calling interfaces of these components.
InterfaceNames sets a name for each interface.
InterfaceArgs declares a group of arguments for each interface.
Functionality describes the functionality of each interface.
8. $Args_{MetaService}$
The function of the MetaService component is to serve other components. Since the services to be provided depend on the specific DAM requirements, the parameters in $Args_{MetaService}$ are also highly customized.
9. $Args_{*Service}$
The parameters for the components in the service layer are mainly used to normalize the way that these components provide services. Note that the service here may refer to several related interfaces.
ServiceNames sets a name for each service.
ServiceArgs declares a group of arguments for each service.
Functionality describes the functionality of each service.

5 Evaluation

In this section, we demonstrate the effectiveness of ONMCA by examining how it meets DAM requirements.

5.1 Functionality Analysis

The requirements covered in this section correspond to the functionality requirements mentioned in Section 4.1.

Proposition 1. *User behavior determines system states.*

We describe the system response mechanism microscopically for giving a more clear perspective. We introduce the concept of TRIGGER that represents the transference of the system state. Trigger $t \Vdash [q, r]$ indicates that the system state will be turn to q when a certain request r is processed. According to Figure 3, we can define the key triggers as follows:

$$
\begin{aligned}
&t_s \Vdash [q_0, submit]\,,\\
&t_b \Vdash [q_1, broadcast]\,,\\
&t_c \Vdash [q_2, communicate]\,,\\
&t_a \Vdash [q_3, approve]\,,\\
&t_j \Vdash [q_0, reject]\,.
\end{aligned}
$$

First, a listener is deployed on each node to monitor the receipt of requests. When request *submit* is received, t_s will be triggered to turn the system state into q_0. After the system enters state q_0, a *broadcast* request will be issued. t_b of each node is triggered by receiving the

broadcast request. After that, each node will send a *communicate* request to other node, by which the system state will converge to q_1. When all nodes cooperate with each other to reach data consistency of the system, q_2 is achieved. During this process, t_c of each node will be triggered. The received transaction will be validated, and one of the two requests *approve* and *reject* will be issued after the validation. If *approve* is issued, t_a will be triggered and the system state will become q_3. Otherwise, t_j will be triggered and the system state will be turn back to q_0, which indicates a failure of state transition. When the system state is q_3, all nodes are waiting for a new *submit* request to finish this round of state transition successfully.

Definition 1. *An operation that can cause a complete change in the state of a system is called a transition operation (OprT), otherwise it is called a non-transition operation (OprNT).*

Definition 2. *The state transition caused by an OprT is called a global state transition, which is defined as*

$$q_i \leftarrow e\left(q_{i-1}, OprT\right), q_i \neq q_{i-1}, i \geq 1,$$

where q_i is the system state of number i, and e the map of state transition. On the contrary, OprNTs may shift the local state of the system, but will not cause a global state transition. This process is defined as

$$\begin{cases} q_i \leftarrow e\left(q_i, OprNT\right), i \geq 1 \\ l_k \leftarrow f\left(l_{k-1}, OprNT\right), l_k \neq l_{k-1}, q_i \rhd l_k, k \geq 1 \end{cases},$$

where l_k is a local state of the system, f a function that executes an OprNT, and $q_i \rhd l_k$ means q_i can tolerate the existence of l_k.

Theorem 1. *At least one OprT is triggered is a sufficient and necessary condition for a system state transition.*

Proof. Let Q be the pre-condition "*At least one OprT is triggered*", and P the post-condition "*A system state transition occurs*".

Proof of $Q \Rightarrow P$: Assume that the current system state is q_m, the user behavior is represented by a set of operations $O_n = \{Opr_0, Opr_1, ..., Opr_n\}$, $n \geq 0$, and there exists Opr_i $(i \leq n)$ such that Opr_i is an $OprT$. By Definition 2, it follows that Opr_i will definitely turn q_m into q_{m+1} under the action of e, where $q_m \neq q_{m+1}$, so $Q \Rightarrow P$ is proved.

Proof of $Q \Leftarrow P$: Assume that no $OprT$ is triggered. If there is a system state transition from q_m to q_{m+1}, then $q_{m+1} \leftarrow e\left(q_m, OprT\right)$ must hold by Definition 2. This contradicts the assumption. So $Q \Leftarrow P$ is proved.

The proof of the theorem is completed. □

Since the user behavior can be represented with a set of operations O_n, the system state will be determined depending on whether $\exists OprT \in O_n$ is true. Therefore, Proposition 1 is true.

Proposition 2. *Contracts can and only can be used to describe asset transactions.*

In ONMCA, the concept of CONTRACT is no longer the same as SMART CONTRACT, but back to its original meaning. A contract is a script signed in advance by the parties of a transaction and automatically executed in the blockchain system. Contracts are generated from templates, and each contract corresponds to an actual digital asset transaction. With this in mind, We define a contract as $C =< S, G, T >$, where S is a set of statements in the contract C, G a set of states of C, and T a set of triggers of C. Here, we also use the definition of TRIGGER mentioned at the beginning of this section. The difference is that the states in G are not about the system, but about the contract.

Definition 3. *For each trigger*

$$t = (g, S'), t \in T, g \in G, S' \subseteq S,$$

there exists

$$\alpha_t = (a_1, a_2, \cdots, a_i)^{\mathrm{T}}, a_i \in \{0, 1\}, i = |S'|,$$

such that

$$term_t = (s_1, s_2, \cdots, s_i) \times \alpha_t \neq 0, s_i \in S',$$

where $term_t$ is a term of C triggered by t. Let us say that α_t is a feature vector of t, and describes the state change of C that t will cause.

Let $\hat{S}_k^b$ and $\hat{S}_k^e$ be two sets of statements in $term_k$ that initiate and complete trigger k, respectively. $\hat{S}_k^b \cap \hat{S}_k^e = \varnothing$. We define two relationships between contract terms as follows.

Definition 4. *Mutual independence: An execution of $term_x$ does not cause the execution of $term_y$ and vice versa, which is denoted by*

$term_x \nleftrightarrow term_y$. It holds that $term_x \nleftrightarrow term_y$, if the following conditions are met:

$$\begin{cases} \hat{S}_x^e \cap \hat{S}_y^b \neq \hat{S}_x^e \\ \hat{S}_x^b \cap \hat{S}_y^e \neq \hat{S}_y^e \end{cases} .$$

Definition 5. *Strict progression: The completion of $term_x$ causes the execution of $term_y$, but not vice versa, which is denoted by $term_x \mapsto term_y$. It holds that $term_x \mapsto term_y$, if the following conditions are met:*

$$\begin{cases} \hat{S}_x^e \cap \hat{S}_y^b = \hat{S}_x^e \\ \hat{S}_x^b \cap \hat{S}_y^e \neq \hat{S}_y^e \\ \hat{S}_x^e \neq \hat{S}_y^b \end{cases} .$$

Theorem 2. *There is no looping between the terms of a contract.*

Proof. Assume that a third relationship exists between $term_x$ and $term_y$ ($x \neq y$), then $term_x \nleftrightarrow term_y$ and $term_x \mapsto term_y$ are not true at the same time. This means that only one condition might hold:

$$\begin{cases} \hat{S}_x^e \cap \hat{S}_y^b = \hat{S}_x^e \\ \hat{S}_x^b \cap \hat{S}_y^e = \hat{S}_y^e \\ \hat{S}_x^e = \hat{S}_y^b \end{cases} .$$

As $x \neq y$, i.e., $term_x \neq term_y$, there must have $\hat{S}_x^b \neq \hat{S}_y^e$, so there is no looping between $term_x$ and $term_y$.

This completes the proof. □

To sum up, the change of contract state can reflect the process of asset transaction, and at the same time, our design limits the contracts to a low level of description, without breaking the logic of asset transactions. So, Proposition 2 is true.

Proposition 3. *Business logic is separated from contracts.*

As shown in Figure 4, the contracts in ONMCA are not independently executable. They are like header files in C programming language, where the terms are descriptions and declarations to business logic. The components ContrControl and ContrService are responsible for connecting the declarations to the actual interfaces. Since these declarations are just identifiers rather than references, business logic is separated from contracts at the programming level. So, Proposition 3 holds.

Proposition 4. *ONMCA-based systems implement basic DAM tasks.*

As listed in Section 4.1, the basic DAM tasks include register, unregister, transfer, distribution, minting, feeding, and transformation. The transition of an asset's states during the on-chain lifecycle of the asset still follows the service request framework shown in Figure 5 and the state transition pattern shown in Figure 3. By Definition 1, we know that all of the DAM tasks shown in Figure 1 are $OprT$s. At the same time, a state transition of an asset will eventually result in a transaction of the asset, which means that the state transition can be described with a contract according to Proposition 2. Therefore, Proposition 4 holds.

5.2 Scalability Analysis

The requirements covered in this section correspond to the scalability requirements mentioned in Section 4.1.

Proposition 5. *Multiple consensus protocols are allowed to coexist in ONMCA.*

Definition 6. *Assume that the size of each message forwarded by a node is fixed. Then, the traffic of each node can be defined as the number of messages forwarded by that node.*

Definition 7. *The pressure of a node is defined as the ratio of the traffic of the node to the total traffic of the network. We use δ_n to denote the pressure of the node n.*

Definition 8. *Let N be the set of nodes in a blockchain network and $Cons$ the consensus protocol used in the network. The max node pressure denoted by Δ_{Cons} can be calculated as follows:*

$$\Delta_{Cons} = \max \delta_n, n \in N.$$

Proposition 5 is true if we can show that multiple consensus protocols coexist without significantly increasing the max node pressure. Without loss of generality, we select three common consensus protocols, Practical Byzantine Fault Tolerance (PBFT) [33], Raft [34], and Cascading Consensus Protocol (CCP) [35], to examine the change of the max node pressure.

Single PBFT. Assume that their is only PBFT in the blockchain network. Firstly, the master node sends a message to other nodes in the pre-preparation stage, and the number of communication for this process is $|N|-1$. Secondly, each node receiving the message will send it again to other nodes, and this needs to take $(|N|-1)^2$ communications. Thirdly, each node sends an acknowledgment message to other nodes in the commit stage, and $|N|(|N|-1)$ communications are made. Finally, all nodes reply the confirmation message to client, and this costs $|N|$ communications. Therefore, the total traffic of the network using single PBFT is calculated by Eq. (3).

$$\begin{aligned} Tr_{all} &= (|N|-1)+(|N|-1)^2+|N|(|N|-1)+|N| \\ &= 2|N|^2-|N|. \end{aligned} \tag{3}$$

Table 4 shows the traffic generated by each node during the above process. So, we can calculate the max node pressure of PBFT with Eq. (4).

$$\Delta_{PBFT}=\frac{Tr_{max}}{Tr_{all}}=\frac{2|N|-1}{2|N|^2-|N|}=\frac{1}{|N|}, \tag{4}$$

where Tr_{max} is the max traffic of a single node.

Table 4 Traffic in single PBFT.

Role	Pre-prepare	Prepare	Commit	Reply
master	$\|N\|-1$	0	$\|N\|-1$	1
others	0	$\|N\|-1$	$\|N\|-1$	1

Single CCP. Assume that their is only CCP in the blockchain network. The process of CCP can be split into two stages. In stage one, the sender sends message m_1 to the receiver, and then the receiver sends m_1 to a random node, which passes m_1 on in the same way to another node, and so on, until m_1 gets back to the sender again. In this process, except for the sender and receiver, all other nodes that receive m_1 reply to the receiver with message m_2. The times of communication of stage one are $2|N|-2$. In stage two, the sender sends message m_3 to a random node, which passes m_3 on to another node. All the nodes except for the sender and receiver do the same thing as they did in stage one, and m_3 finally reaches the receiver. The times of communication of stage two are $2|N|-3$. So, the total traffic of the network using single CCP is calculated by Eq. (5).

$$Tr_{all}=(2|N|-2)+(2|N|-3)=4|N|-5. \tag{5}$$

Table 5 shows the traffic generated by each node during the above process. So, we can calculate the max node pressure of CCP with Eq.(6).

Table 5 Traffic in single CCP.

Role	Stage one	Stage two
sender	1	1
receiver	1	0
others	2	2

$$\Delta_{CCP}=\frac{Tr_{max}}{Tr_{all}}=\frac{2}{4|N|-5}. \tag{6}$$

Single Raft. Assume that their is only Raft in the blockchain network. During the Log Replication phase, the leader sends a block log record to all followers with $|N|-1$ communication times. After receiving the leader's message, each follower sends a reply message to the leader, and the number of communications in this process is also $|N|-1$. When receiving more than half of reply messages from followers, the leader sends a message to the client and the followers for confirming the success, and this process costs $|N|$ times of communication. So, the total traffic of the network using single Raft is calculated by Eq. (7).

$$Tr_{all}=(|N|-1)+(|N|-1)+|N|=3|N|-2. \tag{7}$$

Table 6 shows the traffic generated by each node during the above process. So, we can calculate the max node pressure of Raft with Eq.(8).

Table 6 Traffic in single Raft.

Role	Log replication	Commit	Reply	Notify
leader	$\|N\|-1$	0	1	$\|N\|-1$
follower	0	1	0	0

$$\Delta_{Raft}=\frac{Tr_{max}}{Tr_{all}}=\frac{2|N|-1}{3|N|-2}. \tag{8}$$

PBFT+CCP+Raft. Assume that PBFT, CCP, and Raft coexist in the blockchain network and run simultaneously in one consensus cycle. According to Eq. (4), Eq. (6), and Eq. (8), the max node pressure with multiple consensus protocols can be calculated with Eq. (9).

$$\Delta_{Multi} = \frac{2|N|+2}{8|N|-7}. \tag{9}$$

Now, we use the difference limit to examine the max node pressure in the multi-consensus network as the number of nodes increases.

$$\begin{aligned} &\lim_{|N|\to\infty} (\Delta_{Multi} - \Delta_{PBFT}) \\ &= \lim_{|N|\to\infty} \frac{2|N|^2 - 6|N| + 7}{8|N|^2 - 7|N|} = \frac{1}{4}. \end{aligned} \tag{10}$$

$$\begin{aligned} &\lim_{|N|\to\infty} (\Delta_{Multi} - \Delta_{CCP}) \\ &= \lim_{|N|\to\infty} \frac{8|N|^2 - 18|N| + 4}{32|N|^2 - 68|N| + 35} = \frac{1}{4}. \end{aligned} \tag{11}$$

$$\begin{aligned} &\lim_{|N|\to\infty} (\Delta_{Multi} - \Delta_{Raft}) \\ &= \lim_{|N|\to\infty} \frac{-10|N|^2 + 24|N| - 11}{24|N|^2 - 37N + 14} = -\frac{5}{12}. \end{aligned} \tag{12}$$

Eq. (10), Eq. (11), and Eq. (12) show that, compared with a single consensus protocol, multiple consensus protocols will not cause excessive node pressure when the number of nodes increases. This proves Proposition 5.

Proposition 6. *No limit needs to be put on the number of chains created and operated by each user.*

Under the constraints of ONMCA, users access the underlying network by running a node program, which allows users to create and operate multiple chains locally. According to the design principles mentioned in Section 4.2.2, a user can use the node program to create and operate any number of chains without being interfered with by other users and without taking up other users' resources. These chains can employ different data models and consensus protocols. Based on Proposition 5, it is no problem that each chain has its own consensus protocol. So, Proposition 6 is true.

5.3 Customizability Analysis

The requirements covered in this section correspond to the customizability requirements mentioned in Section 4.1.

Proposition 7. *Blockchain data models determine transaction modes.*

The data structure is one of the key elements of a blockchain data model. ONMCA's customizability to multiple data assets comes from the flexible design of the transaction data structure (denoted by txn). Next, we focus on the transaction modes shown in Table 2 to examine how txns can describe different transaction modes.

To describe the transfer-type transaction mode, we can make the current transaction txn_{cur} connect to previous transaction txn_{pre} by adding the hash pointer of txn_{pre} as an element of txn_{cur}. Doing so will create a chain of transactions, which represents a digital asset, much like Bitcoin does. The key to this transaction data structure is that the asset owner must generate a signature $sig\left(hash\left(txn_{pre}\right)\right)$ for txn_{cur} to indicate the handing over of the asset ownership.

To describe the distribution-type transaction mode, we just need to slightly change the way of signing transfer-type transactions. The asset owner has to sign each transactions for each new user of the asset to keep the ownership from transferring.

To describe the value-added transaction mode, we can add a new element to store the new feature ftr that a new transaction txn contains, and then let the provider of the new feature offer a signature $sig\left(ftr\right)$ to txn to claim the new shared ownership of the asset.

Figure 7 shows the above different designs. Apparently, ONMCA allows blockchain data models to be customized to describe different transaction modes. So, Proposition 7 is true.

Proposition 8. *Consensus protocols, contract templates, and control interfaces are customizable in ONMCA.*

Based on the design principles we proposed in Section 4.2.2, it is mandated by design that a BDAMS following ONMCA can have customizable consensus protocols and contract templates. As we discussed in Section 4.3, users can deploy consensus protocols on different chains with the node component at the network layer. Meanwhile, users can create and modify contract templates

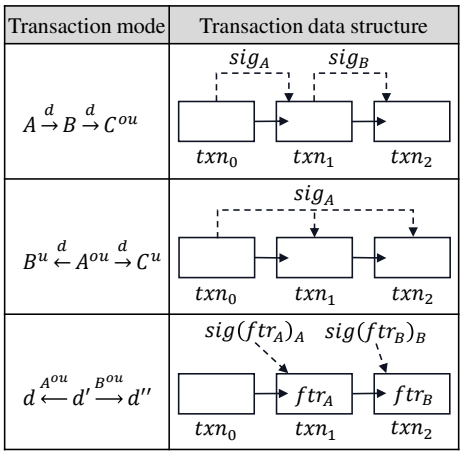

Fig. 7 Blockchain data models for different transaction modes.

by invoking interfaces of the ContrControl component at the control layer. In addition, what interfaces the components of the control layer can provide is also determined at the beginning of the development of the BDAMS. Thus, Proposition 8 is true.

Proposition 9. *There is no limit on the programming language in which ONMCA-based systems can be developed.*

The contracts in ONMCA are template-based, and the terms therein will be translated into interface calls, so there is no need to use a special contract language to implement these contracts. At the same time, all components are based on the service request framework, so any programming language that can provide the framework, such as Java, Python, C++, etc., can develop BDAMSs based on ONMCA. In sum, Proposition 9 is true.

5.4 Security Analysis

The requirements covered in this section correspond to the security requirements mentioned in Section 4.1.

Proposition 10. *Each user can contribute at most one node to the blockchain network.*

ONMCA provides UserService and UserControl components to ensure the uniqueness of each account in the target system. Furthermore, ONMCA is geared towards permission-ed environment, which means that we can administratively guarantee the uniqueness of the identities of network participants. Therefore, having each user contribute up to one node to the blockchain network in a unique identity is easy to achieve at both the network layer and the service layer. In other words, Proposition 10 is true.

Proposition 11. *ONMCA-based systems are able to resist common attacks.*

We analyze ONMCA's defense capabilities against common attacks faced by blockchain systems, including Sybil attack, relay attack, denial of service (DoS) attack, and contract vulnerability attack.

Resisting Sybil attacks. Sybil attack is a cyber attack that disrupts the balance of a system by creating multiple virtual identities, accounts, or nodes [36, 37]. If an attacker creates a sufficient number of fake identities, he/she may take control of the entire system by outvoting other honest nodes. By Proposition 10, however, it is stipulated that users can only contribute at most one node to the network and be authenticated both online and offline. So the success rate of Sybil attacks is extremely low.

Resisting relay attacks. A relay attack is a type of cyber attack where an attacker intercepts and relays communication between two legitimate parties [38, 39]. The attacker uses specialized devices to capture and transmit signals, allowing them to bypass security measures and gain unauthorized access. ONMCA allows designers to use session tokens to prevent this attack from happening. Once a transaction starts, the sender sends the receiver with a one-time token, which is used by the receiver to transform a secret. Suppose that an attacker has poached this secret and tries to initiate a new session. Then, the token of the new session will be changed, making the attacker's work meaningless. This design can be implemented by customizing the consensus protocols in the network layer. So ONMCA-based systems are able to resist relay attacks effectively.

Resisting DoS attacks. A malicious client may send a request of service without any effort, but the system has to use a lot of resources to process the request and respond to the client. This is a common vulnerability that DoS attackers usually exploit [40, 41]. In ONMCA-based systems, however, a user who conducts a DoS attack will certainly leave a trace online. The users entering the system must have been authenticated both online and offline. The evildoer will suffer high social penalties. At the same time,

the variables qualified for users to request can be restricted through setting the MetaService component. In addition, users are prohibited from interacting directly with the service data layer, as shown in Figure 6, preventing invalid requests from depleting storage resources.

Resisting contract vulnerability attacks. Smart contracts are essentially human-written computer code, so their flaws and vulnerabilities are inevitable [42, 43]. ONMCA's contracts are not smart contracts but contract templates. We can circumvent most of the risks with ONMCA by using contract templates that are certified by the market. Meanwhile, ONMCA's contracts do not involve business logic, so they are less likely to have code vulnerabilities that affect assets. Moreover, using contract templates not only reduces duplication of development, but also reduces the severe economic losses caused by smart contract vulnerabilities.

In sum, Proposition 11 is true.

Proposition 12. *Chains are isolated from each other.*

Chains in ONMCA may have different data models and consensus protocols, and each chain can have only one initiator. An initiator determines the intended use of a chain, and organizes other nodes to take part in this chain's consensus. The initiator operates all these processes by its own node program. Then, no one else will be able to change this chain's elements and characteristics. Also, the transactions with different data structures can only be packed into their own chains. The above setting can be regulated by the components of the control layer. So, Proposition 12 is true.

5.5 Solving Existing Architectural Issues

Table 7 shows how ONMCA addresses the four issues mentioned in Section 1.1.

Table 7 Issues addressing.

Issues	Propositions
Pseudo decentralization	Proposition 1, Proposition 10
Not asset-oriented	Proposition 2, Proposition 4, Proposition 6, Proposition 7
Contract dependency	Proposition 3, Proposition 7
High load of chains	Proposition 3, Proposition 12

6 Conclusion

The development of today's digital asset management systems needs the support of a more credible and customizable blockchain system architecture. Against this background, we propose ONMCA, which is a new architecture for building multi-chain BDAMS. Compared to existing architectures, ONMCA eliminate the key issues of DAM, including pseudo decentralization, not asset-oriented, contract dependency, and high load of chains. Therefore, ONMCA will greatly promote the engineering application of blockchain in DAM.

Some limitations of our work point out several future research directions. First, we have not discuss ONMCA-based software development yet, but it is important for ONMCA to come into use. So we leave this line of research as future work for public, especially for engineering researchers. Second, we did not discuss ONMCA at a full management level, whereas management schemes do need further study as they may determine the vitality of the architecture. Finally, ONMCA is still need more architecture patterns to verify its effectiveness in practice.

We hope that more scholars will be involved in the improvement and validation of ONMCA and that ONMCA will help more engineers and managers.

Acknowledgments. This paper is funded in part by the Natural Science Foundation of Hebei Province under grant No. F2023201032.

Declarations

Funding This paper is funded in part by the Natural Science Foundation of Hebei Province under grant No. F2023201032.
Conflict of interest All the authors declare that they have no confict of interests.
Data Availability Not applicable.
Ethics Approval Not applicable.
Consent to participate All the authors involved in this manuscript have agreed to participate in this submitted paper.
Consent for publication All the authors involved in this manuscript give full consent for publication of this submitted paper.

Availability of data and materials Not applicable.
Code availability Not applicable.
Authors' Contributions L.W. designed the research. W.Z. built the models and drafted the paper. L.Z. proved the propositions and helped organize the paper. H.L. performed the formal analysis. All authors reviewed the manuscript.

References

[1] S. Alam, M. Shuaib, W. Z. Khan, S. Garg, G. Kaddoum, M. S. Hossain, Y. B. Zikria, Blockchain-based initiatives: current state and challenges, Computer Networks 198 (2021) 108395.

[2] J. Zhang, S. Zhong, T. Wang, H.-C. Chao, J. Wang, Blockchain-based systems and applications: a survey, Journal of Internet Technology 21 (2020) 1–14.

[3] F. Casino, T. K. Dasaklis, C. Patsakis, A systematic literature review of blockchain-based applications: Current status, classification and open issues, Telematics and informatics 36 (2019) 55–81.

[4] G. T. Ho, Y. M. Tang, K. Y. Tsang, V. Tang, K. Y. Chau, A blockchain-based system to enhance aircraft parts traceability and trackability for inventory management, Expert Systems with Applications 179 (2021) 115101.

[5] Z. Yang, K. Yang, L. Lei, K. Zheng, V. C. Leung, Blockchain-based decentralized trust management in vehicular networks, IEEE internet of things journal 6 (2018) 1495–1505.

[6] D. D. F. Maesa, P. Mori, Blockchain 3.0 applications survey, Journal of Parallel and Distributed Computing 138 (2020) 99–114.

[7] D. Berdik, S. Otoum, N. Schmidt, D. Porter, Y. Jararweh, A survey on blockchain for information systems management and security, Information Processing & Management 58 (2021) 102397.

[8] Y. Lu, The blockchain: State-of-the-art and research challenges, Journal of Industrial Information Integration 15 (2019) 80–90.

[9] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, M. Imran, An overview on smart contracts: Challenges, advances and platforms, Future Generation Computer Systems 105 (2020) 475–491.

[10] R. B. Uriarte, H. Zhou, K. Kritikos, Z. Shi, Z. Zhao, R. De Nicola, Distributed service-level agreement management with smart contracts and blockchain, Concurrency and Computation: Practice and Experience 33 (2021) e5800.

[11] M. Taghavi, J. Bentahar, H. Otrok, K. Bakhtiyari, A blockchain-based model for cloud service quality monitoring, IEEE Transactions on Services Computing 13 (2019) 276–288.

[12] P. Zheng, Z. Wang, C.-H. Chen, L. P. Khoo, A survey of smart product-service systems: Key aspects, challenges and future perspectives, Advanced engineering informatics 42 (2019) 100973.

[13] M. Ozkaya, F. Erata, A survey on the practical use of uml for different software architecture viewpoints, Information and Software Technology 121 (2020) 106275.

[14] J. Beese, S. Aier, K. Haki, R. Winter, The impact of enterprise architecture management on information systems architecture complexity, European Journal of Information Systems (2022) 1–21.

[15] M. Wang, Y. Guo, C. Zhang, C. Wang, H. Huang, X. Jia, Medshare: A privacy-preserving medical data sharing system by using blockchain, IEEE Transactions on Services Computing (2021).

[16] B. Zaabar, O. Cheikhrouhou, F. Jamil, M. Ammi, M. Abid, Healthblock: A secure blockchain-based healthcare data management system, Computer Networks 200 (2021) 108500.

[17] R. Jabbar, N. Fetais, M. Krichen, K. Barkaoui, Blockchain technology for healthcare: Enhancing shared electronic health record interoperability and integrity, in: 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT), IEEE, 2020, pp. 310–317.

[18] Z. Wang, H. Jin, W. Dai, K.-K. R. Choo, D. Zou, Ethereum smart contract security research: survey and future research opportunities, Frontiers of Computer Science 15 (2021) 1–18.

[19] B. Alhijawi, M. A. Alrub, M. Al-Fayoumi, Generalized ethereum blockchain-based recommender system framework, Information Systems 111 (2023) 102113.

[20] S. Zhang, W. Li, X. Li, B. Liu, Authros: Secure data sharing among robot operating systems based on ethereum, in: 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2022, pp. 147–156.

[21] M. Zulfiqar, F. Tariq, M. U. Janjua, A. N. Mian, A. Qayyum, J. Qadir, F. Sher, M. Hassan, Ethreview: an ethereum-based product review system for mitigating rating frauds, Computers & Security 100 (2021) 102094.

[22] Q. Lu, A. Binh Tran, I. Weber, H. O'Connor, P. Rimba, X. Xu, M. Staples, L. Zhu, R. Jeffery, Integrated model-driven engineering of blockchain applications for business processes and asset management, Software: Practice and Experience 51 (2021) 1059–1079.

[23] J. Chen, Z. Lv, H. Song, Design of personnel big data management system based on blockchain, Future generation computer systems 101 (2019) 1122–1129.

[24] S. Hong, Y. Noh, J. Hwang, C. Park, Fabasset: Unique digital asset management system for hyperledger fabric, in: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2020, pp. 1269–1274.

[25] P. Rodrigo, J. Pouwelse, M. d. Vos, Unicon: Universal and scalable infrastructure for digital asset management, in: Proceedings of the 2nd International Workshop on Distributed Infrastructure for Common Good, 2021, pp. 5–10.

[26] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, W. Mann, Specification and analysis of system architecture using rapide, IEEE transactions on software engineering 21 (1995) 336–354.

[27] B. N. Silva, M. Khan, K. Han, Towards sustainable smart cities: A review of trends, architectures, components, and open challenges in smart cities, Sustainable cities and society 38 (2018) 697–713.

[28] C. Gehrmann, M. Gunnarsson, A digital twin based industrial automation and control system security architecture, IEEE Transactions on Industrial Informatics 16 (2019) 669–680.

[29] J. Kim, Y. Dvorkin, A p2p-dominant distribution system architecture, IEEE Transactions on Power Systems 35 (2019) 2716–2725.

[30] A. I. Sanka, M. Irfan, I. Huang, R. C. Cheung, A survey of breakthrough in blockchain technology: Adoptions, applications, challenges and future research, Computer communications 169 (2021) 179–201.

[31] B. K. Mohanta, D. Jena, S. S. Panda, S. Sobhanayak, Blockchain technology: A survey on applications and security privacy challenges, Internet of Things 8 (2019) 100107.

[32] T. Kulik, B. Dongol, P. G. Larsen, H. D. Macedo, S. Schneider, P. W. Tran-Jørgensen, J. Woodcock, A survey of practical formal methods for security, Formal Aspects of Computing 34 (2022) 1–39.

[33] M. Castro, B. Liskov, et al., Practical byzantine fault tolerance, in: OsDI, volume 99, 1999, pp. 173–186.

[34] D. Ongaro, J. Ousterhout, In search of an understandable consensus algorithm, in: 2014 USENIX annual technical conference (USENIX ATC 14), 2014, pp. 305–319.

[35] L. Wang, J. Liu, W. Liu, Staged data delivery protocol: A blockchain-based two-stage protocol for non-repudiation data delivery, Concurrency and Computation: Practice and Experience 33 (2021) e6240.

[36] M. Baza, M. Nabil, M. M. Mahmoud, N. Bewermeier, K. Fidan, W. Alasmary, M. Abdallah, Detecting sybil attacks using proofs of work and location in vanets, IEEE Transactions on Dependable and Secure Computing 19 (2020) 39–53.

[37] C. Pu, K.-K. R. Choo, Lightweight sybil attack detection in iot based on bloom filter and physical unclonable function, computers & security 113 (2022) 102541.

[38] N. V. Abhishek, A. Tandon, T. J. Lim, B. Sikdar, A glrt-based mechanism for detecting relay misbehavior in clustered iot networks, IEEE Transactions on Information Forensics and Security 15 (2019) 435–446.

[39] Y.-J. Tu, S. Piramuthu, On addressing rfid/nfc-based relay attacks: An overview, Decision Support Systems 129 (2020) 113194.

[40] Y. Pan, Y. Wu, H.-K. Lam, Security-based fuzzy control for nonlinear networked control systems with dos attacks via a resilient event-triggered scheme, IEEE Transactions on Fuzzy Systems 30 (2022) 4359–4368.

[41] L. F. Eliyan, R. Di Pietro, Dos and ddos attacks in software defined networks: A survey of existing solutions and research challenges, Future Generation Computer Systems 122 (2021) 149–171.

[42] M. Zhang, X. Zhang, Y. Zhang, Z. Lin, {TXSPECTOR}: Uncovering attacks in ethereum from transactions, in: 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 2775–2792.

[43] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu, X. Wang, Combining graph neural networks with expert knowledge for smart contract vulnerability detection, IEEE Transactions on Knowledge and Data Engineering (2021).