frontiers | Frontiers in Blockchain

TYPE Original Research
PUBLISHED 27 July 2023
DOI 10.3389/fbloc.2023.1006058

OPEN ACCESS

EDITED BY
Roman Vitenberg,
University of Oslo, Norway

REVIEWED BY
Victoria L. Lemieux,
University of British Columbia, Canada
Chhagan Lal,
Delft University of Technology, Netherlands

*CORRESPONDENCE
Peng Zhang,
peng.zhang@vanderbilt.edu

RECEIVED 28 July 2022
ACCEPTED 07 July 2023
PUBLISHED 27 July 2023

CITATION
Zhang P, Kelley A, Schmidt DC and White J (2023), Design pattern recommendations for building decentralized healthcare applications. *Front. Blockchain* 6:1006058. doi: 10.3389/fbloc.2023.1006058

Design pattern recommendations for building decentralized healthcare applications

Peng Zhang[1,2]*, Adair Kelley[1], Douglas C. Schmidt[1] and Jules White[1]

[1]Department of Computer Science, School of Engineering, Vanderbilt University, Nashville, TN, United States, [2]Data Science Institute, Vanderbilt University, Nashville, TN, United States

Blockchain and distributed ledger technologies (DLT) are emerging decentralized infrastructures touted by researchers to improve existing systems that have been limited by centralized governance and proprietary control. These technologies have shown continued success in sustaining the operational models of modern cryptocurrencies and decentralized finance applications (DeFi). These applications has incentivized growing discussions in their potential applications and adoption in other sectors such as healthcare, which has a high demand for data liquidity and interoperability. Despite the increasing research efforts in adopting blockchain and DLT in healthcare with conceptual designs and prototypes, a major research gap exists in literature: there is a lack of design recommendations that discuss concrete architectural styles and domain-specific considerations that are necessary for implementing health data exchange systems based on these technologies. This paper aims to address this gap in research by introducing a collection of design patterns for constructing blockchain and DLT-based healthcare systems that support secure and scalable data sharing. Our approach adapts traditional software patterns and proposes novel patterns that take into account both the technical requirements specific to healthcare systems and the implications of these requirements on naive blockchain-based solutions.

KEYWORDS

blockchain technology, distributed ledger technology, software engineering, design patterns, smart contracts, smart contract security and vulnerability, healthcare, data sharing

1 Introduction

Blockchain and distributed ledger technologies have successfully been implemented in managing transactions of digital assets through cryptographic currencies (cryptocurrency) in a decentralized manner, with the most prevalent being Bitcoin Nakamoto (2008) and Ethereum (Buterin, 2014). These decentralized technologies are a new paradigm that is fundamentally supported by mature concepts from computer science and mathematics. They differ from traditional infrastructures that have placed many constraints on system services and capabilities due to centralization. To achieve decentralization, a certain amount of key information must become transparent and thus immutable to a certain degree, which enabled "trustless" exchanges of Bitcoin-like cryptocurrencies Blundell-Wignall (2014) that directly involve both parties of a transaction without the need for a trusted middleman or third party. The revolutionary fundamentals underlying blockchain have sparked significant interest from technologists and domain experts to explore meaningful services across various industries, such as finance, healthcare, transactive energy, and the food industry.

Ethereum blockchain has become a mainstream platform today for developing applications as it has extended the capabilities of cryptocurrency-based blockchains to enable programmability and near Turing-complete computations via "smart contracts" (Buterin, 2014). Smart contracts are similar to a typical software program in that they have internal data and supported instructions (program code) to directly control or manipulate the data to facilitate exchanges and redistributions of digital assets. The instructions define rules and agreements established between involved parties in advance to produce deterministic outputs. The successful implementations of programmable smart contracts by Ethereum has promoted the development of decentralized apps (DApps) Johnston et al. (2014), which are autonomously operated applications that interact with cryptography-protected data stored on the blockchain and maintain the records of transactions also on-chain. DApps enable end users to directly interact with the blockchain and/or access relevant data on-chain.

Blockchain and smart contracts have been explored widely by researchers in healthcare to address interoperability challenges DeSalvo and Galvez (2015); Das (2017) among use cases. Interoperability is defined as the ability for different information technology systems and applications to communicate, exchange data, and effectively digest the exchanged information Geraci et al. (1991). Healthcare authorities and experts have attempted to improve healthcare interoperability for decades Richesson and Nadkarni (2011) in order to provide more continuous and consistent medical services, including but not limited to delivering patient data across multiple care practices securely and reliably, facilitating effective medical communications between providers, and accurately managing medical devices and promptly delivering their outputs to the appropriate patients or providers Lesh et al. (2007). Despite the growing interest in creating blockchain-based healthcare systems, the current literature on the concrete design recommendations deserves greater attention for applying blockchain technology to address domain-specific challenges in healthcare.

This research paper addresses the need for software design patterns tailored to the unique challenges of creating healthcare DApps. The paper offers two key contributions: firstly, a summary of five key domain challenges faced by blockchain technologies—evolvability, on-chain storage requirements, data privacy, communication scalability, and data authorization. Secondly, a set of design patterns is presented, including Layered Ring, Guarded Update, Contract Manager, Database Connector, Database Proxy, Entity Registry, Tokenized Exchange, and Publisher-Subscriber. Code examples and healthcare use cases are provided to illustrate how these patterns can effectively address the challenges associated with blockchain-based healthcare systems. The intended audience for this paper is health information technology (IT) system architects and developers seeking to implement blockchain and related technologies in their system design. The importance of our contributions lies in the potential impact they can have on software engineering practices and outcomes for the blockchain community. Utilizing design patterns can lead to more robust and adaptable systems, ultimately improving overall software quality. This can benefit developers, architects, and researchers in designing and maintaining complex systems. Furthermore, our research contributes to the theoretical understanding of blockchain-oriented software engineering principles by exploring the relationship between healthcare-specific requirements and design patterns.

In software engineering practice, design patterns offer general and reusable solutions to recurring problems. They allow software engineers to communicate using well-known and well-understood vocabulary for interactions in the software Shvets (2015). By documenting a collection of reoccurring blockchain-specific patterns that take into account domain-specific requirements, this work can assist the target audience to more quickly and effectively adopt this technology and create robust solutions with it in the healthcare domain. Design patterns play a particularly crucial role in blockchain development, as blockchain technology is still relatively new, and developers face unique challenges in building decentralized applications that can scale, remain secure, and interact seamlessly with other parts of the ecosystem. By using design patterns, developers can avoid common pitfalls, reduce development time, and create robust and reliable applications that can help drive the widespread adoption of blockchain technology in domain-specific applications.

Not applying design patterns can have negative consequences for software systems–software systems are more prone to becoming rigid and difficult to maintain, resulting in increased costs and effort. Additionally, neglecting important aspects such as scalability can limit the ability to leverage emerging technologies or adapt to increasing user needs, potentially leading to diminished competitiveness. Furthermore, insufficient attention to evolvability and design patterns can result in brittle systems that are prone to errors, security vulnerabilities, and performance degradation, ultimately compromising software quality. By investigating and integrating those important aspects in the proposed design patterns, we aim to provide practical insights and guidance for software practitioners and researchers, ultimately improving the robustness, adaptability, and overall quality of software systems.

The remainder of this paper is organized as follows: Section 2 provides an overview of blockchain technology and the Ethereum implementation; Section 3 summarizes existing research related to this work; Section 4 presents our proposed pattern collection in detail with example code and their use cases in healthcare applications; and Section 5 concludes the paper and summarizes future work on applying blockchain and related technologies in the healthcare domain.

2 Key concepts of blockchain technology

This section provides an overview of the general concept behind blockchain technology in addition to the open-source Ethereum implementation that supports smart contracts, which enable computation and facilitate the development of decentralized apps beyond cryptocurrencies. Solidity, the programming language for writing smart contracts in Ethereum and the basis of our pattern collection discussed in Section 4 will also be introduced.

2.1 Blockchain concepts

In essence, a blockchain is a decentralized, replicated, and continually reconciled ledger that maintains an append-only list of ordered transactions grouped into blocks, as shown in Figure 1.

Transactions with timestamps are recorded in the blockchain and distributed to all network nodes to provide transparency of the

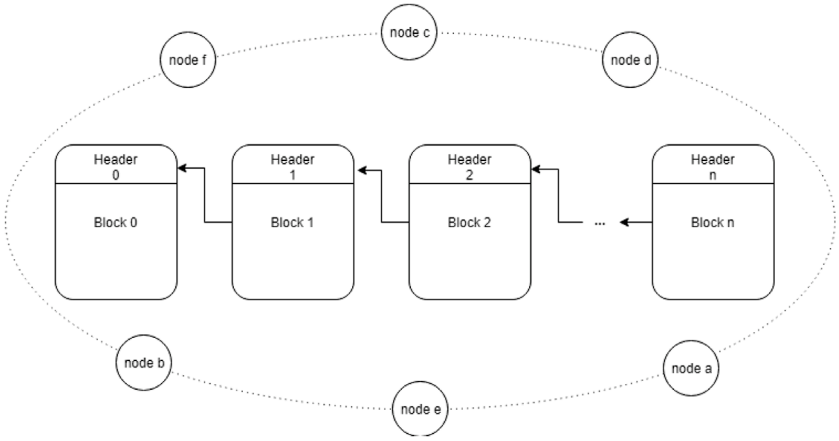

FIGURE 1
Blockchain structure: A shared, append-only list of ordered transactions grouped into blocks.

exchanges of digital data, such as cryptocurrencies. Only one block may be added to the blockchain at a time following a mathematical verification (based on cryptography) to ensure that it is in sequence from the previous block and that it contains all valid transactions. Network nodes are incentivized to verify transactions and blocks via a process called mining, during which they will gain some financial reward for their work Nakamoto (2008). This mechanism requires that 1) information in the blockchain is transparent so that anyone easily verifying that a specific transaction occurred at a particular point in time, 2) validated blocks are tamper-proof, which prevents existing transaction history from being altered, and 3) all blocks of transactions are replicated to each network node, which makes the network resilient to a single point of failure Hub (2017).

The Bitcoin blockchain serves as a public ledger that facilitates the direct exchange of its native token between individual users that is secured with cryptography. The Bitcoin blockchain primarily focuses on token transactions but is less suitable for other types of data exchange with higher complexity. To provide a more flexible framework, Ethereum was created as an alternative, general-purpose blockchain that enables programmability and thus more sophisticated computation via the support of smart contracts (Buterin, 2014).

Ethereum is a decentralized computing system that uses a near Turing-complete Ethereum Virtual Machine (EVM) and a native token ETH to power the network. EVM enables the creation and execution of smart contracts that can store data of different structures and codify operations on the data. Ethereum enforces a payment policy in terms of "gas" for creating, storing, and executing smart contracts. This policy provides both a financial incentive for network nodes to verify and execute valid transactions and a financial disincentive against malicious attacks on the network. In addition, there is a global maximum gas limit defined by the Ethereum protocol and a sender-specified gas limit that indicates the max gas amount that the sender is willing to pay. If gas spent during the execution of a transaction exceeds either of these two limits, computation will be stopped, and the sender still has to pay for the performed computation. This protocol protects senders from completely running out of funds and also further deters malicious attacks and abuse, such as distributed denial of service attacks in the network or hostile infinite loops in smart contract code (Buterin, 2014).

2.2 Overview of consensus mechanisms

In addition to Proof of Work, there are other consensus mechanisms that may be more fitting in blockchain-based healthcare applications. One popular alternative is Proof of Stake (PoS), which is based on the concept of stakeholder consensus E. Napoletano (2022). In PoS, instead of miners competing to solve a mathematical puzzle, network nodes (or "validators") are chosen to create the next block based on the amount of cryptocurrency they have "staked" as collateral. This creates an economic incentive for validators to act honestly, as their stake is at risk if they act maliciously. PoS is considered to be more energy-efficient than PoW, as it does not require significant computational power to maintain the network. Ethereum aims to migrate from PoW to PoS as part of the Ethereum 2.0 upgrade, which aims to improve the scalability, security and energy efficiency of the network Rene Millman (2020).

Another consensus mechanism that has gained popularity in recent years is Tendermint consensus. Tendermint is a protocol for secure, Byzantine fault-tolerant (BFT) consensus in decentralized networks GMBH, 2023a (accessed on 01-13-2023a). Tendermint consensus uses a two-step process: first, validators propose and pre-vote on the next block, and then they commit and vote on the block. This creates a more efficient and fault-tolerant consensus mechanism compared to PoW or PoS. Tendermint is also more flexible than other consensus mechanisms as it can be used in any blockchain platform and it can be configured to use different consensus algorithms. Tendermint is most widely used as the consensus mechanism in the Cosmos ecosystem GMBH, 2023b (accessed on 01-13-2023b), where it enables the creation of independent, app-specific chains that can interoperate with one another. An app-chain may be the best approach for such a healthcare application as it can be customized to fit the needs of the healthcare industry and can provide higher scalability, security, and privacy than a general-purpose blockchain[1].

1 App-chains are not explored in-depth in the context of this paper, though will be one direction of future work.

2.3 Overview of solidity

Solidity is an object-oriented language and is designed primarily for developing smart contracts in Ethereum Foundation (2015a). A Solidity class is realized through a "contract," which is an object prototype (some code template) stored on-chain. Just like an object-oriented class can be instantiated into a concrete object at runtime that can then be executed, a contract may be instantiated into a useable "smart contract account" (SCA) by a transaction or a function call from another contract. When a contract is instantiated, it is assigned a unique address that is similar to a reference or pointer in C/C++-like languages. The contract can then be referenced and its functions invoked using the address. A smart contract can also define state variables to store data and functions that interact with the data. Although one contract can be instantiated into many SCAs, it should be treated as a singleton (i.e., an object that has only one instance of) to avoid undesired behavior and storage overhead. A common practice is to store the address of an instantiated contract in a static location, such as a configuration file or a database Dourlens (2017).

Solidity also supports multiple inheritance and polymorphism Ethereum.io (2017). When a contract inherits from one or more other contracts, a new contract instance is created by copying all the base contracts code into the child contract prototype. An abstract contract in Solidity can declare function headers but without concrete implementations, which means that it cannot be compiled into an SCA but can be used as a base contract. In this paper, we focus the pattern discussions based on the use of Solidity.

3 Related work

Although relatively few studies focus on realizing software patterns in blockchains, some relate to healthcare blockchain solutions and design principles in this space. This section gives an overview of related research on design principles and recommended practices for developing blockchain-based apps.

Porru et al. (2017) highlighted evident challenges in state-of-the-art blockchain-oriented software development by analyzing open-source software repositories and addressed future directions for developing blockchain-based software. Their work focused on macro-level design principles such as improving collaboration, integrating effective testing, and evaluations of adopting the most appropriate software architecture. Bartoletti et al. Bartoletti and Pompianu (2017) surveyed the usage of smart contracts and identified nine common software patterns shared by the studied contracts, e.g., using "oracles" to interface between contracts and external services and creating "polls" to vote on some question. These patterns summarize the most frequent solutions to handle some repeated scenarios. Xu et al. (2018) presented a collection of patterns in categories of communication with off-chain channels, data management, security, and structural design that are applicable to general blockchain-based applications. Similarly, Six et al. (2022) created a taxonomy of existing patterns for designing DApps as a path to build out an ontology for performing semantic queries.

A number of attacks on Ethereum smart contracts have been reported, including the infamous DAO attack Siegel (2016) where $50 million worth of Ether was stolen and the critical Parity wallet hack Palladino (2017) that incurred in $30 million worth of Ether being exploited. Atzei et al. (2017) surveyed existing attacks on Solidity smart contracts with code snippets showing related vulnerabilities. Meanwhile, the blockchain community also compiled a number of software patterns and anti-patterns targeting Solidity programming around cryptocurrency transactions in order to maximize the security of Ethereum smart contract design ConsenSys (2018). More recently, Moreno et al. (2019) proposed a security pattern focusing on the use of blockchain in big data systems. Relatedly, Ellervee et al. (2017) described a comprehensive reference model for blockchain-based systems using software architecture concepts.

Recent surveys summarized existing applications of blockchain technologies in the healthcare sector and provided a positive outlook on their potential. De Aguiar et al. (2020) discussed the potential applications of blockchain technology in healthcare, focusing on privacy, safety, and the sharing of health information. The study highlighted that blockchain has the potential to guarantee privacy by using techniques such as data immutability, attribute-based encryption, and zero-knowledge proof. It compared different consensus protocols and suggested that Practical Byzantine Fault Tolerance (PBFT) is more suitable for healthcare applications due to its lower computational costs and permissioned nature. Chukwu et al. Chukwu and Garg (2020) presented functional use cases of blockchain in data sharing, access control, audit, distributed computing, data storage, and data aggregation, with applications in various health domains such as HIV aids, Cancer, Clinical trials, Insurance, and more. Their analysis highlighted scalability issues, low performance, and high costs as significant challenges in implementing blockchain in healthcare. In the survey conducted by Arbabi et al. (2022), state-of-the-art efforts in utilizing blockchain-based solutions in healthcare were classified and analyzed, focusing on interactions between healthcare entities, functional components of healthcare storage systems, challenges in the healthcare domain, and benefits derived from blockchain technology. It also discussed compliance with privacy regulations such as GDPR and HIPAA.

Many research and engineering ideas have been proposed to apply blockchain technology in healthcare, and implementation attempts are underway Azaria et al. (2016); Peterson et al. (2016); Porru et al. (2017); Bartoletti and Pompianu (2017). Prior research efforts have provided a number of design recommendations for implementing Solidity smart contracts involving general-purpose decentralized applications. Few published studies, however, have addressed software design considerations needed to implement blockchain-based healthcare apps effectively with concrete examples. While it is crucial to understand the fundamental properties of blockchains and design patterns that are applicable when developing general solutions based on blockchain, it is crucial to apply them properly so that healthcare-specific requirements are satisfied in these designs as well. Even though a subset of principles from prior work may be relevant to the healthcare space, a systematic approach to documenting appropriate design practice that specifically targets technical challenges in healthcare is still essential.

4 A pattern collection for designing blockchain-based healthcare applications

The US Office of the *National Coordination for Health Information Technology* (ONC) defined the basic technical

requirements for achieving interoperability ONC (2014). Based on these domain-specific requirements, the study in Zhang et al. (2017a) outlined five aspects of relevant interoperability challenges faced by blockchain-based apps:

1. Evolvability: Maintaining evolvability while minimizing integration complexity: unlike traditionally centralized applications, data, changes to data, and its change history persist on-chain–due to the immutable nature of blockchain, it is computationally infeasible to revert them in a decentralized environment. At the same time, systems are subject to updates or upgrades required by clinical workflow or healthcare regulations. Balancing the need for change while avoiding a complete redesign of previous system component requires a good design.
2. On-Chain Storage: Minimizing data storage requirements on the blockchain: healthcare applications are expected to serve thousands to millions of users, which, if run on centralized systems, are much easier to scale. When data exchanges are brought to decentralized networks, how to minimize the storage overhead while maintaining necessary, traceable histories needed to support interoperability is another key design consideration.
3. On-Chain Privacy: Balancing data storage with privacy concerns: encryption is a modern approach applied to secure sensitive data–healthcare applications use encryption to guard personal health information. As computing resources become substantially cheaper and more powerful, existing encryption algorithms are yet to stand the test of time. Centralized systems face serious consequences when data privacy is breached, but when data is replicated across multiple locations, it will be more challenging to trace the source of the breach.
4. Scalable Communication: Tracking relevant health changes across large patient populations at-scale: according to a recent study, physicians on average spend 5.5 h documenting and reviewing electronic health records (EHR) of their patients for every 8 h of scheduled patient time Melnick et al. (2021). When patients visit multiple healthcare providers, there will be new information for each provider to review. A decentralized system should allow timely delivery and communication of changes in patient notes.
5. Security: Preventing unintended software loopholes and safeguarding on-chain data: the implication of blockchain's decentralized nature is that system is at higher risks of unintended loopholes and unauthorized data access, as information is exposed to a much wider and, if not carefully designed, uncontrolled audience, and security issues cannot be immediately rectified. In recent years, software loopholes that existed in several major blockchain-based crytocurrency services have been exploited by attackers, causing significant financial losses to users and the service providers Atzei et al. (2017). To prevent similar predicament from happening to healthcare services hosted in blockchain-based infrastructures, security risks must be recognized in the early stage of system designs.

This section presents a pattern collection for creating healthcare DApps that address these major challenges. We applied three different approaches for developing this collection. First, we extracted a subset of patterns using commonality and variability analysis Coplien et al. (1998) by reviewing a number of verified smart contract source code from Etherscan.io Etherescan to capture common portions repeatedly used across various contracts and/or supporting library contracts. This included contracts that resulted in attacks to produce our recommendations for safer smart contract design. Second, based on design practice and lessons learned from prior research Zhang et al. (2017b, 2018a, 2017c, 2018b), we proposed several patterns that target healthcare requirements. Third, we reviewed and adapted several design principles that are widely accepted as general system design recommendations to our pattern collection with blockchain-focused design considerations. Additionally, due to the growing popularity of Solidity and attacks that have occurred to public smart contracts, the Ethereum community has captured a number of Solidity code patterns for preventing similar attacks. Although those code patterns were almost exclusively targeting cryptocurrency or other apps with financial incentives, we identified one code pattern that would be particularly critical in a healthcare system.

The remainder of this section applies a pattern form variant to motivate and show how our pattern collection aids in designing blockchain-based healthcare applications. In particular, we present eight design patterns—LAYERED RING, GUARDED UPDATE, CONTRACT MANAGER, DATABASE CONNECTOR, DATABASE PROXY, ENTITY REGISTRY, TOKENIZED EXCHANGE, and PUBLISHER-SUBSCRIBER Gamma et al. (1995); Buschmann et al. (2007) in the context of the permission-less Ethereum blockchain. We describe key healthcare challenges these patterns resolve in the blockchain context and detail their structure and composition[2].

Table 1 provides an overview of the pattern collection, showing how the patterns relate to healthcare-specific challenges described at the beginning of this section and what specific sub-challenge each pattern aims to solve.

Each of the patterns in this collection is discussed in depth below.

4.1 A decentralized infrastructure for health data sharing systems

4.1.1 Design problem faced by DApps for healthcare use cases

Healthcare data is known to be fragmented across heterogeneous healthcare data warehouses managed by large healthcare organizations, private practices, and, more recently, mobile health app providers Ajami and Bagheri-Tadi (2013); Zhang et al. (2018a). Despite the adoption of certified EHRs or other data vendors that provide direct data exchange between providers within the same network, impediments for healthcare providers and researchers to access those heterogeneous data silos still exist.

4.1.2 Solution → apply the Layered Ring pattern to define a decentralized base architecture for a health data sharing system

The emerging blockchain technology that supports decentralized data storage and executable code via smart

2 Naturally, there are other patterns relevant in this domain, which will be the focus of future work.

TABLE 1 Overview of proposed pattern collection for designing blockchain-based healthcare apps.

Pattern	Targeted Category	Specific challenge to solve
Layered Ring	Evolvability	Defining the basic architecture of data sharing system
Guarded Update	Evolvability & Security	Preventing unexpected reentrancy attacks
Contract Manager	Evolvability	Splitting data from logic to ensure data availability via clean separation of concerns
Database Connector	On-Chain Storage	Ensuring on-chain storage scalability and interoperability with standardized and minimal interfaces to off-chain storage
Database Proxy	On-Chain Privacy	Providing an additional layer of security by performing lightweight tasks before permitting access to database connectors
Entity Registry	On-Chain Storage	Managing healthcare entities on-chain and other types of common data at scale
Tokenized Exchange	On-Chain Privacy	Authorizing access to data storage and maintaining verifiable access logs
Publisher-Subscriber	Scalable Communication & Security	Providing user notifications when events of interest occur across the decentralized system

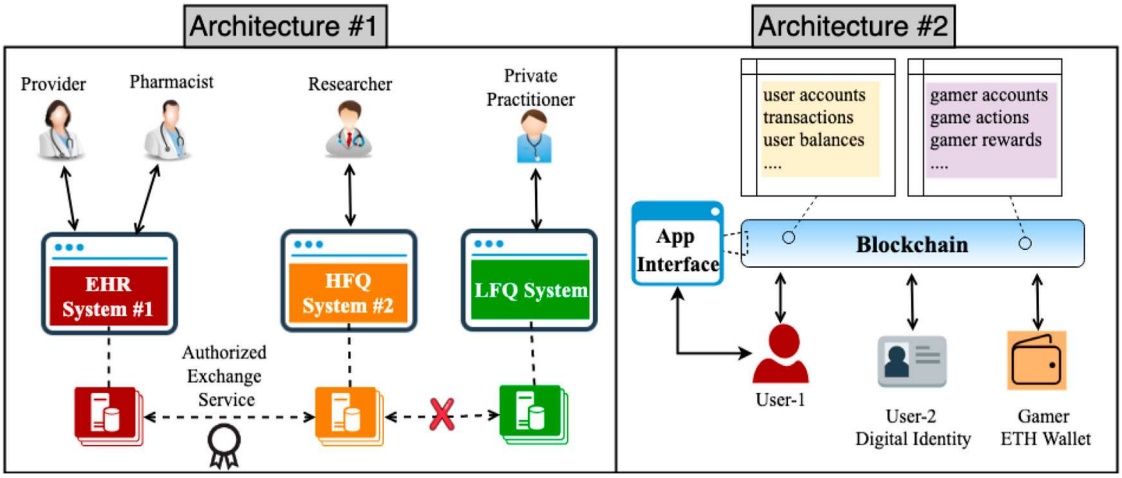

FIGURE 2
Comparing the current state of traditionally centralized healthcare architecture with that of popular blockchain-based use cases.

contracts, with Ethereum (Buterin, 2014) being the most popular, has presented itself as a potential infrastructure to connect existing healthcare data silos Peter B. Nichol (2016); Broderson et al. (2016); Dubovitskaya et al. (2017). It has successfully maintained tamper-proof cryptocurrency transactions between Internet users worldwide Nakamoto (2008); Cap (2023) and managed verifiable collectibles and rewards from cryptogaming like CryptoKitties and artwork through Non-Fungible Tokens Kugler (2021).

Figure 2 compares the high-level architecture of modern healthcare data sharing systems with blockchain-based digital asset management platforms. In this figure, the bottom layer in *Architecture 1* represents heterogeneously represented objects, such as siloed healthcare data sources, low-frequency, high-fidelity clinical data (LFQ) captured by trusted sources, and high-frequency, low-fidelity data (HFQ) generated by patients or wearable and mobile devices Zhang et al. (2018c), and geographically dispersed Internet users in *Architecture 2*. Data sources generated by healthcare professionals via diverse, centralized EHR systems on the left may or may not inter-operate, depending on if an authorized exchange service is available between the data sources. Whereas on the right, data (like identifiers of users or asset owners) and data requests flow into and out of the same service implemented on the decentralized ledger and is thus widely accessible, with or without a common user interface.

Architecture 1 can be modified to achieve more decentralization by applying the basic pattern of *Architecture 2* in Figure 2, except that a user interface is needed for healthcare users who need not concern themselves with acquiring advanced knowledge about blockchain or smart contract function invocation. In fact, most dApps, such as CryptoKitties (a cryptogame for collecting and breeding digial cats) CryptoKitties, 2023, Fomo3D (a gambling

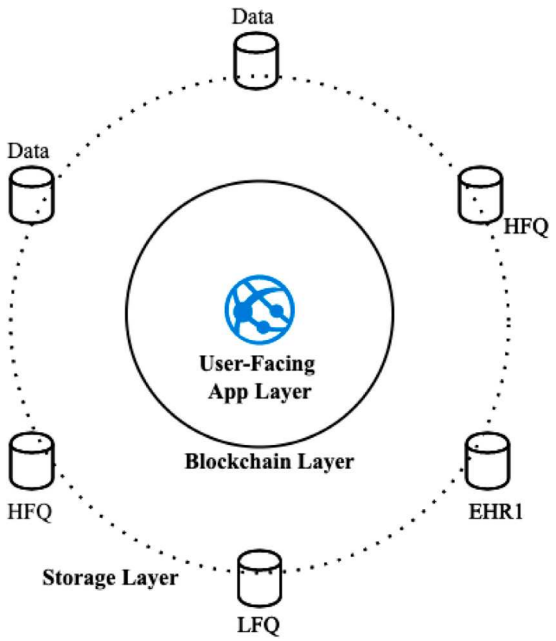

FIGURE 3
Structure of the Layered Ring pattern that defines the base architecture of the data sharing system.

game for winning cryptocurrency lotteries) FOMO3D, 2023, and IDEX (a cryptocurrency trading platform) IDEX - Decentralized Ethereum Asset Exchange, 2018, implement a user-friendly interface that encapsulates the blockchain component to provide a familiar user experience as any other centralized applications.

Figure 3 presents the first pattern in the collection, LAYERED RING, which is generalized from *Architecture 2* above with a high-level view to illustrate the scale of involved entities in each layer. The outermost layer is a *Storage Layer* composed of a large number of data sources, each maintained by its owner (e.g., a private practitioner, a healthcare organization, or a 3rd party HFQ data provider). The middle *Blockchain Layer* connects data sources from the outer layer and is to be maintained by key stakeholders or mid-to large-size healthcare organizations in a federated environment. The innermost *User-Facing App Layer* provides a convenient interface for interacting with data and operations defined in the blockchain. It is also the most centralized piece of the system because a user-facing app, whether a mobile or web-based one, is usually hosted in a centralized server. However, the app server can be designed to log activities and events directly to the shared ledger and preserve the system's overall transparency.

TheLayered Ring pattern is also a variant of the ENTERPRISE SERVICE BUS (ESB) pattern Zdun et al. (2006); Fernandez (2013), which provides a common data model and a messaging infrastructure to allow various systems to communicate through a shared set of interfaces. In LAYERED RING, the *Blockchain* layer acts as the messaging bus that provides services to the rest of the components. However, unlike the ESB, the interface of *Layered Ring* focuses only on the structural and syntactic level of the exchanged data and does not itself create a common ontology for interpreting the semantics of shared data, which is an entirely separate and complex topic being researched by domain experts. The infrastructure also requires careful design to ensure that stewards or managing nodes of the network are decentralized.

4.2 Preventing reentrancy attack in the blockchain

4.2.1 Design problem faced by DApps for healthcare use cases

The replicated and decentralized nature of blockchain makes DApps prone to mistakes that are exploited by attackers. This raises security concerns regarding the use of this technology in data-sensitive industries such as healthcare that requires compliance to strict security and privacy regulations. An infamous example of prior attacks on the blockchain was the DAO attack in 2016 Siegel (2016) in which a reentrancy bug was discovered and exploited that caused at the time worth $30 million of Ethereum being stolen. Even though the immutability and decentralization properties of blockchain technology can provide tremendous value to the direct exchange of digital information, without proper design decisions made prior to deploying a system on-chain could yield destructive consequences for healthcare users.

4.2.2 Solution → apply the Guarded Update pattern to prevent unexpected reentrancy attacks

We deem attack prevention as the utmost important design consideration in the development cycle of a blockchain-based healthcare system, so we propose the use of this pattern as a mechanism against reentrancy attack early on during the

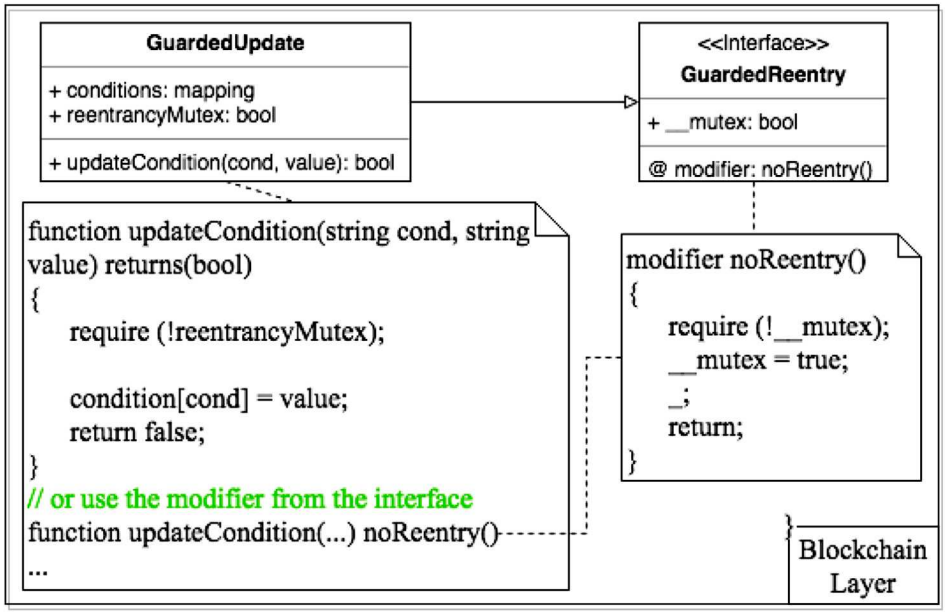

FIGURE 4
Structure and example solidity code snippet of Guarded Update pattern to prevent reentrancy attacks on-chain.

development cycle in order to help design the rest of the system wherever this pattern may apply. A simplified reentrancy bug that affected the DAO app is shown in the code snippet below:

```
...
contract VulnerableContract {
  mapping (address => uint( public balances;

  ... // code to deposit funds

  function withdraw() {
      // vunerable code
      if (!msg.sender.call.value(balances[msg.sender])()) {
          throw;
      }
      balances[msg.sender] = 0;
  }
}

contract ExploitVulnerableContract {
    VulnerableContract public vc;

    ... // code to reference VulnerableContract object

    function() payable {
      vc.withdraw(); // This is the default fallback that can recursively
                     // execute the vulnerable code above
    }
}
...
```

The *withdraw()* function in *VulnerableContract* sets the caller's balance after checking if the asset transfer to the caller (*msg.sender*) is successful. The attack in *ExploitVulnerableContract* exploits this vulnerability by calling the *withdraw()* function in a fallback function that is executed by the *call.value()* method, creating recursions that bypass the statement on the line of code that sets the user balance after the vulnerable statement returns ConsenSys (2018).

Although the reentrancy bug primarily targets digital assets such as cryptocurrencies that hold financial values, this bug could also affect system designs for healthcare functions if prevention is not implemented in advance. As a key pattern in the collection, GUARDED UPDATE prevents reentrancy attacks by ensuring atomic update to critical data in the system. The structure and code examples of this pattern appears in Figure 4[3]. As shown in the figure, a guarding conditional flag *reentrancyMutex* is used to control operations on protected state variable(s) (i.e., *conditions*). Once the variable(s) has been modified, the guarding condition is reset to the initial state to permit other memory contexts to operate the guarded data. Another, more systematic way to achieve this is to create a modifier in a Solidity interface contract, which can then be included in the declaration header of functions in other contracts.

Protecting atomic updates to state variables in the smart contracts prevents serious reentrancy attacks to occur, however, one major drawback is that atomic executions may both slow down runtime performance and increase transaction costs of the system, particularly in a decentralized network.

Functions that require an additional call through an intermediary contract can also be protected using a modified GUARDED UPDATE pattern. A situation where this is required is when the smart contract requires a call that casts an address of a contract to a contract instance. It should be noted that when making the call from the casted contract, the *msg.sender* value no longer contains the actual caller but rather the contract making the call itself. If the destination of the call passes through an intermediary contract and is destined for the original contract,

3 The code examples are based on https://github.com/o0ragman0o/ReentryProtected.

it may be necessary to cache the *_data* field and validate that it was not altered in between calls.

An example of an implementation for such a modifier can be found below.

```
...
uint256 private pendingInternalRecipientId;
modifier cachedInternalTransfer(bytes memory _data) {
    // check that data contains a uint256 value
    if (_data.length == 32) {
        require(pendingInternalRecipientId == 0, "Reentrant call");
        assembly {
            sstore(pendingInternalRecipientId.slot, mload(add(_data, add(0x20, 0))))
        }
    }
    _;
    pendingInternalRecipientId = 0;
}
...
```

In the above example, *pendingInternalRecipientId* is the ID of a token to which another token may be transferred. This is part of an implementation for composable tokens, where tokens can be owners of tokens in addition to standard addressed owners. Within the assembly code, *_data* is the memory address of the start of the array data. The first 32 bytes (0 × 20 bytes) are reserved for the length of the array, which needs to be stepped over in order to retrieve the first value. The array elements are numbered consecutively starting at 0 and each occupies one 32 byte word. The *mload* function retrieves the value at a specific point in the *_data*, stored in memory. Solidity's *sstore* loads the data from the second argument into the slot for the *pendingInternalRecipientId*. As in GUARDED UPDATE, this implementation also protects against reentrant calls on line 6 by ensuring that *pendingInternalRecipientId* is set to the 0 address before entering the modified function at line 11. Following the function call, *pendingInternalRecipientId* is set to 0 once more to reset the reentrancy check on line 12.

```
...
function safeTransferChildFrom(
    address _to,
    address _childContract,
    uint256 _childTokenId,
    uint256 _amount,
    bytes memory _data
) public override cachedInternalTransfer(_data) {
    ...
    ERC1155(_childContract).safeTransferFrom(address(this), _to,
          _childTokenId, _amount, _data);
    ...
}

function onERC1155Received(
    address _operator,
    address _from,
    uint256 _childTokenId,
    uint256 _amount,
    bytes memory _data
) virtual external override returns (bytes4) {
    require(_data.length == 32, "data must contain the unique uint256
      tokenId to transfer the child token to");

    // load the recipient tokenId
    uint256 recipientTokenId;
    assembly {recipientTokenId := calldataload(msg.data.length - 32)}

    if (_from != address(0)) {
        require(
            pendingInternalRecipientId == recipientTokenId,
            "Recipient was not validated before transfer."
        );
        ...
    }
    ...
    return this.onERC1155Received.selector;
}
...
```

In the above code sample, the *safeTransferChildFrom* method has an additional argument deviating from the standard *safeTransferFrom* from the OpenZeppelin ERC1155 standard contract interface OpenZeppelin, 2023a (accessed on 01-16-2023), which represents a child contract on which a *safeTransferFrom* call needs to be made. As mentioned above, the *_data* argument must be verifiably unaltered when the function has a callback to the original contract to ensure the original intention of the call is honored. By including the *cachedInternalTransfer* modifier, the address to which the child token is transferred is cached until the function is fully completed. If the argument *_to* is the address of the original contract, *onERC1155Received* is called because of the requirement of the ERC1155 contract standards by OpenZeppelin (2023b) (accessed on 01-16-2023).

On the invocation of *onERC1155Received*, the *recipientTokenId* from the *_data* argument is parsed and subsequently compared to the *pendinInternalRecipientId* cached in the *safeTransferChildFrom* call. This validates that the *_data*'s recipient address has not been altered and that no reentrancy has occurred.

4.3 Separating data from logic via a manager contract

4.3.1 Design problem faced by DApps for healthcare use cases

The inherent immutability of blockchains provides non-repudiation of data operations and transactions, but it can also become a major bottleneck to data liquidity. On the one hand, immutability is helpful for achieving interoperability in a healthcare environment as it makes data objects (whether it is a reference pointer to a remote data repository or an authorization request that grants a provider access to healthcare data) on the blockchain always available, even when one of the key maintainers of the network becomes unavailable. On the other hand, without a loosely-coupled design that focuses on clean separation of data and logic, immutability leads to difficulties during system upgrades. Data that is exchanged in healthcare systems includes not only information shared across various healthcare participants but also meta data related to the system or the most up-to-date list of network users. Logic is operation on the data (read, creation, update, or removal) or an event that occurs when a predefined condition is met.

4.3.2 Solution → apply the Contract Manager pattern to separate data from logic to ensure data availability

The CONTRACT MANAGER pattern aims to address the separation of data and logic via a *permanent storage* structure, which has been described in ConsenSys (2018). Figure 5 presents the structure of this pattern with sample interfaces and code snippets.

Permanent storage maintains one or more data fields used throughout the system with getter and setter functions for each data field. This ensures that all meta data needed by the system (such as the version or address information of any smart contract dependencies and other data structures shared across different smart contracts) remains readable even when logic contracts are

outdated. Additionally, CONTRACT MANAGER stores a *Contract Repository* of meta data that describes versions of the system (including but not limited to the addresses of the latest logic contract components and history contract addresses). To ensure upgradeability of the system, CONTRACT MANAGER also defines access privilege of smart contracts by allowing the original owner of the storage contract to configure an access group for delegating or revoking some or all rights of data access and operations to other members to prevent data locking.

One drawback of CONTRACT MANAGER is that all other logic contracts would need to access this contract for versioning checks and data queries. An alternative design is to leverage the AUTHORIZER pattern Fernandez (2013) along with fine-grained authorization models such as role-based access control models Sandhu et al. (1996) or access matrix Sandhu and Samarati (1994) to separate the definition of access rules, further decoupling the rules from the storage component. OpenZeppelin provides such an implementation of role-based access control models as a standard for Ethereum-based smart contracts OpenZeppelin (2023a).

4.4 Standardized on-chain interfaces to off-chain storage access

4.4.1 Design problem faced by DApps for healthcare use cases

EHR systems have served the U.S. healthcare for decades and have accumulated enormous amounts of valuable medical records that either exist in legacy systems or in more modern certified EHRs. Despite there being significant efforts to create compatible data sharing features, centralized systems still present a major barrier to interoperability. For decentralized systems, the large scale and stringent privacy requirements of healthcare data also present considerable challenges: 1) replicated copies of data make storing encrypted or hashed versions of the actual data on the blockchain vulnerable to attacks and 2) the prevalence of existing EHR systems make it impractical to replace or completely duplicate their functionality with decentralized systems. The design question now becomes how to bridge the gap between legacy central systems and the new paradigm of decentralized services that facilitate interoperability.

4.4.2 Solution → apply the Database Connector pattern promote interoperability via standardized and scalable interfaces to existing off-chain (centralized) systems

Figure 6 presents the composition of the DATABASE CONNECTOR pattern. The *Database Connector* component defines a standardized interface between the blockchain and storage layers. The interface provides an abstraction of the heterogeneous health data silos (e.g., EHR or other LFQ databases and HFQ data) to present only a minimal set of information about each data source to the blockchain layer. As shown in Figure 6, the interface may only need to capture meta information of a data source (such as name of owner or description) and then provide reference pointers to the original data source and a verifiable credential from the data source owner that proofs integrity of the data. *Database Connector* is also closely associated with the DATABASE PROXY pattern (discussed next in Section 4.5) that uses a *Connector Handler* component in the blockchain layer to provide data access to the connector.

The main benefits of DATABASE CONNECTOR are 1) it provides on-chain scalability that allows efficient sharing of the necessary connectors and 2) it offers a standardized interface that unifies the on-chain representations of off-chain databases. The drawback is the additional implementations that are required for creating standardized connectors to existing databases.

4.5 Security checking before accessing off-chain storage

4.5.1 Design problem faced by DApps for healthcare use cases

Patients in healthcare systems are not represented uniformly, so data sharing between providers often involve exchanging personally identifying information. As aforementioned, the replicated nature of blockchain is not suitable for sharing such sensitive information as any data and its transaction history stored on-chain are available to all network managers in an immutable and verifiable way. For financial applications focusing on verifying that a transfer of an asset indeed took place, these properties are critical. When the objective is to store data with an intricate structure and a large scale, it is important to understand how these properties will impact the use case.

For example, immutability makes it difficult for anyone, including data owners, to modify or remove the data change history from the blockchain. However, certain scenarios, such as when a security flaw is found, when a medical error is discovered in the data, or when data standards are updated, may demand change to data or data history over time. In these situations, the immutable nature of blockchains creates a fundamental tension that must be resolved between the need to present data to providers with integrity and the flexibility needed to change data when patient data privacy is at risk or when mistakes in the data need to be rectified.

4.5.2 Solution → apply the Database Proxy pattern to provide an additional layer of security by performing lightweight tasks before permitting access to database connectors

Database Proxy is akin to the traditional Proxy pattern Gamma et al. (1995) with a focus that is unique to a DApp design. To reduce computational costs on-chain, the *Database Proxy* interface 1) defines a lightweight representation as a placeholder of the original data object and 2) encodes necessary lightweight security checks and auditing tasks until retrieval of the original data object is required. It is worth noting that protected health information should only exist as an off-chain data object for which a proxy is created. This is so that any regulatory or security checks (which is much more rigorous) defined by HIPAA or other privacy standards, are performed off-chain.

Figure 7 illustrates the structure of DATABASE PROXY pattern and its interaction with the *Database Connector* object described previously in Section 4.4.

Database Proxy is an interface that maintains a reference to a *Connector Handler* object and forwards read/write requests to the

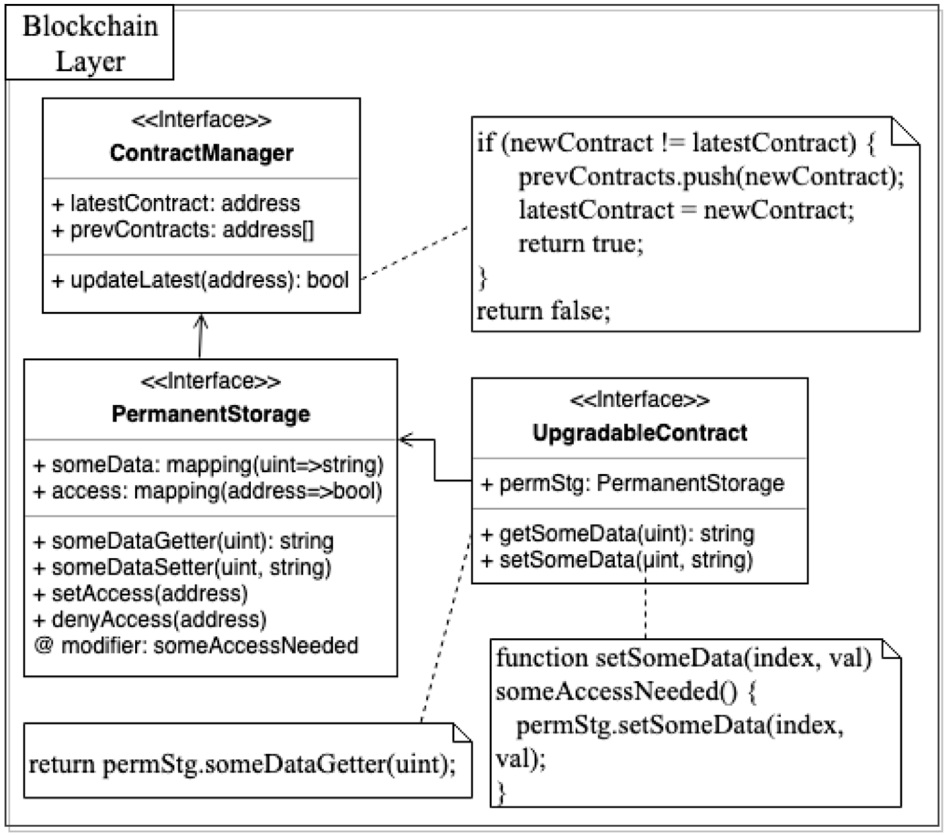

FIGURE 5
Structure and example solidity code of Contract Manager pattern for maintaining key meta-data on-chain.

appropriate *Database Connector* component for data access from the storage layer of the system. Each request through the *Connector Handler* is then logged on-chain, making the request history transparent to the blockchain network for verification against data corruption or unauthorized access. This design encapsulates lower-level implementation variations of the proxified contract. When the *Database Connector* that contains more heavyweight implementations is updated with a new storage configuration (e.g., when a data source has been introduced a new management system that requires some change in its *Database Connector* abstraction layer), the interface of the proxy contract remains unaffected.

Similar to the original PROXY pattern Gamma et al. (1995), a proxy object can perform lightweight housekeeping tasks, such as security checks of administrative access and auditing tasks that log existing data requests, on commonly used metadata stored in its internal states before retrieving the actual data. This component follows the same interface as the real object and can execute the original data object's function implementations as needed. It provides an additional layer for securing access to the real data object. However, *Database Proxy* may cause disparate behavior when the real object is accessed directly by some other component in the system while the proxy surrogate is accessed by others. It also creates an additional level of indirection for accessing actual data objects. This pattern complements the *Entity Registry* pattern when applied to the digital identity management use case. The user accounts from the registry can each have a proxy to the complete user data object. As the original data object builds up, its proxy contract stays unchanged. If the proxy is linked with an identifier along with the complete data object, it can also serve as a mechanism to retrieve stolen or lost identity, as implemented by an earlier version of the UPort identity system Lundkvist et al. (2017).

4.6 Managing healthcare entities and common data on-chain at scale

4.6.1 Design problem faced by DApps for healthcare use cases

Decentralization can only be achieved when digital assets and their transaction histories are shared with every network manager, which implies intensive storage requirements. For a DApp to serve healthcare use cases well, it should minimize on-chain storage burden yet still enable data sharing among participants.

Suppose a DApp stores some encrypted patient billing data on-chain. Billing data typically includes detailed patient insurance

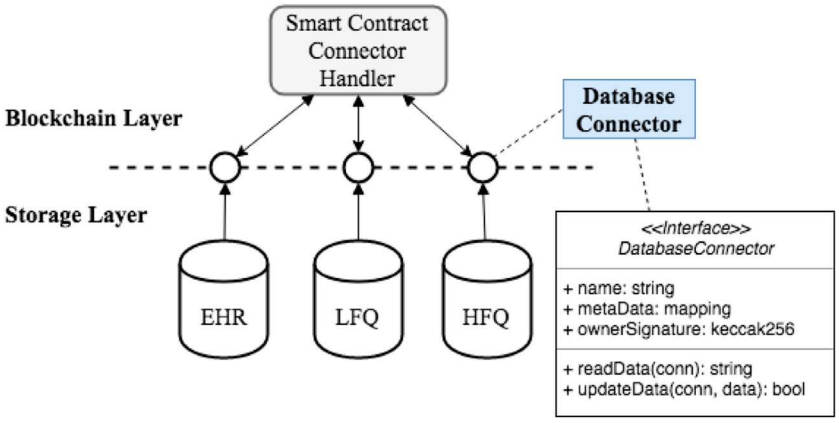

FIGURE 6
Structure of the Database Connector pattern used to standardize on-chain Interfaces to off-chain storage access.

information, such as their policy number, insurance contact information, coverage details, and other aspects needed by providers to bill for services. This implies that millions of records being replicated on all nodes of the network.

Realistically, most patients are covered by a relatively small subset of insurance groups (particularly in comparison to the total number of patients, e.g., each insurer may cover 10,000s or 100,000s of patients). Therefore, a substantial amount of intrinsic, non-varying information is common across patients that can be reused and shared, such as details on what procedures are covered by an insurance policy. To bill for a service, however, this common intrinsic information must be combined with extrinsic information (such as the patient's policy number) that is specific to each patient. A good design consideration is to create an on-chain data structure to capture such common data to reduce replication overhead while providing access to the complete data objects on demand.

4.6.2 Solution → apply the Entity Registry pattern for managing healthcare entities and common data on-chain at scale

As shown in Figure 8, the ENTITY REGISTRY mimics the traditional FLYWEIGHT pattern Gamma et al. (1995) with a factory Gamma et al. (1993) object to help manage healthcare entities on-chain. In particular, *getEntity* uses a factory to create entity objects and maintain references (addresses) to created Entity objects in a common smart contract (i.e., *Entity Registry*). It internalizes common data across a number of *Entity*'s *data* field while externalizing varying data storage in entity-specific contracts (such as *Patient* or *Provider* entity). Using references (i.e., addresses) to entity-specific contracts stored in the *registry*, combined extrinsic and intrinsic data can be retrieved upon request to return a complete dataset.

Applying this pattern to the earlier scenario, shared, encrypted insurance information is only stored once in the *registry* instead of being repeated stored in all patient accounts. Varying, patient-specific billing information is maintained in corresponding patient-specific entity contracts. The registry can also maintain a look up table (or a mapping) between unique entity identifiers and the referencing addresses of already deployed entity contracts to prevent account duplication. To retrieve complete insurance and billing information of a particular patient, clients need only invoke a function call from the registry with the patient identifier to obtain the combined intrinsic and extrinsic data object.

Entity Registry intends to provide more efficient management of large volumes of objects (such as user accounts in the example above). It minimizes redundancy in similar objects by maximizing data and operation sharing. Particularly in the insurance example, if common insurance policy details are not extracted from each patient's contract, the cost to change a policy detail will be immense–it will require rewriting a huge number of impacted contracts. Data sharing with flyweight registry helps minimize the cost to change the common state in objects stored on-chain. However, the application of this pattern alone cannot ensure integrity of the data being exchanged because it exposes only reference information for retrieving complete data objects for security and privacy reasons. It would rely on an off-chain or a 3rd-party oracle service Xu et al. (2016) to certify the integrity of the data either via hashing functions or other data verification protocols.

This pattern is particularly suitable for creating a standardized digital identity management system for healthcare participants that have varying roles. For instance, common data of participants includes identifiers, role type, role description, which can be stored in an ENTITY REGISTRY. Whereas specific data structures unique to each role type can be implemented in their respective contract classes or stored in off-chain locations, which can then be referenced in the registry. The registry in this case would serve as a global directory where identifiers are used for looking up specific information if given the access.

4.7 Securing and recording data access

4.7.1 Design problem faced by DApps for healthcare use cases

Smart contracts are powerful for automating executions of predefined agreements directly between involved entities. They

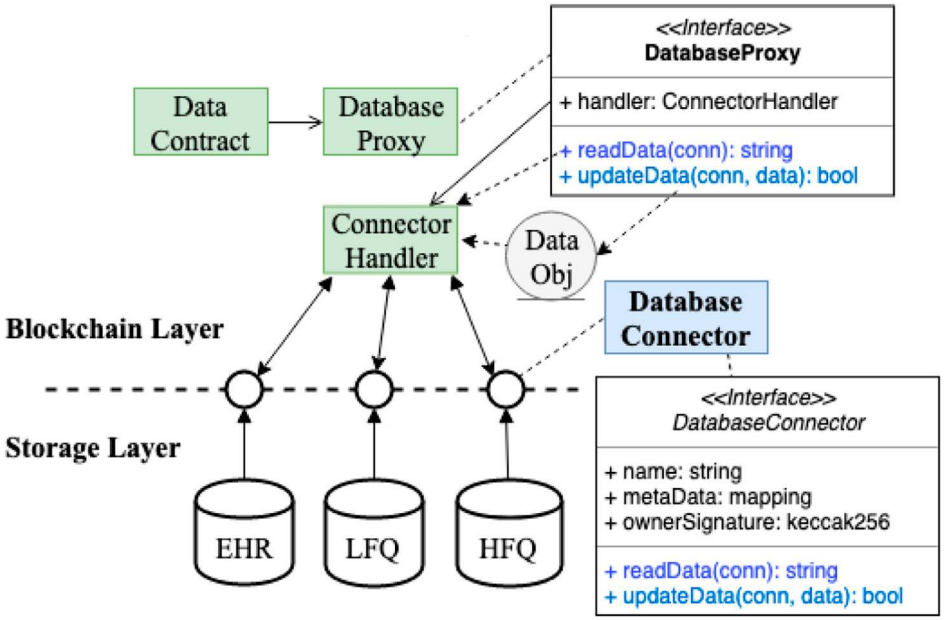

FIGURE 7
Composition of Database Proxy pattern for performing additional security checks before accessing off-chain data store.

have successfully been leveraged in DeFi applications Chen and Bellavitis (2020) to register entities on the blockchain using cryptographic keys and define mutually agreed rules that dictate the updates of appropriate cryptocurrency digital wallets and balances. The direct on-chain exchange of digital healthcare assets is unfortunately hard to achieve due to its high complexity and variability in its management systems. Even if data sharing would be enabled in a decentralized environment, the shared information cannot be openly visible to anyone in the network. Proper authorizations to access sensitive health data must be safeguarded.

4.7.2 Solution → apply the Tokenized Exchange pattern to authorize access to off-chain data storage with a verifiable data access trail

Variability of off-chain data sources can be encapsulated with a standardized interface that encodes high-level information about the data source and a set of basic operations on the data source (e.g., functions to retrieve from the data source or verify origin and integrity of the data source.). Figure 9 presents the structure of the TOKENIZED EXCHANGE pattern that defines an off-chain interface named *Token* to consistently represent each data source. With this interface, the *Database Connector Object* (from the DATABASE CONNECTOR pattern discussed in Section 4.4) that references an off-chain data component can be "tokenized" off-chain with access authorizations being encoded to a standard format using encryption and digital certification algorithms. The employed security mechanism along with any public keys used to generate the tokens are stored as attributes defined in the interface. Tokens generated are maintained on-chain in a shared *Token Registry* smart contract, which captures a history of all events related to the tokens, such as the creation, update, deletion, and access requests. To retrieve the *Database Connector Object*, the recipient needs to possess the authorized party's secret key in order to decrypt the desired data via the DATABASE PROXY pattern presented in Section 4.5.

With this pattern, shared tokens that carry access authorizations can only be consumed by the intended recipient(s) with proper cryptographically paired keys. One limitation to this pattern is that tokens may be hard to standardize in some situations, in which case, implementations of other interfaces may be required. Example interfaces include role-based access control models Sandhu et al. (1996) and access matrix Sandhu and Samarati (1994), which provide more fine-grained authorizations and organization-specific rules that define lower-level permissions to the access tokens. Combined with other patterns in this collection, TOKENIZED EXCHANGE can help design patient-centered DApps such as a patient healthcare record system, where patients can grant providers data access to data they own, provided that a tokenizer interface is implemented.

4.8 Providing notifications of relevant healthcare activities at scale

4.8.1 Design problem faced by DApps for healthcare use cases

The immutability of blockchain is accomplished through a replicated, complete event history, such as digital asset transactions and smart contract function executions. The availability of this information also makes blockchain potentially suitable for improving the coordination of patient care among participants (e.g., physicians, pharmacists, insurance agents, etc.) who traditionally communicate through diverse channels with some degree of manual effort. For instance, to discuss a patient's care case, a provider may share the

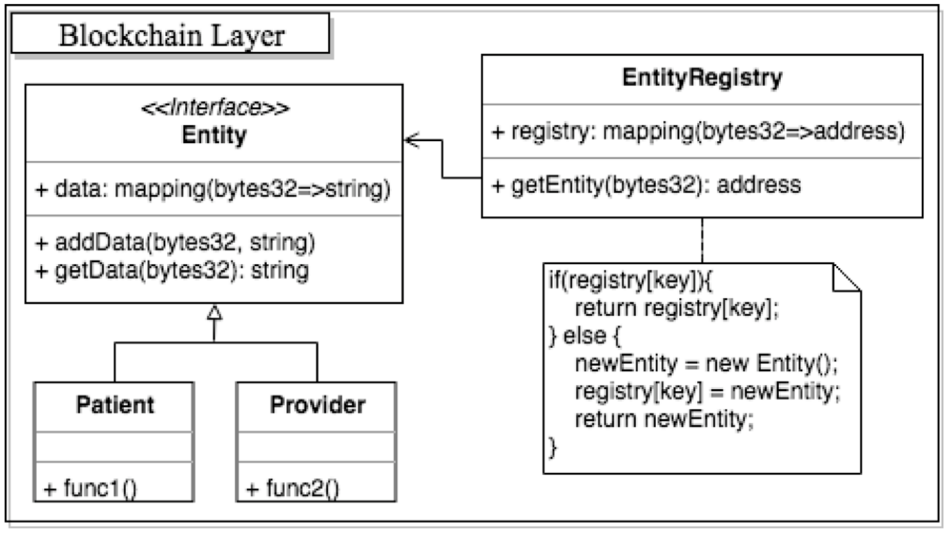

FIGURE 8
Entity registry pattern used with a factory to manage entities and other types of common data while minimizing on-chain storage requirements.

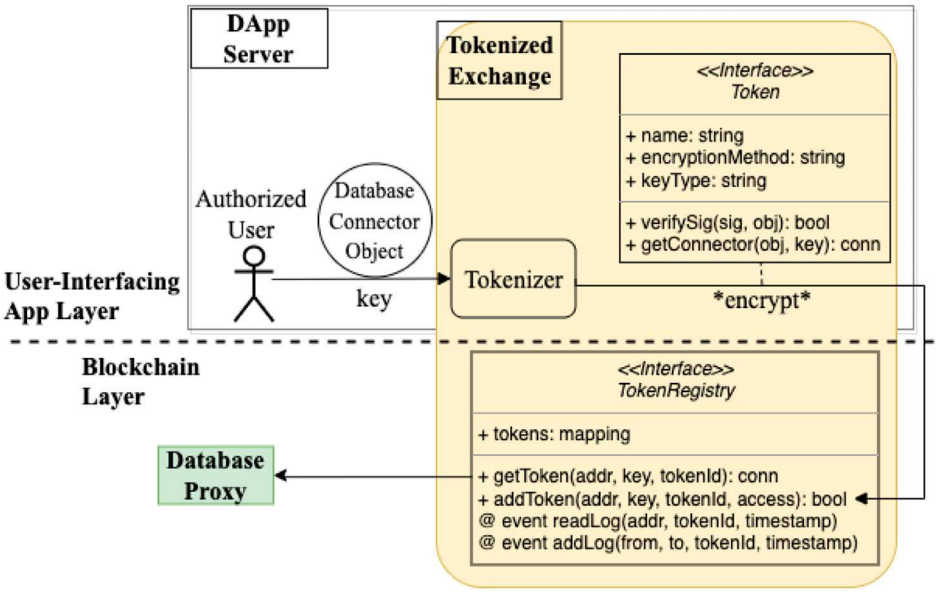

FIGURE 9
Structure of Tokenized Exchange pattern for authorizing off-chain data access and recording verifiable data access logs.

patient's information with a specialist by phone or fax. In a decentralized systems, these interactions are to be captured as part of the event history, which creates a challenge for directly capturing specific health-related topics from an exhaustive search of transactions or topic filtering, which requires non-trivial computation and may result in delayed responses and user fatigue.

An effective design should facilitate coordinated care and provide timely notifications when appropriate. For instance, health-related activities should be communicated to relevant parties from the point when a patient self-reports illness to their prescription pickup activity; clinical reports and follow-up procedures should be relayed to and from the associated care provider offices in a timely manner.

4.8.2 Solution → apply the *Publisher-Subscriber* pattern to manage user notifications at scale when events of interest occur across the decentralized network

To facilitate information filtering and relaying at scale, a notification service using the *Publisher-Subscriber* pattern Buschmann et al. (2007) is needed. In this design, changes in health activities are broadcast to providers who subscribe to

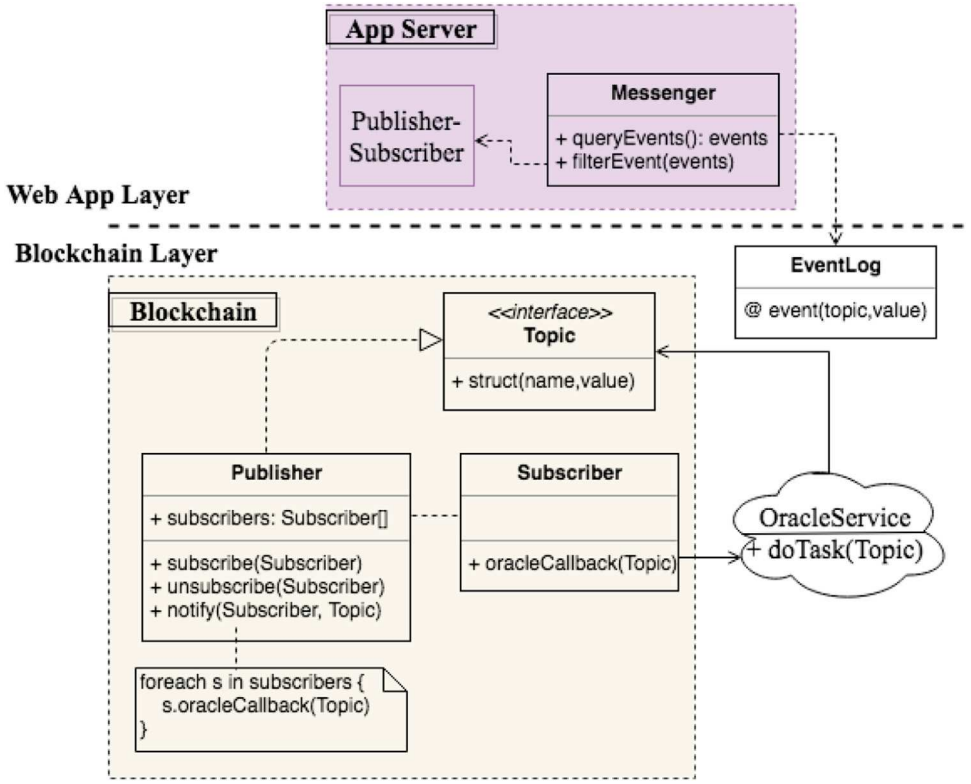

FIGURE 10
Two variants of the publisher-subscriber pattern for providing clinical notifications of relevant healthcare activities at scale.

events related to their patients, which removes the overhead of filtering or manual searching. Due to the deterministic nature of smart contract computations, communications between the on-chain address space and off-chain services can occur in two ways: 1) via a regular polling mechanism, in which an off-chain server delegates a *Messenger* component to monitor changes or new events in the system, and 2) pushing data out to a trusted 3rd party Oracle who performs some computation off-chain and then forwards the results back to the blockchain address space via a callback function[4], such as in Foundation (2015b). The scalability of an Oracle service is yet to be tested however.

The first communication method avoids computation overhead on-chain as it delegates the querying and event processing task to an off-chain server. Specifically, when the publisher has an update, its subscribers only need to do a simple update to an internal state variable that records the publisher's address, which a *Messenger* is created to monitor changes actively. When a change event is detected, the responsibility for the computation-heavy task of content filtering is delegated to the DApp server from the blockchain: the change activity is retrieved directly from the publisher using the address. The DApp server is context-aware at this point because each subscriber has an associated contract address accessible by the server. The *Messenger* can then filter the content based on subscribed topics and update the contract states of appropriate subscribers as needed.

The second approach shifts the responsibility of topic subscriptions and updates to the smart contract component on-chain. When a topic, such as a patient their provider wishes to be notified of any health-related activities, experiences a new event or has a value update, the smart contract logic that notifies the subscribers pushes the updated topic to an Oracle service, which executes some task related to the topic (e.g., sending a secure message to the subscriber regarding the updated event) and sends the result back to the smart contract caller upon task completion.

Figure 10 shows the two variants of PUBLISHER-SUBSCRIBER to provide the notification service.

A solution presently available for DApps makes use of a decentralized interoperability layer called the *Query Execution Layer* GraphProtocol (2018). Figure 11 shows a high-level overview of the Web3 application stack, where the Query Execution Layer is built over the blockchain protocols and interfaces with a DApp. This layer can operate using a *Decentralized Query Protocol*, defined to be a "collection of rules by which clients pay a decentralized network of nodes for indexing, caching, and querying data that is stored on public blockchains and decentralized storage networks such as IPFS/Swarm" GraphProtocol (2018). This protocol can enable

4 https://blockchainhub.net/blockchain-oracles/

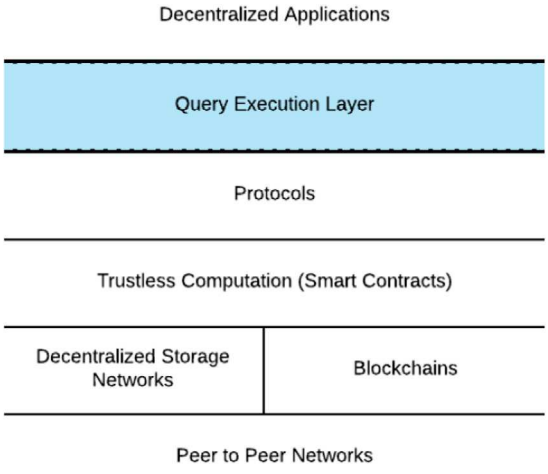

FIGURE 11
A Web3 application stack including the *Query Execution Layer*.

users and DApps to query a chain-agnostic network's data without having to manage a centralized query infrastructure for the DApps.

The solution provided by GraphProtocol (2018) for indexing information on blockchain networks is similar to the *Publisher-Subscriber* pattern, where the topics are published events emitted from a smart contract, subscribed to by decentralized nodes, which subsequently index and curate the information. Custom resolvers for the subgraph may be deployed to nodes to ensure further decentralization of the DApp.

Implementing a notification service in a healthcare DApp is useful when a state change in the shared environment must be reported to interested parties without a complex, many-to-many communication model. Although, a disadvantage to the polling approach is the complexity in the implementation of the messenger component that regularly monitors smart contract events, but it is much more efficient to offload the topic filtering task to off-chain services. The drawbacks to the push-to-oracle approach are on-chain computation overhead and potential costs of Oracle services despite this approach being relatively easier to implement.

5 Concluding remarks

Blockchain and programmable smart contracts provide an ecosystem for creating DApps that have the potential to improve healthcare interoperability on the technical level. However, key properties that make blockchain successful for financial applications–decentralization, immutability, and transparency–also pose major concerns when adopted to create healthcare systems. Specifically, we analyzed concerns related to system evolvability, on-chain storage requirements and the overhead they cause, data privacy, communication scalability in the face of a large number of healthcare users and healthcare data, as well as data authorization. The paper then described a collection of patterns–Layered Ring, Guarded Update, Contract Manager, Database Connector, Database Proxy, Entity Registry, Tokenized Exchange, and Publisher-Subscriber–to mitigate these concerns with code examples and/or their healthcare use cases.

The decentralized nature of blockchain has the potential to enable a more interoperable environment that cannot be easily achieved with traditionally centralized systems, but it also requires careful design choices to implement a reliable and sustainable healthcare DApp. Smart contracts enable programmability on the blockchain, but they can also produce overhead in data storage and communication in addition to expose system vulnerability to malicious attackers. By combining time-proven design practices with an understanding of domain-specific requirements, the collection of patterns and their use cases are proposed to help create healthcare DApps that respect the security and privacy requirements of the domain in addition to being modular, scalable, easy to integrate and maintain.

Data availability statement

Publicly available datasets were analyzed in this study. This data can be found here: https://etherscan.io/.

Author contributions

PZ contributed to conception and design of the study and wrote the first draft of the manuscript. AK wrote sections of this manuscript. DS and JW contributed research ideas and guidance to this study and edited drafts of the manuscript. All authors contributed to the article and approved the submitted version.

Funding

This research is supported by NSF's CISE-CRII program (Project No. 2153232). The authors would like to thank Dr. Kelly Aldrich from Vanderbilt University and COMBINEDBrain (combinedbrain.org) for providing domain expertise to this research.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Ajami, S., and Bagheri-Tadi, T. (2013). Barriers for adopting electronic health records (ehrs) by physicians. *Acta Inform. Medica* 21, 129. doi:10.5455/aim.2013.21.129-134

Arbabi, M. S., Lal, C., Veeraragavan, N. R., Marijan, D., Nygård, J. F., and Vitenberg, R. (2022). A survey on blockchain for healthcare: Challenges, benefits, and future directions. *IEEE Commun. Surv. Tutorials* 25, 386–424. doi:10.1109/COMST.2022.3224644

Atzei, N., Bartoletti, M., and Cimoli, T. (2017). *Principles of security and trust*. Springer, 164–186.A survey of attacks on ethereum smart contracts (sok)

Azaria, A., Ekblaw, A., Vieira, T., and Lippman, A. "Medrec: Using blockchain for medical data access and permission management," in Proceedings of the Open and Big Data (OBD), International Conference on (IEEE), Vienna, Austria, August 2016, 25–30.

Bartoletti, M., and Pompianu, L. (2017). An empirical analysis of smart contracts: Platforms, applications, and design patterns. https://arxiv.org/abs/1703.06322.

Blundell-Wignall, A. (2014). "The bitcoin question: Currency versus trust-less transfer technology," in *OECD working papers on finance, insurance and private pensions* (Paris, France: OECD Publishing), 1.

Broderson, C., Kalis, B., Leong, C., Mitchell, E., Pupo, E., and Truscott, A. (2016). Blockchain: Securing a new health interoperability experience. https://pdfs.semanticscholar.org/8b24/dc9cffeca8cc276d3102f8ae17467c7343b0.pdf.

Buschmann, F., Henney, K., and Schimdt, D. (2007). *Pattern-oriented software architecture: On patterns and pattern language*. Hoboken, New Jersey, United States: John Wiley & Sons.

Buterin, V. (2014). A next-generation smart contract and decentralized application platform. *white paper* 3, 2-1.

Cap, C. M. (2023). Cryptocurrency market capitalizations. https://coinmarketcap.com/.

Chen, Y., and Bellavitis, C. (2020). Blockchain disruption and decentralized finance: The rise of decentralized business models. *J. Bus. Ventur. Insights* 13, e00151. doi:10.1016/j.jbvi.2019.e00151

Chukwu, E., and Garg, L. (2020). A systematic review of blockchain in healthcare: Frameworks, prototypes, and implementations. *Ieee Access* 8, 21196–21214. doi:10.1109/access.2020.2969881

ConsenSys (2018). "Recommendations for smart contract security in solidity," in *Recommendations for smart contract security in solidity* (New York, NY, United States: ConsenSys).

Coplien, J., Hoffman, D., and Weiss, D. (1998). Commonality and variability in software engineering. *IEEE Softw.* 15, 37–45. doi:10.1109/52.730836

CryptoKitties (2023). Cryptokitties. https://www.cryptokitties.co/.

Das, R. (2017). Does blockchain have a place in healthcare. https://www.forbes.com/sites/reenitadas/2017/05/08/does-blockchain-have-a-place-in-healthcare/.

De Aguiar, E. J., Faiçal, B. S., Krishnamachari, B., and Ueyama, J. (2020). A survey of blockchain-based strategies for healthcare. *ACM Comput. Surv. (CSUR)* 53, 1–27. doi:10.1145/3376915

DeSalvo, K., and Galvez, E. (2015). Connecting health and care for the nation: A shared nationwide interoperability roadmap—version 1.0. https://www.healthit.gov/buzz-blog/.

Dourlens, J. (2017). *Ethereum smart contracts lifecycle*.

Dubovitskaya, A., Xu, Z., Ryu, S., Schumacher, M., and Wang, F. (2017). Secure and trustable electronic medical records sharing using blockchain. https://arxiv.org/abs/1709.06528.

E Napoletano, B. C. (2022). Proof of stake. Web. https://www.forbes.com/advisor/investing/cryptocurrency/proof-of-stake/.

Ellervee, A., Matulevicius, R., and Mayer, N. (2017). A comprehensive reference model for blockchain-based distributed ledger technology. http://kodu.ut.ee/~andrease/ellervee blockchain.

Etherescan (2023). Etherescan - the ethereum blockchain explorer. https://etherscan.io/.

Fernandez, E. B. (2013). *Security patterns in practice: Designing secure architectures using software patterns*. Hoboken, New Jersey, United States: John Wiley & Sons.

Fomo3D (2023). Fomo3d. https://fomo3d.net/.

Foundation, E. (2015b). Oraclize limited. http://www.oraclize.it/.

Foundation, E. (2015a). Solidity. https://solidity.readthedocs.io/en/develop/.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. "Design patterns: Abstraction and reuse of object-oriented design," in Proceedings of the European Conference on Object-Oriented Programming, Kaiserslautern, Germany, July 1993, 406–431.

Gamma, E., Vlissides, J., Johnson, R., and Richard, H. (1995). *Design patterns: Elements of reusable object-oriented software*. London, United Kingdom: Pearson Education.

Geraci, A., Katki, F., McMonegal, L., Meyer, B., Lane, J., Wilson, P., et al. (1991). IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries. https://ieeexplore.ieee.org/document/182763.

Gmbh, I. (2023b). Cosmos, the internet of blockchains, features. https://cosmos.network/features.

Gmbh, I. (2023a). What is tendermint. Web. https://docs.tendermint.com/v0.33/introduction/what-is-tendermint.html.

GraphProtocol (2018). The graph: A decentralized query protocol for blockchains. https://github.com/graphprotocol/research/blob/master/papers/whitepaper/the-graph-whitepaper.pdf.

Hub, B. (2017). Blockchain oracles. Web. https://insights.sei.cmu.edu/sei_blog/2017/07/what-is-bitcoin-what-is-b lockchain.html.

Idex - Decentralized Ethereum Asset Exchange (2018). Idex - decentralized ethereum asset exchange. https://idex.io/.

Johnston, D., Yilmaz, S. O., Kandah, J., Bentenitis, N., Hashemi, F., Gross, R., et al. (2014). The general theory of decentralized applications, dapps. *GitHub, June* 9.

Kugler, L. (2021). Non-fungible tokens and the future of art. *Commun. ACM* 64, 19–20. doi:10.1145/3474355

Lesh, K., Weininger, S., Goldman, J. M., Wilson, B., and Himes, G. "Medical device interoperability-assessing the environment," in Proceedings of the 2007 Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability (HCMDSS-MDPnP 2007), Boston, MA, USA, June 2007, 3–12.

Lundkvist, C., Heck, R., Torstensson, J., Mitton, Z., and Sena, M. (2017). Uport: A platform for self-sovereign identity. URL: https://whitepaper.uport.me/uPort_whitepaper_DRAFT20170221.pdf.

Melnick, E. R., Fong, A., Nath, B., Williams, B., Ratwani, R. M., Goldstein, R., et al. (2021). Analysis of electronic health record use and clinical productivity and their association with physician turnover. *JAMA Netw. Open* 4, e2128790. doi:10.1001/jamanetworkopen.2021.28790

Moreno, J., Fernandez, E. B., Fernandez-Medina, E., and Serrano, M. "A security pattern to incorporate blockchain in big data ecosystems," in Proceedings of the EuroPLoP-24th European Conference on Pattern Languages of Programs, Irsee, Germany, July 2019.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf.

Onc (2014). Connecting health and care for the nation: A 10-year vision to achieve an interoperable health it infrastructure. https://www.healthit.gov/sites/default/files/ONC10yearInteroperabilityConceptPaper.pdf.

OpenZeppelin (2023b). Access control. https://docs.openzeppelin.com/contracts/4.x/access-control.

OpenZeppelin (2023a). Erc1155. https://docs.openzeppelin.com/contracts/4.x/api/token/erc1155.

Palladino, S. (2017). The parity wallet hack explained. https://blog.zeppelin.solutions/on-the-parity-wallet-multisig-hack-405a 8c12e8f7.

Peter, B., and Nichol, J. B. (2016). Co-creation of trust for healthcare: The cryptocitizen. framework for interoperability with blockchain. *Res. Propos. Res.*

Peterson, K., Deeduvanu, R., Kanjamala, P., and Boles, K. (2016). A blockchain-based approach to health information exchange networks. https://www.healthit.gov/sites/default/files/12-55-blockchain-based-approach-final.pdf.

Porru, S., Pinna, A., Marchesi, M., and Tonelli, R. "Blockchain-oriented software engineering: Challenges and new directions," in Proceedings of the 39th International Conference on Software Engineering Companion, Buenos Aires, Argentina, May 2017, 169–171.

Rene, M. L., and Stephen, G. (2020). What is ethereum 2.0? ethereum's consensus layer and merge explained. https://decrypt.co/resources/what-is-ethereum-2-0.

Richesson, R. L., and Nadkarni, P. (2011). Data standards for clinical research data collection forms: Current status and challenges. *J. Am. Med. Inf. Assoc.* 18, 341–346. doi:10.1136/amiajnl-2011-000107

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-based access control models. *Computer* 29, 38–47. doi:10.1109/2.485845

Sandhu, R. S., and Samarati, P. (1994). Access control: Principle and practice. *IEEE Commun. Mag.* 32, 40–48. doi:10.1109/35.312842

Shvets, A. (2015). Design patterns explained simply. https://sourcemaking.com/.

Siegel, D. (2016). Understanding the dao attack. http://www.coindesk.com/understanding-dao-hack-journalists.

Six, N., Herbaut, N., and Salinesi, C. (2022). Blockchain software patterns for the design of decentralized applications: A systematic literature review. *Blockchain Res. Appl.* 3, 100061. doi:10.1016/j.bcra.2022.100061

Solidity.Readthedocs (2017). Ethereumio. Contracts. http://solidity.readthedocs.io/en/develop/contracts.html.

Xu, X., Pautasso, C., Zhu, L., Gramoli, V., Ponomarev, A., Tran, A. B., et al. "The blockchain as a software connector," in Proceedings of the 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), Venice, Italy, April 2016, 182–191.

Xu, X., Pautasso, C., Zhu, L., Lu, Q., and Weber, I. "A pattern collection for blockchain-based applications," in Proceedings of the 23rd European Conference on Pattern Languages of Programs, Irsee, Germany, July 2018, 1–20.

Zdun, U., Hentrich, C., and Van Der Aalst, W. M. (2006). A survey of patterns for service-oriented architectures. *Int. J. Internet Protoc. Technol.* 1, 132–143. doi:10.1504/ijipt.2006.009739

Zhang, P., Schmidt, D. C., White, J., and Lenz, G. (2018b). Blockchain technology use cases in healthcare. *Blockchain technology: Platforms, tools, and use cases. Adv. Comput.* 111, 1–41. doi:10.1016/bs.adcom.2018.03.006

Zhang, P., Walker, M. A., White, J., Schmidt, D. C., and Lenz, G. "Metrics for assessing blockchain-based healthcare decentralized apps," in Proceedings of the 2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom), Dalian, China, October 2017c, 1–4.

Zhang, P., White, J., and Schmidt, D. "Architectures and patterns for leveraging high-frequency, low-fidelity data in the healthcare domain," in Proceedings of the 2018 IEEE International Conference on Healthcare Informatics (ICHI), New York, NY, USA, June 2018c, 463–464.

Zhang, P., White, J., Schmidt, D. C., and Lenz, G. "Design of blockchain-based apps using familiar software patterns with a healthcare focus," in Proceedings of the 24th Conference on Pattern Languages of Programs, Vancouver British Columbia Canada, October 2017a, 19.

Zhang, P., White, J., Schmidt, D. C., and Lenz, G. "Design of blockchain-based apps using familiar software patterns with a healthcare focus," in Proceedings of the 24th Conference on Pattern Languages of Programs, Vancouver British Columbia Canada, October 2017b, 19.

Zhang, P., White, J., Schmidt, D. C., Lenz, G., and Rosenbloom, S. T. (2018a). Fhirchain: Applying blockchain to securely and scalably share clinical data. *Comput. Struct. Biotechnol. J.* 16, 267–278. doi:10.1016/j.csbj.2018.07.004