MANCHESTER 1824
The University of Manchester

Understanding Blockchain Applications from Architectural and Business Process Perspectives

A thesis submitted to the University of Manchester for the degree of
Doctor of Philosophy
in the Faculty of Science and Engineering

2023

Fouzia E. Alzhrani
School of Engineering
Department of Computer Science

Contents

Word count: 58,100

List of figures

List of publications

F. E. Alzhrani, K. A. Saeedi, and L. Zhao, "A taxonomy for characterizing blockchain systems," *IEEE Access*, vol. 10, pp. 110 568–110 589, 2022, issn: 2169-3536. doi:10.1109/ACCESS.2022.3214837.

F. Alzhrani, A collection of Industry-developed Blockchain-based Applications, version 1.0, Zenodo, Nov. 2020. doi: 10.5281/zenodo.4268953.

F. Alzhrani, A Collection of Event Logs of Blockchain-based Applications, version 1.0, Zenodo, Jun. 2022. doi: 10.5281/zenodo.6637059.

F. Alzhrani, A Collection of Papers on Blockchain Business Applications, Zenodo, Jan. 2023. doi: 10.5281/zenodo.7564584.

F. Alzhrani, BELA: Blockchain Event Log App, https://github.com/alzhraniFouz/BELA, version 1.3.0, 2023. doi: 10.5281/zenodo.7620035.

Abstract

Blockchain is a promising cross-industry technology. With the rapid evolution of the technology, academia and industry are exploring the applicability of blockchain in various domains, including healthcare, supply chain management, and Internet of Things. This technology, with its characteristics of decentralization, anonymity, persistency, and auditability, delivers a new way to enforce trust among distrusted business partners. It combines cryptography, peer to peer networking, data management, consensus protocols and incentive mechanisms to support optimal execution of transactions between involved parties. Blockchain applications are complex, heterogeneous, and require cooperation and interoperation with non-blockchain systems. Their complexity is further exacerbated by the lack of a clear understanding of their composition, as well as the stringent demand on functional and non-functional requirements. This thesis aims to address these shortfalls and is set out to gain an understanding of blockchain applications from architectural and business process perspectives. This understanding is elaborated through several relevant, yet independent, research contributions: a taxonomy, software patterns and pattern languages, and a process-aware framework design and implementation. These artifacts are supported by comprehensive datasets of Industry-developed and Academia-researched blockchain applications, as well as a set of event logs related to these applications. Several research methodologies were adopted to produce the contributions, including literature review, software decomposition, domain analysis, and automated business process discovery. The research was validated through a mixed method approach which proofs that such understanding can better inform software architects and developers in their analysis, design and implementation of blockchain applications.

Declaration of originality

I hereby confirm that no portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright statement

i The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made *only* in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see `http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420`), in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations (see `http://www.library.manchester.ac.uk/about/regulations/`) and in The University's policy on Presentation of Theses.

Dedication

To my mother, who asks for so little but gives so very much, for all the love, inspiration, support, and guidance.

Forever in my heart, my late father, for this big dream that we made together, and I celebrate alone.

Acknowledgements

I wish to gratefully acknowledge the scholarship I received from King Abdulaziz University that enables me to pursue a Ph.D. degree at the University of Manchester.

I would also like to thank my supervisor, Dr. Liping Zhao, for her invaluable supervision and encouragement during the course of my Ph.D. degree. Her treasured support was really influential in shaping my research contributions and critiquing my results. I also thank my co-supervisor, Dr. Kawther Saeedi, for her encouragement and guidance, and for always having my back and believing in me during my difficult times.

I would like to express my gratitude to my mother, my husband, and my children. Without their tremendous understanding and encouragement in the past few years, this would not be possible. My appreciation also goes out to the rest of my family for their encouragement and support all through my study.

Chapter 1

Introduction

1.1 Research Motivation

A blockchain is a secured management system in which business transactions are stored as chains of blocks [1]. Blockchains are considered highly secure due to their anonymity, persistency, auditability, and decentralization. Originally developed to record cryptocurrency transactions [2], this technology gains popularity with applications in diverse domains that extend far beyond cryptocurrencies [3]–[6]. These domains include education [7], healthcare [8], energy supply [9], and supply chain management [10].

Blockchain technology brings new opportunities for organizations to re-imagine business models [11], [12]. This technology offers the ability of a peer-to-peer (P2P) business management and eliminates the complex and hierarchical management of business transactions. It helps to ensure the integrity of business processes and data without involving a trusted third party [12], [13]. Encoding organizational business processes within smart contracts allows for confidential execution and monitoring of the processes [14], [15]. As blockchain technology enables cross-organizational processes, blockchain applications can be integrated with legacy systems [16].

However, developing such applications is obstructed by several challenges [17], [18]. First, blockchain applications are highly complex and heterogeneous, involving cooperation and interoperation between non-blockchain and blockchain systems. Thus, the lack of a detailed understanding of the composition of blockchain systems hinders the design and implementation of highly interoperable blockchain systems [19]. There is a large number of blockchain taxonomies have been proposed in the literature [19]–[28]. However, a comprehensive classification of blockchain systems is still missing.

Second, there is a stringent demand on non-functional requirements (NFRs) or quality requirements, such as performance, scalability, portability, interoperability, reliability, privacy, and security [29]. A good software architecture can aid in the design of a software system that will satisfy key quality requirements, but a bad one can be disastrous [30]. Work on software architectures for blockchain applications is rather limited, as it has primarily been focused on computing networks of blockchain systems using the P2P architecture [29], [31].

Third, defining proper smart contracts and solution architecture requires a tight connection between analysis and design phases [32]. Current practices of blockchain application development face a difficulty of communicating functional requirements (FRs) in the design of the application [33], [34]. The lack of requirement analysis makes it challenging to identify which requirements, if they are addressed by smart contracts, constitute value to stakeholders. Current research focuses mainly on software design, rarely on requirement analysis, or advances to implementation and overlooking the requirements [32], [35], [36].

Finally, Process Mining (PM) [37] can help in recovering FRs embedded as business processes within smart contracts. However, the cryptographic format and heterogeneous structures of event data generated by different smart contracts require technical expertise to extract encoded transactions data from blockchain networks and transform the data into a format adequate for PM tools [38], [39]. There are a number of approaches have been proposed to generate useful event logs from blockchain data. However, a detailed discussion on how to extract and decode event data before formatting them is lacking. Moreover, the availability of event data is constrained by the process awareness of a blockchain application, which is under-reported by the literature.

This thesis aims to address the aforementioned shortfalls by providing a systematic understanding of blockchain applications from architectural and business process perspectives.

1.2 Research Aim and Objectives

The fundamental assumption of my Ph.D. thesis is that it is possible to identify recurring architecture and business process patterns from different blockchain applications. Based on this assumption, my research aim is to discover reusable architectural and business process patterns from existing blockchain applications and to validate using these patterns to support the design of blockchain applications. In line with this research aim, my thesis has five objectives:

- **Objective 1:** To study different types of blockchain application systems and their characteristics from industry and academia
- **Objective 2:** To synthesize characteristics of blockchain application systems in a comprehensive taxonomy
- **Objective 3:** To identify architectural patterns for blockchain application development
- **Objective 4:** To develop a framework to support process mining from blockchain applications
- **Objective 5:** To identify reusable business process patterns for blockchain application development

1.3 Research Methodology and Contributions

To achieve the above objectives, I have adopted a range of research methods, including domain analysis, case study, literature review, peer-review, automated business process discovery, conformance checking, and prototyping. The output resulted from each objective aims to make a research contribution. Figure 1.1 outlines my research contributions and their interrelationships. In what follows, I summarize the research methods used to achieve my objectives and the resulting contributions.

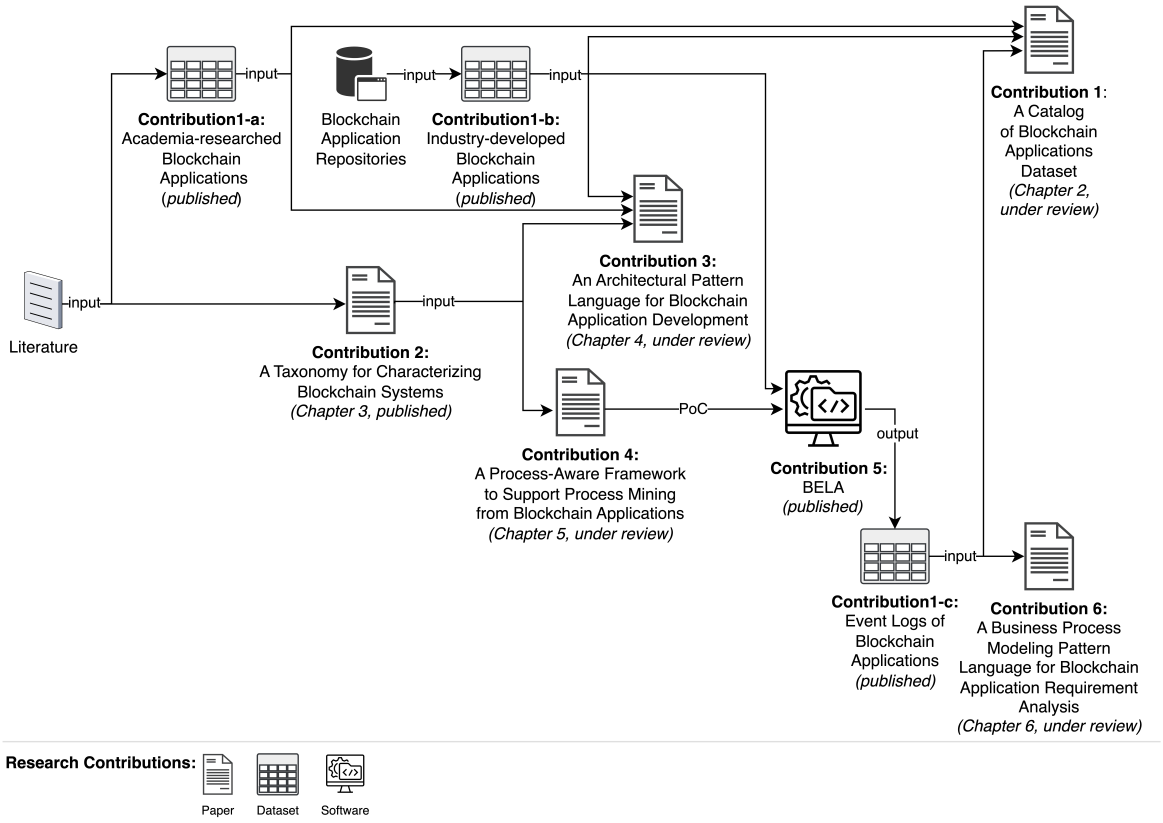

Figure 1.1. An overview of thesis contributions and their interrelationships

- **Contribution 1:** A comprehensive catalog of blockchain applications from industry and academia. The datasets are published open access on **Zenodo**. This catalog consists of three datasets: a) 400 Industry-developed applications [40], b) event logs of 101 applications [41], and c) 63 publications on blockchain applications [42]. This contribution resulted in a data paper. The paper is submitted to **Data in Brief** and presented in Chapter 2. The datasets were systematically collected using a mixed method approach. The industry-developed applications were manually selected based on a set of Inclusion/Exclusion criteria. This dataset was then filtered based on their process-awareness, in which the event logs were automatically generated for the filtered applications. The set of publications was selected through snowballing [43].
- **Contribution 2:** A component-based taxonomy for characterizing blockchain system applications. This taxonomy is comprehensive, characterizing a blockchain system using three subsystems, eight fundamental components, 83 aspects, and 198 features. This contribution resulted in a research article that is published in **IEEE ACCESS**

[44] and presented in Chapter 3. To build the proposed taxonomy, a blockchain system was decomposed into a hierarchy of components. At the top level, the system is organized into three subsystems: *an External Subsystem*, *an Internal Subsystem* and *an Execution Environment Subsystem*. Then, these subsystems were decomposed into eight fundamental components. Each component is further divided into different aspects, and each aspect is characterized by different features. The aspects and features are identified inductively and deductively, following Nickerson et al.'s approach for taxonomy development [45].

- **Contribution 3:** An architectural pattern language (APL) to support the architectural design of blockchain applications. This pattern language consists of 12 patterns organized within three architectural views: *Structural*, *Interactional*, and *Transactional*. This contribution resulted in a research article that is submitted to **Information and Software Technology** and presented in Chapter 4. To build the proposed pattern language, the set of Industry-developed applications were analyzed based on the proposed taxonomy to identify their architectural components. Then, these components and their relationships were mapped onto the known software architecture patterns [46], [47] to identify a set of architectural patterns for blockchain applications.

- **Contribution 4:** A process-aware framework (PAF) to support identifying process-aware blockchain applications and to support generating useful event logs for these applications. This framework consists of two modules: 1) *Process Awareness Recognizer (PAR)*, a rule-based classifier to assess the process awareness of a given blockchain application, and 2) *Event Log Generator (ELG)*, an automated batch processing model to generate useful event logs from extracted event data from blockchain networks. This contribution resulted in a research article that is submitted to the special issue of **Artificial Intelligence for Process Mining** in **Engineering Applications of Artificial Intelligence** and presented in Chapter 5. Based on a domain analysis [48], which identified several challenges of mining business process from blockchain applications, the framework was designed, instantiated, and tested [49].

- **Contribution 5:** A software application to extract event data from blockchain networks compatible with Ethereum Virtual Machine (EVM) and to generate useful event logs from these data. This contribution is a proof of concept prototype of the PAF, which is described in Chapter 5. The prototype is called a *Blockchain Event Log App (BELA)* and is documented on **GitHub** [50]. The software is implemented in JavaScript as a Node.js[1] application.

- **Contribution 6:** A business process modeling pattern language (BPMPL) to support requirement analysis of blockchain applications. This pattern language consists of nine data-driven patterns organized into two categories: *Token-Oriented Patterns* and *Smart Contract-Oriented Patterns*. This contribution resulted in a research article that is submitted to **Requirements Engineering** and presented in Chapter 6. This study applies PM techniques to identify the patterns. The business process models were first

[1] https://nodejs.org

recovered through automated business process discovery on an analysis subset of the event logs dataset. Then, the discovered models were generalized and, finally, validated through models conformance checking on a validation subset of event logs.

1.4 Thesis Structure and Chapter Overview

This thesis is written in the journal format in accordance with the University of Manchester policy for thesis presentation[2]. The main chapters within this thesis, i.e. Chapters 2 — 6, contain five papers that are formatted using the template of the journals to which they are submitted. The order of these chapters represents the trajectory of my research and how the early chapters influence the latter chapters, as Figure 1.1 shows. However, as each chapter reports a stand-alone research contribution, it discusses related work in the context of the reported research; it also describes the methodology used, research findings, and limitations. The remaining chapters of this thesis are organized as follows:

- **Chapter 2: A Catalog of Blockchain Applications Dataset**. This chapter contains a submitted journal paper that describes the collection, curation, and significance of the datasets.
- **Chapter 3: A Taxonomy for Characterizing Blockchain Systems**. This chapter contains a published research article [44]. The paper proposes a taxonomy for characterizing blockchain applications and demonstrates its utility on a set of real-world blockchain systems. It presents a holistic background on blockchain technology and its fundamental characteristics. It, also, reviews related work on blockchain-oriented taxonomies.
- **Chapter 4: An Architectural Pattern Language for Blockchain Application Development**. This chapter contains a submitted research article that describes and validates the proposed architectural pattern language. It synthesizes findings on blockchain application types and shows how this classification helps to understand blockchain application architectures. The chapter reviews related work on blockchain architectures and software patterns.
- **Chapter 5: A Process-Aware Framework to Support Process Mining from Blockchain Applications**. This chapter contains a submitted research article that describes and validates the proposed framework. It presents a demonstration of using BELA to produce useful event logs. It, also, reviews related work on process mining from blockchain applications.
- **Chapter 6: A Business Process Modeling Pattern Language for Blockchain Application Requirement Analysis**. This chapter contains a submitted research article that describes, validates, and discusses the implications and limitations of the proposed pattern language. The chapter is supplemented by a running example of design-

[2]http://www.regulations.manchester.ac.uk/pgr-presentation-theses

ing a blockchain application using the proposed pattern language. The chapter also reviews related work on blockchain-oriented software patterns.

- **Chapter 7: Conclusion**. This chapter concludes the thesis by summarizing the achievements and limitations of my Ph.D. research and outlining some work that I wish to carry out in the future.

Chapter 2

A Catalog of Blockchain Applications Dataset

Research Objective

Objective 1: To study different types of blockchain application systems and their characteristics from industry and academia

Thesis Context

This chapter provides a comprehensive resource of software-, process-, and application-centric datasets that are analyzed in and produced by the contributions in Chapters 4, 5, and 6. This chapter is based on a paper that has been submitted to Data in Brief. The full reference of this paper is as follows:

F. Alzhrani, K. Saeedi, and L. Zhao, "A data catalog of blockchain applications," *Data in Brief*, Under Review.

CRediT authorship contribution statement

Fouzia Alzhrani: Conceptualization, Data Curation, Methodology, Software, Writing - Original Draft, Writing - Review& Editing. **Kawther Saeedi**: Conceptualization, Supervision, Writing - Review & Editing. **Liping Zhao**: Conceptualization, Supervision, Writing - Review & Editing.

A Catalog of Blockchain Applications Dataset

Fouzia Alzhrani[a,b], Kawther Saeedi[a] and Liping Zhao[b,*]

[a]King Abdulaziz University, Jeddah, Saudi Arabia
[b]The University of Manchester, Manchester, United Kingdom

ARTICLE INFO

Keywords:
event log
business process management
process mining
software applications
Ethereum Virtual Machine

ABSTRACT

This catalog is a collection of systematically curated and processed blockchain applications data. The data are organized into three sub datasets: industry-developed applications, event data and logs generated from process-aware industry-developed applications, and academia-researched applications. The data were systematically collected through a mixed method approach. The collection process is semi-automated by implementing a reusable software application to extract, decode, and format event data from blockchains. This catalog allows research on understanding blockchain applications from architectural and business process perspectives. In particular, three research contributions were made based on the analysis of data in this catalog. Further insights and results could be obtained from this dataset on aspects related to blockchain applications, business process management, and machine learning.

Specifications Table

Subject:	Software Engineering
Specific subject area:	Blockchain research and blockchain application development
Type of data:	Table
How the data were acquired:	The set of industrial applications were manually collected from online blockchain application repositories. A systematically selected subset of these applications were used to generate the event logs. These logs were automatically generated using a software application specifically developed for extracting and encrypting event data from blockchain networks. The set of academia-researched applications were manually collected via snowballing from online databases.
Data format:	Raw, Analyzed, Processed
Description of data collection:	The data were collected through a mixed method approach according to a set of inclusion and exclusion criteria for each sub dataset and at different time frames.
Data source location:	State of The Dapps, IBM Code Pattern, R3 Marketplace, IEEE Xplore, PubMed, ScienceDirect, Google Scholar, ACM Digital Library, Wiley Online Library, MDPI, SpringerOpen, SpringerLink, Ethereum blockchain, POA blockchain
Data accessibility:	Repository name: Zenodo. 1) A Collection of Industry-Developed Blockchain-Based Applications: Data identification number:10.5281/zenodo.4268952 Direct URL to data: https://doi.org/10.5281/zenodo.4268952 2) A Collection of Event Logs of Blockchain-Based Applications: Data identification number:10.5281/zenodo.6637059 Direct URL to data: https://doi.org/10.5281/zenodo.6637059 3) A Collection of Papers on Blockchain Business Applications: Data identification number: 10.5281/zenodo.7564584 Direct URL to data: https://doi.org/10.5281/zenodo.7564584

*Corresponding author
feaalzhrani@kau.edu.sa (F. Alzhrani); ksaeedi@kau.edu.sa (K. Saeedi); liping.zhao@manchester.ac.uk (L. Zhao)
ORCID(s): 0000-0002-5780-0469 (F. Alzhrani)

Value of the Data

- This dataset contains raw business process data that are encrypted and have not been preprocessed, allowing researchers and analysts to perform any preprocessing. As such, evaluating encryption algorithms and generating new event logs from different viewpoints.
- The processed (decrypted) event logs are readily-available for process mining tools. They can be reused to accelerate research on business process management to study blockchain applications. This eliminates the need for technical expertise to extract and process event data from blockchains, which is a challenging task [1, 2].
- Researchers can use data in this catalog to evaluate consensus algorithms based on transactions event data. For example, researchers can use the companion software [3] to generate new event logs from the new version of Ethereum networks that uses Proof-of-Stake protocol, and then, conduct comparative analysis with the provided event data in this catalog that are produced by Proof-of-Work networks.
- Future research can benefit from this catalog to perform analysis of blockchain applications based on historical data. For example, the event logs can stimulate research in developing blockchain monitoring systems and new techniques to analyze user/customer behaviors, also based on prices.
- The data are structured and labeled, making them a comprehensive resource for data mining and machine learning research.
- The datasets can be collectively analyzed through different approaches to produce valuable results. For example, by analyzing the datasets in this catalog, we were able to identify a set of interrelated business process modeling patterns [4] through the application of process mining techniques [5] and a set of interconnected architectural patterns [6] through software decomposition based on the blockchain systems taxonomy by Alzhrani et al. [7].

1. Objective

This catalog was built to support research on understanding blockchain applications from architectural and business process perspectives.

2. Data Description

The collected blockchain applications datasets fall into three major categories: Industry-developed applications, event data logs, and academia-researched applications. First, **DApps Open Dataset.xlsx** contains a list of 400 industry-developed blockchain applications. These applications are listed in Zenodo [8]. The attributes of this dataset are described in Table 1. The selected applications are built on nine different platforms: Ethereum, Steem, EOS, Blockstack, Klaytn, POA, Hive, Corda, and Hyperledger Fabric. The distributions of these applications across the different platforms and sources are listed in Table 2. The distribution of the applications across different development stage, application domains, and license are shown in Fig. 1, Fig. 2, and Fig. 3, respectively.

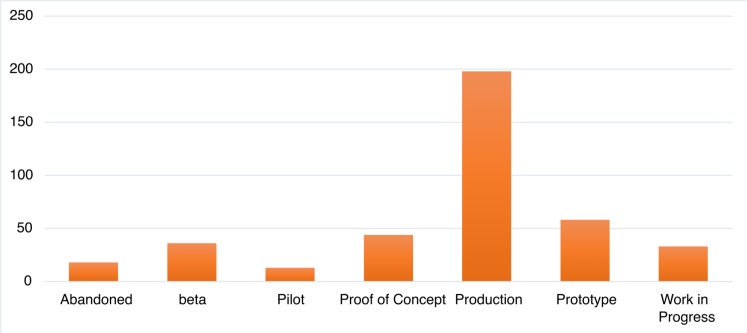

Figure 1: The distribution of blockchain applications across different development stages

Second, there are 232 event data files in Comma Separated Values (CSV) format. They are listed in Zenodo [9]. They were extracted for 101 blockchain applications selected from the 400 ones. For each application, there are two

Table 1
Attributes of the industrial blockchain applications dataset

Attribute	Meaning
ID	Identifier of the application in the dataset
Application Name	Name of the application as provided by its source
Source	Repository where the application is listed
Webpage	A URL to the application's web page
Email	Developers' contact email address
Twitter	Link to the application's Twitter account
Facebook	Link to the application's Facebook account
LinkedIn	Link to the application's LinkedIn account
Medium	Link to the application's Medium account
Application Domain	Business industry, classified by the researcher
Category	Application category, classified by the source repository
Blockchain Type	Type of hosted blockchain network (public, private, consortium)
Development Stage	Development status/version of the application
License	Software license of the application
Source Code	Link to application's source code
Platform Name	Name of the host blockchain platform

Table 2
The distributions of blockchain applications across the different platforms and repositories

Repository	Platform	Number of Applications
State of The DApps	Ethereum	230
	Steem	20
	EOSIO	29
	Blockstack	1
	Klaytn	18
	POA	5
	Hive	5
R3 Marketplace	Corda	75
IBM Code	Hyperledger Fabric	17

files: 1) raw event data file named as **original-*network-application***, where *network* is the host blockchain network and *application* is the application name. 2) decoded event data decrypted into a human-readable format named as **decoded-*network-application***, where *network* is the host blockchain network and *application* is the application name. If the application is hosted on multiple blockchain networks, then there are two event log files (encoded and decoded) for each network. In total, the dataset has 116 encoded and 116 decoded event logs. The attributes of this dataset are described in Table 3. Fig. 4 shows the distribution of event logs per blockchain network.

In addition, this dataset has **Event Registry from Contract ABIs.csv** file contains a comprehensive list of event names and their signatures calculated by our developed software for the 101 applications.

Finally, **list-of-papers.xlsx** contains a list of 63 peer-reviewed publications published between 2016 and 2020 inclusively. These publications are listed in Zenodo [10]. The attributes of this dataset are described in Table 4. There are 53 journal papers, 8 conference papers, and 2 book chapters. The yearly distribution of the publications is shown in Fig. 5. The publications are from nine databases and 11 publishers. The databases are IEEE Xplore, PubMed, ScienceDirect, Google Scholar, ACM Digital Library, Wiley Online Library, MDPI, SpringerOpen, SpringerLink. The publishers are IEEE, Springer, Elsevier, MDPI, Edward Elgar Publishing, Korean Society of Medical Informatics, Taylor and Francis Ltd., Inderscience Enterprises Ltd., John Wiley and Sons Ltd., Oxford University Press, and ACM.

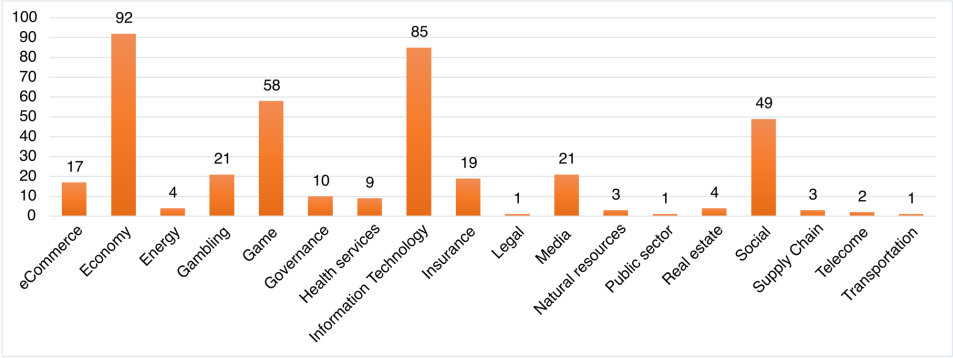

Figure 2: The distribution of blockchain applications across different application domains

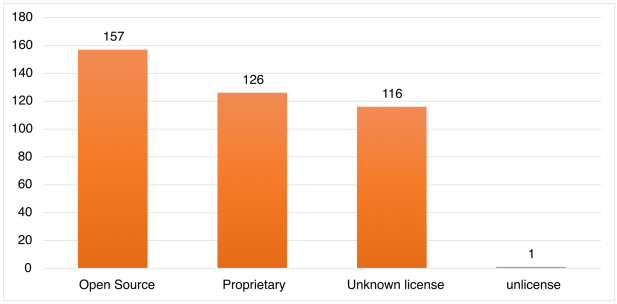

Figure 3: The distribution of blockchain applications across different licenses

Table 3
Attributes of the event logs dataset

Attribute	Meaning
address	Smart contract address
topics	Encrypted event name and indexed parameters
data	Non-indexed parameter of event
blockNumber	Number of the block in a particular chain
timeStamp	Time of block confirmation
gasPrice	Amount of ETH that must be paid to miners for processing transactions on the network
gasUsed	the Sum of the gas for each operation executed by the Ethereum Virtual Machine
logIndex	Index of the event in the block logs
transactionHash	Unique identifier used to identify a particular transaction
transactionIndex	Index of the transaction in the block

The distribution of the publications across these databases is shown in Fig. 6 whereas Fig. 7 shows the distribution across different publishers.

3. Experimental design, materials and methods

The process of building this catalog consists of three activities: 1) select Industry-developed applications, 2) generate event logs, and 3) select academia-researched applications. Fig. 8 shows the process, illustrated in Integration DEFinition (IDEF0) notation for process modeling [11]. In what follows, we elaborate each activity.

3.1. Select blockchain applications

We manually selected 400 blockchain applications between September 2019 and February 2020 from these nine different blockchain platforms: Ethereum, Steem, EOS, Blockstack, Klaytn, POA, Hive, Corda, and Hyperledger

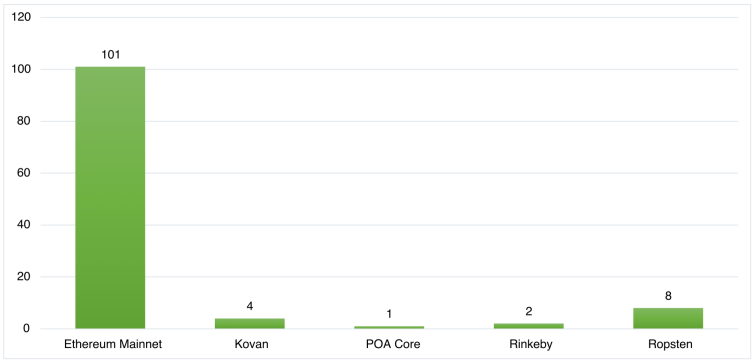

Figure 4: The distribution of event logs per blockchain networks

Table 4
Attributes of the blockchain academia-researched applications dataset

Attribute	Meaning
No.	number of publication record in the dataset
Reference Type	type of the publication (Journal Article, Conference paper, Book Section)
Author	list of authors of the publication
Title	title of the publication
Year	publication year
Journal	venue of the publication
Database	publication source database
Publisher	publisher of the publication
Application Scope	scope of blockchain applications discussed by the publication (general, specific)

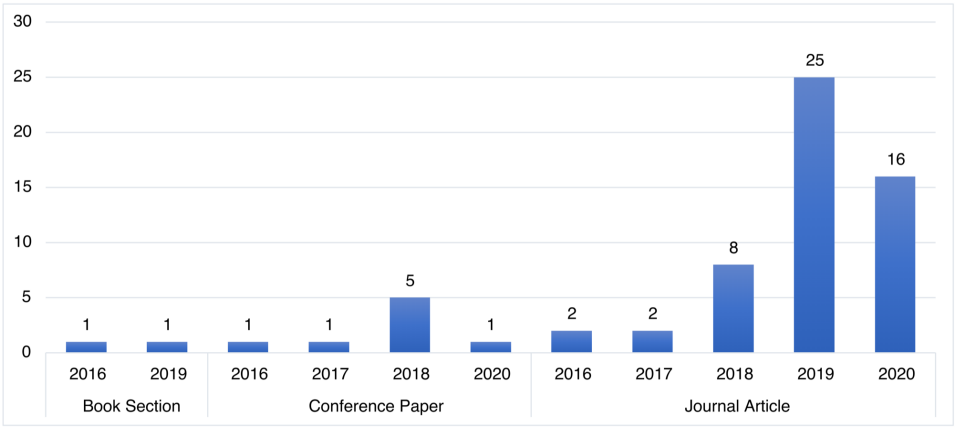

Figure 5: The yearly distribution of the publications per type

Fabric. They were selected from three blockchain application repositories: *State of The DApps*[1], *R3 Marketplace*[2], and *IBM Code Pattern*[3]. The set of industrial applications were selected based on the following criteria:

- Include applications with smart contract functionality, i.e. Blockchain 2.0 and Blockchain 3.0
- Include applications built on open source blockchain platform
- Include applications built on cross-industry blockchain platform
- Exclude Blockchain 1.0 applications

[1]https://www.stateofthedapps.com, last accessed: July 2022
[2]https://marketplace.r3.com, last accessed: November 2020
[3]https://developer.ibm.com/patterns, last accessed: July 2022

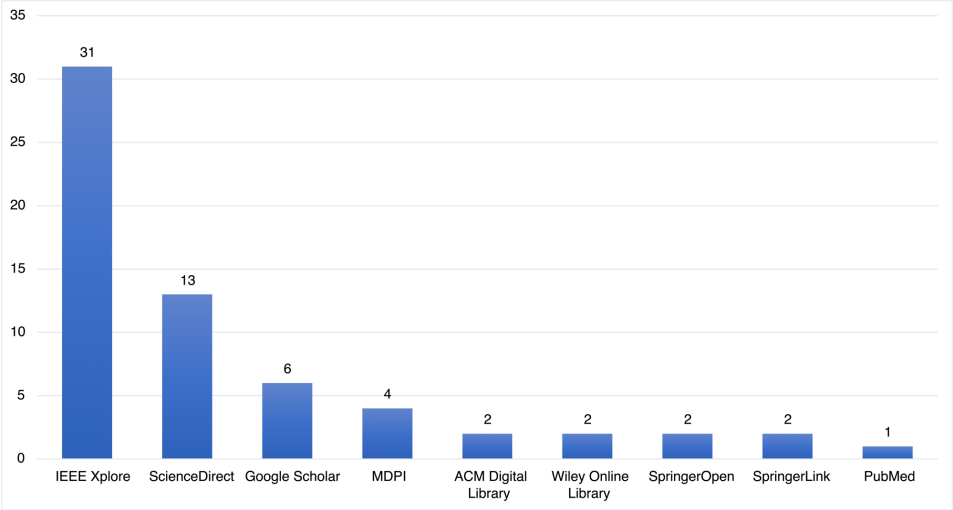

Figure 6: The distribution of the publications across different databases

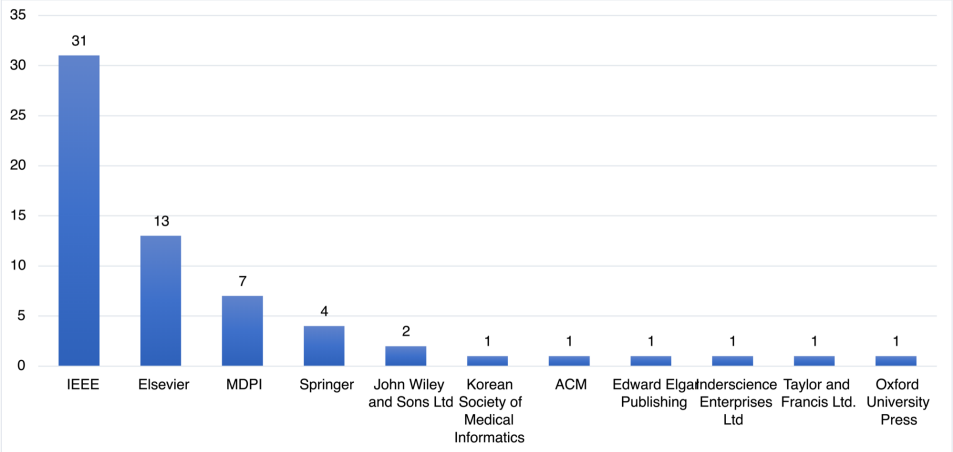

Figure 7: The distribution of the publications across different publishers

- Exclude applications lack supporting documents
- Exclude other applications that are related to protocols, application programming interfaces, and software development kits

The naming of application *Category* is inconsistent between the application repositories. Therefore, we have created a list of business domains and mapped the selected applications to the list, as in *Application Domain*. In particular, *Finance, Exchanges, High risk* are mapped to *Economy*, *Development, Wallet, Security, IoT, Technology, Storage, Asset Management, Identity, Business and Information, Property* are mapped to *Information Technology*, and *Marketplace* is mapped to *eCommerce*.

3.2. Generate event logs

Based on the findings from [12], we identified a set of criteria that identifies process-aware blockchain applications. We then applied these criteria on the Industry-developed application set and identified 103 process-aware blockchain applications[4]. The set of event logs were generated for applications that meets the following criteria:

- Host blockchain platform is Ethereum Virtual Machine EVM-Compatible
- Host blockchain platform supports defining smart contract events

[4] As of July 2020

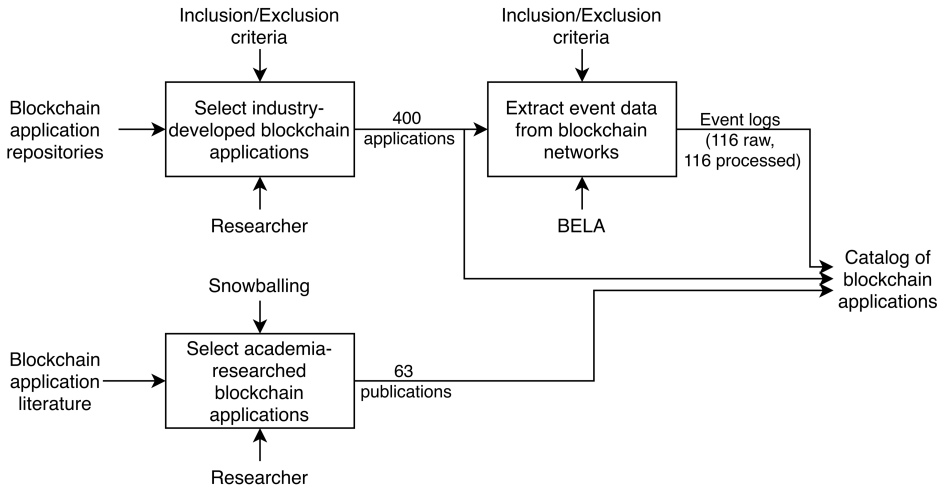

Figure 8: The methodology of building the catalog of blockchain applications dataset

- Host blockchain network is accessible
- Application has at least one verified smart contract
- The smart contract has programmer-defined events

For the 103 applications, we collected their smart contract addresses during 24-30 July 2020. There were a total of 252 contract addresses. The process of extracting event data from blockchains and generating useful event logs was automated via developing an event log extractor application, BELA [3]. BELA is documented on GitHub, however, we briefly describe the workflow of BELA as follows:

1. Generate dynamic Remote Procedure Calls (RPCs) to blockchain networks
2. Retrieve contract Application Binary Interface (ABI) for a verified contract address
3. Calculate event signature - *topic0* - for each event in the contract ABI
4. Retrieve event data for each event in the contract ABI
5. Decode event data:
    - Convert *timestamps* from Unix to UTC format
    - Match and replace *topics[0]* with the relevant calculated event signature *topic0*
    - Convert *blockNumber, gasPrice, gasUsed, logIndex, transactionIndex* from hexadecimal to string format item Decode *data* from detected types to string
6. Generate an event log file in CSV format for an application.

The event logs were retrieved in January 2021 from five blockchain networks: Ethereum Mainnet, POA Core, Ropsten, Kovan, and Rinkeby [5]. Two APIs were used as *BaseAPI*: Etherscan API[6] for Ethereum Mainnet and Testnets (i.e., Ropsten, Kovan, and Rinkeby), and Blockscout API[7] for POA Core. By default, the APIs returns the 1000 most recent event data records. Therefore, we maximized the number of retrieved records by executing five RPCs per event. The first RPC starts from *first* block and the subsequent RPCs start from the last retrieved block in the preceding call. All RPCs are to *latest* block. In sum, a set of 252 addresses of 103 applications was ingested into the extractor application, and a total of 232 event logs were produced (116 raw, 116 processed).

3.3. Select peer-reviewed publications on business applications

In 2022, we systematically selected a set of 63 publications on blockchain business applications. We adapted a snowballing approach [13]. The start set was identified by selecting highly cited papers from Google Scholar. The set has 16 papers. Then we performed several iterations of backward and forward snowballing to identify relevant papers. The set of publications were selected based on the following criteria:

[5] In 2022, Ropsten, Kovan, and Rinkeby were depricated
[6] https://etherscan.io/apis
[7] https://blockscout.com/poa/core/api-docs

- Include Peer-reviewed research articles, conference proceedings papers, book chapters, review papers.
- Include publications that discuss business applications of blockchain technology
- Exclude non-English articles, articles with missing abstracts, notes, editorials
- Exclude generic articles related to the blockchain technology and/or blockchain architecture
- Exclude software-oriented articles related to the blockchain technology
- Exclude articles addressing technical aspects of blockchain technology

Acknowledgments

Funding: This work was supported by King Abdulaziz University, Jeddah, Saudi Arabia.

Ethics statements

Authors state that the data supporting their paper meet the data requirements of the journal, ensuring that they have written an original work, not submitting this paper for consideration in another journal. Authors, also, declares not potential ethical issues in the paper or during its writing.

Declaration of Competing Interest

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

References

[1] C. Klinkmüller, A. Ponomarev, A. B. Tran, I. Weber, W. van der Aalst, Mining blockchain processes: Extracting process mining data from blockchain applications, in: International Conference on Business Process Management, Business Process Management: Blockchain and Central and Eastern Europe Forum, Springer International Publishing, 2019, pp. 71–86. doi:10.1007/978-3-030-30429-4_6.

[2] R. Mühlberger, S. Bachhofner, C. Di Ciccio, L. García-Bañuelos, O. López-Pintado, Extracting event logs for process mining from data stored on the blockchain, in: International Conference on Business Process Management, Business Process Management Workshops, Springer International Publishing, 2019, pp. 690–703. doi:10.1007/978-3-030-37453-2_55.

[3] F. Alzhrani, BELA: Blockchain Event Log App, https://github.com/alzhraniFouz/BELA, 2023. doi:10.5281/zenodo.7620035.

[4] F. Alzhrani, K. Saeedi, L. Zhao, A business process modeling pattern language for blockchain application requirements analysis (Under Review).

[5] A. Burattin, Process Mining, Springer International Publishing, Cham, 2015, pp. 33–47. doi:10.1007/978-3-319-17482-2_5.

[6] F. Alzhrani, K. Saeedi, L. Zhao, An architectural pattern language for blockchain application development (Under Review).

[7] F. E. Alzhrani, K. A. Saeedi, L. Zhao, A taxonomy for characterizing blockchain systems, IEEE Access 10 (2022) 110568–110589.

[8] F. Alzhrani, A collection of industry-developed blockchain-based applications, 2020. doi:10.5281/zenodo.4268953.

[9] F. Alzhrani, A Collection of Event Logs of Blockchain-based Applications, 2022. doi:10.5281/zenodo.6637058.

[10] F. Alzhrani, A Collection of Papers on Blockchain Business Applications, 2023. doi:10.5281/zenodo.7564584.

[11] A. Presley, D. H. Liles, The use of idef0 for the design and specification of methodologies, in: Proceedings of the 4th industrial engineering research conference, 1995, pp. 442–448.

[12] F. Alzhrani, K. Saeedi, L. Zhao, A process-aware framework to support process mining from blockchain applications (Under Review).

[13] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, Association for Computing Machinery, New York, NY, USA, 2014, pp. 1–10. doi:10.1145/2601248.2601268.

Chapter 3

A Taxonomy for Characterizing Blockchain Systems

Research Objective

Objective 2: To synthesize characteristics of blockchain application systems in a comprehensive taxonomy

Thesis Context

This chapter provides a technical foundation for the other contributions in Chapter 4, 5, and 6. This chapter is based on a paper that has been published by IEEE ACCESS. The full reference of this paper is as follows:

F. E. Alzhrani, K. A. Saeedi, and L. Zhao, "A taxonomy for characterizing blockchain systems," *IEEE Access*, vol. 10, pp. 110 568–110 589, 2022, issn: 2169-3536. doi:10.1109/ACCESS.2022.3214837.

CRediT authorship contribution statement

Fouzia Alzhrani: Conceptualization, Data Curation, Methodology, Writing - Original Draft, Writing - Review & Editing. **Kawther Saeedi**: Conceptualization, Supervision, Writing - Review & Editing. **Liping Zhao**: Conceptualization, Supervision, Writing - Review & Editing.

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 17 August 2022, accepted 5 October 2022, date of publication 14 October 2022, date of current version 21 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3214837

RESEARCH ARTICLE

A Taxonomy for Characterizing Blockchain Systems

FOUZIA E. ALZHRANI[1,2], KAWTHER A. SAEEDI[1], (Member, IEEE), AND LIPING ZHAO[2]
[1]Information Systems Department, King Abdulaziz University, Jeddah 21589, Saudi Arabia
[2]Department of Computer Science, The University of Manchester, M13 9PL Manchester, U.K.

Corresponding author: Liping Zhao (liping.zhao@Manchester.ac.uk)

This research work was funded by Institutional Fund Project under grant no (IFPIP: 474-612-1443); therefore, authors gratefully acknowledge technical and financial support from the Ministry of Education and King Abdulaziz University, Jeddah, Saudi Arabia.

ABSTRACT Since its inception more than a decade ago, blockchain technology has been quickly adopted by a large number of sectors, including finance, healthcare, energy supply, and logistics, due to its numerous benefits, such as decentralization, persistency, anonymity and auditability. However, blockchain applications are intrinsically complex, as they are heterogeneous in nature and require cooperation and interoperation with non-blockchain systems. The complexity of the blockchain systems is further exacerbated by the lack of a clear understanding of their composition. This paper aims to bridge this knowledge gap by proposing a taxonomy for characterizing blockchain systems. The proposed taxonomy classifies a blockchain system into a hierarchy of components: At the top level, the system is organized into an external subsystem, an internal subsystem and an execution environment subsystem. These subsystems are then decomposed into eight fundamental components: Platform, Network, Distributed Ledger, Smart Contract, Consensus Protocol, Digital Wallet, Token, and Node. Each component is further divided into different aspects and each aspect is characterized by various features. This taxonomy thus provides a comprehensive understanding of the composition of a blockchain system, which can better inform software developers in their design and implementation of blockchain systems. The paper presents and illustrates the proposed taxonomy through some application scenarios and a case study.

INDEX TERMS Blockchain taxonomy, blockchain applications, blockchain systems, blockchain components, blockchain characteristics.

I. INTRODUCTION

Blockchain technology is a decentralized transaction and data management technology capable of providing security, anonymity, and data integrity without any third-party organization in the control of the transactions [1]. Originally developed for Bitcoin cryptocurrency, blockchain technology has now been applied to a wide variety of applications in diverse domains [2], [3], such as education [4], [5], healthcare services [6], smart cities [7], energy supply [8], agriculture and food [9], supply chain management [10], and Internet of Things (IoT) [11]. The great potential of blockchain technology lies in its key characteristics of decentralization, persistency, anonymity, and auditability [12], which make it possible for transactions to be carried out securely without the involvement of a third party. Such a decentralized environment ensures independent control of transactions and data, providing security, anonymity, and data integrity.

However, despite its great potential, developing blockchain applications faces many technical difficulties, as such applications are highly complex and heterogeneous, involving cooperation and interoperation between non-blockchain and blockchain systems. These difficulties are further exacerbated by the lack of a detailed understanding of the composition of blockchain systems. This knowledge gap is considered to be the main obstacle in the design and implementation of highly interoperable blockchain systems [13], [14]. With the increased importance of blockchain technology, there is

The associate editor coordinating the review of this manuscript and approving it for publication was Aasia Khanum.

an urgent need for a common terminology to describe the detailed components of blockchain systems.

Although a large number of blockchain taxonomies have been proposed in the literature [13], [15], [16], [17], [18], [19], [20], [21], [22], [23], a comprehensive classification of blockchain systems is still missing. This paper aims to bridge this knowledge gap by proposing a taxonomy for characterizing blockchain systems. The proposed taxonomy classifies a blockchain system into a hierarchy of components: At the top level, the system is organized into an external subsystem, an internal subsystem and an execution environment subsystem. These subsystems are then decomposed into eight fundamental components: Platform, Network, Distributed Ledger, Smart Contract, Consensus Protocol, Digital Wallet, Token, and Node. Each component is further divided into different aspects and each aspect is characterized by different features. The aspects and features are identified inductively and deductively [24]. This taxonomy thus provides an elaborate and comprehensive understanding of the composition of a blockchain system, which can be used to inform software developers in their design and implementation of blockchain systems.

The remainder of this paper is organized as follows: Section II provides a brief overview of the blockchain technology and its fundamental characteristics, while Section III reviews related work on blockchain taxonomies. Section IV presents our taxonomy. Section V illustrates the proposed taxonomy through some application scenarios and a case study. Section VI discusses the results and observations through a comparative analysis of the proposed taxonomy and the reviewed literature. Finally, Section VII concludes the paper and suggests future research.

II. OVERVIEW OF BLOCKCHAIN TECHNOLOGY

Since its inception in 2008, blockchain technology has gone through five generations of transformations [25], [26], known as Blockchain 1.0, Blockchain 2.0, etc., as depicted in Figure 1. In 2008, the first generation of blockchain technology was developed to support the transfer of Bitcoin.[1] Blockchain technology was used to create an immutable, distributed ledger – a record-keeping system – to store data and perform transactions with minimal trust requirements. Business logic is used to exchange cryptocurrencies. In 2013, smart contracts and blockchain were combined in a proposal by Ethereum[2] to allow for more complex financial transactions than simple payments and transfers. Such transactions include loans, mortgages, bonds, and stock. Blockchain 2.0 was for money transfer and was based only on cryptocurrencies. In 2015, it became possible to store and exchange physical world assets on blockchains via tokenization, which marked the evolution of Blockchain 3.0; IOTA[3] is one such example [27]. In 2018, consensus protocols in blockchain systems were empowered by artificial intelligence in Seele[4] as Blockchain 4.0. In 2019, Blockchain 5.0 emerged via Relictum Pro.[5] It restructured blockchains by offering N-dimensional smart contracts. It has expanded blockchain possibilities to enable the formalization of any activity in human life.

According to the literature [28], [29], [30], [31], [32], [33], blockchain technology has these four fundamental characteristics:

1) *A Distributed Transactional Platform Supported by a Peer-to-Peer Network*. Due to the limited trust and transparency in traditional business models, intermediaries and third parties are required. This requirement makes these models process heavy. Blockchain offers the ability to trust peer-to-peer exchange and automated execution of business contracts. This eliminates the third party and ensures the integrity of processes and data with security and cryptography. The elimination of intermediaries drives a new ecosystem of players and fosters the creation of new profit pools, microeconomies, competitors, consumers, and distributed ecosystems [30]. In general, transactions on blockchains can be in the form of transfers, broadcasts, or contract calls. Transfer transactions can be one- or two-way transactions. A one-way transfer is when one participant moves value to another participant with no need to receive value in return, such as ownership transfer and right transfer. A two-way transfer is an exchange between two or more participants, such as cryptocurrency, data, and digital asset exchange. A broadcast is when one participant updates the ledger with value not intended for the participant, such as contract creation transactions. In contract call transactions, an implemented function within a smart contract is invoked by another contract.
2) *Computational Capability through Smart Contracts.* Blockchain technology offers computational capabilities via smart contracts. A smart contract is a program that defines a set of functions to express highly complex transaction logic [34]. These functions enable various blockchain applications in various domains. The capability of a smart contract depends on its operating platform, including its programming language and execution machine. Although smart contracts are assumed to be deterministic by nature, the availability of the required information upon executing a smart contract can affect its determinism [35]. When a smart contract requires information from the external world of blockchains, it becomes nondeterministic. Blockchain technology serves two purposes when multiple organizations are involved. First, the correctness of the flow controls and workflow structure is guaranteed as being a repository for business process descriptions. Second,

[1] https://bitcoin.org/en/
[2] https://ethereum.org/
[3] https://www.iota.org/
[4] https://seele.pro/
[5] https://relictum.pro/

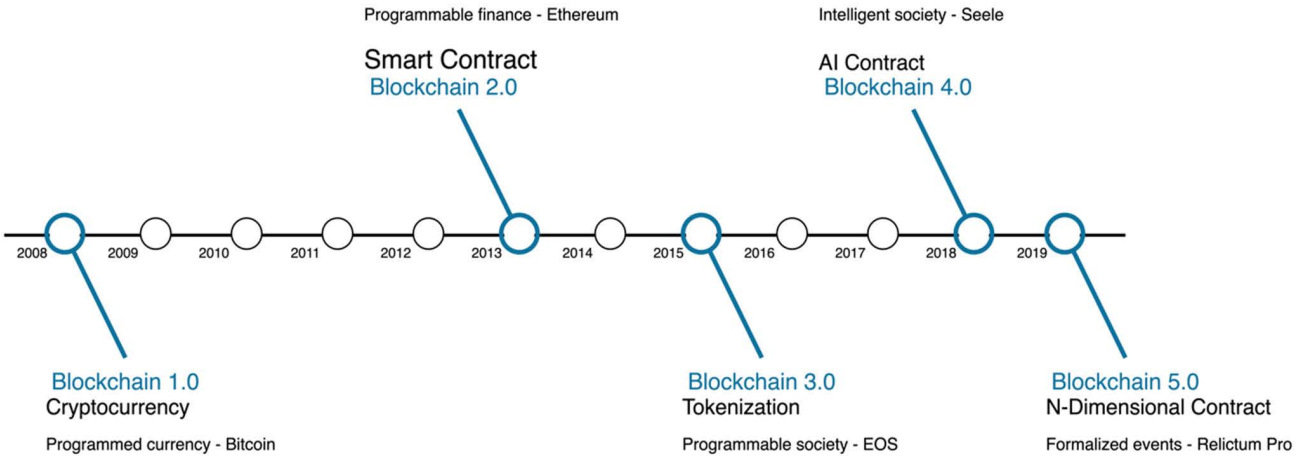

FIGURE 1. Blockchain revolution timeline.

it guarantees the validity of operations as the generated process executions are encoded into smart contracts. Global platform and application-specific rules in smart contracts are used to validate the consistency of transactions in blockchains [28], [29], [30], [31], [32].

3) *Security by Design for Immutability, Integrity, and Transparency.* The most commonly reported properties of current blockchains are immutability, integrity, and transparency [25], [28], [32], [36], [37], [38]. Once transactions are confirmed, data are permanently stored in the ledger, tamper-proof and immutable. Cryptographic mechanisms in blockchains ensure data integrity. Transparency is achieved through public access to the stored data. In addition to these nonfunctional properties, the two significant functionalities of blockchain systems are data storage and computational infrastructure. In addition, blockchain systems can offer data and computation communication, coordination, and facilitation mechanisms as software connectors [28], [29], [36]. Given these characteristics, we can claim that blockchain technology has been architected to be secure by design. Its in-built security is derived from the assumption that the peers involved in a transaction are trustless. It was structured from the beginning to safeguard these transactions. In terms of design, blockchain technology includes cryptography and distributed consensus as preventive mechanisms to reduce the risks of cyberattacks. Cybersecurity is necessary to safeguard transactions from external threats.
4) *Complementary Technology through Interoperability.* Blockchain technology is a sophisticated complement to current technologies rather than replacing them. Although systems built on top of blockchains can operate alone, there is a need to communicate with other centralized or distributed systems in real business. To a large extent, a blockchain system needs to be interoperable with other blockchain or non-blockchain systems. However, the closed execution environment of a blockchain system does not allow smart contracts to interact directly with external servers or states. Thus, unique approaches to blockchain interoperability have been developed [39]. They are oracles [28], [36], hash locking [40], [41], [42], notary [41], [42], relays [41], [42], and application programming interface (API) gateways [43], [44]. Blockchain interoperability approaches can be classified into three categories. The first is non-blockchain to blockchain, where data are passed from traditional systems to blockchain systems. The second is a blockchain-to-compatible blockchain, where cross-chain data transfer is enabled between compatible blockchains. The last category is blockchain-to-non-compatible blockchain, where cross-chain data transfer is enabled between non-compatible blockchains.

Through our literature study we found that regardless of their variations, all blockchain systems are based on the following eight fundamental architectural components [16], [28], [32], [38], [45], [46], [47]:

- ***Platform***: A computing environment in which blockchain applications are executed.
- ***Peer-to-Peer Network***: A computer network formed by two or more interconnected computer systems (nodes). The systems on the network are equal (thus called peers) and can share resources without requiring a central server.
- ***Distributed Ledger***: A shared, decentralized database system in the blockchain network that provides a transparent, immutable, and cryptographically verifiable transaction record. The ledger has built-in mechanisms that prevent unauthorized transaction entries and enforce consistency in the shared view of the recorded transactions. Unlike traditional distributed databases,

distributed ledgers do not have central data storage or special administrative functionality.

- ***Consensus Protocol***: A set of rules based on which users of a blockchain application can reach an agreement or consensus.
- ***Smart Contract***: A computer program that can automatically execute transaction agreements between multiple participants.
- ***Token***: A digital asset that represents cryptocurrency or a real-world object.
- ***Digital Wallet***: A program that stores and manages digital keys and digital assets for blockchain users.
- ***Node***: A computer system on the blockchain network. Nodes are used to store the aforementioned distributed ledgers, consensus protocols, smart contracts, tokens, and digital wallets.

III. RELATED WORK

Numerous studies have attempted to classify the characteristics of the blockchain technology. Ankenbrand et al. [21] proposed a universal taxonomy of asset characteristics. It aims to bridge traditional finance and emerging cryptographic assets by providing a consistent view and standardized terminology for different asset types. It extends existing crypto-assets classification frameworks and summarizes the findings in a morphological box. The taxonomy consists of 14 dimensions: claim structure, technology, underlying, governance, consensus/validation mechanism, legal status, information complexity, legal structure, information interface, total supply, issuance, redemption, transferability, and fungibility. The taxonomy consists of 14 dimensions and 44 characteristics. The utility of this taxonomy was demonstrated through an illustrative scenario using six crypto assets.

Schulze et al. [17] proposed a taxonomy of blockchain platforms. The taxonomy consists of 10 dimensions and 26 characteristics organized in a morphological box. The dimensions are orientation, system architecture, ownership, interaction, governance, access, transaction validation, consensus difficulty, codebase, and reference type. The utility of the taxonomy was illustrated using a single blockchain platform.

In the context of distributed trust and reputation management systems (DTRMS), blockchain technology has been addressed by Bellini et al. [18]. The authors aimed to provide a guide for adapting blockchain technology in DTRMS. For this purpose, they developed two taxonomies: the first one is for blockchain technology, and the other one is for DTRMS. Then, a formal concept analysis was applied to the proposed taxonomies to identify the most stable and recurrent features between both taxonomies. The study identified nine dimensions and 24 characteristics organized in a tree structure. The dimensions are openness, access management, business logic, data, ledger distribution, fee, tokenization, consensus protocols, and type. The utility of the taxonomy was not demonstrated.

Another contribution to crypto-assets was due to Arslanian and Fischer [22], who addressed the lack of a universally accepted classification system for such assets and proposed a simplified crypto-asset taxonomy. Their taxonomy consisted of two nested dimensions and five characteristics, which were organized into a hierarchy. The taxonomy was not evaluated.

The study by Tasca and Tessone [13] contributed to the standardization of the classification of blockchain technology. It provided a hierarchical classification of blockchain characteristics. This study aimed to understand the architectural configurations of the technology. Blockchain systems were decomposed into functional and logical components. However, the taxonomy-building approach and the classification basis were not described. Their proposed taxonomy consists of eight dimensions: consensus, transaction capabilities, native currency/tokenization extensibility, security and privacy, codebase, identity management, and charging and rewarding system. The taxonomy consisted of 37 nested dimensions and 72 characteristics. However, the utility of the taxonomy was not demonstrated.

The work by Wieninger, Schuh and Fischer [15] provided different variations in blockchain systems beyond the popular classification of public, private, and hybrid systems. It enabled the differentiation of blockchain types by systematically classifying the fundamental characteristics of the technology in a morphological box. The proposed taxonomy consisted of 11 dimensions: authorization to view transactions, right of proposal, validation of transactions, awareness of identities, type of token, incentive for validation, location of the asset, possibility of change, source code, consensus mechanism, and Turing-complete. In total, the taxonomy consisted of 11 dimensions and 27 characteristics. The utility of the taxonomy was illustrated using a single blockchain platform.

With the goal of investigating blockchain characteristics similar to ours, Sarkintudu et al. [19] developed a taxonomy of financial blockchain platforms. That taxonomy aimed to address the complexity of blockchain implementation to enable access to its full potential. The taxonomy was structured as a table with five columns (dimensions): Operation mode, visibility, task, design architecture, and consensus mechanism. In total, the taxonomy consists of five dimensions and 15 characteristics. The utility of the taxonomy was demonstrated using 24 blockchain platforms.

An earlier taxonomy that addressed the characteristics and classifications of tokens was proposed by Oliveira et al. [23]. The proposed taxonomy was organized into four dimensions: purpose, governance, functional, and technical. In total, the taxonomy consisted of 17 nested dimensions and 40 characteristics. This taxonomy was evaluated through a case study.

The study by Tönnissen and Teuteberg [20] focused on smart contracts as the fundamental blockchain components. The study aimed to systematically classify terminology related to smart contracts in order to facilitate the investigation of current use cases and future challenges of smart

TABLE 1. A comparison of existing taxonomies based on the eight fundamental components (Y=Yes, N=No).

Reference	Platform	P2P Network	Ledger	Consensus Protocol	Smart Contract	Token	Network Node	Digital Wallet
Ankenbrand et al. [21]	N	N	N	N	N	Y	N	N
Schulze et al. [17]	Y	N	N	N	N	N	N	N
Bellini et al. [18]	Y	Y	Y	Y	Y	N	N	N
Arslanian and Fischer [22]	N	N	N	N	N	Y	N	N
Tasca and Tessone [13]	Y	N	N	Y	N	Y	N	N
Wieninger et al. [15]	Y	N	N	Y	N	Y	N	N
Sarkintudu et al. [19]	Y	N	N	Y	N	N	N	N
Oliveira et. al. [23]	N	N	N	N	N	Y	N	N
Tönnissen and Teuteberg [20]	N	N	N	N	Y	N	N	N
Xu et. al. [16]	Y	Y	Y	Y	N	N	N	N

contracts. The taxonomy consisted of nine dimensions and 24 characteristics which were organized into a table. The dimensions were subject matter, function of contracts, time horizon, ability to renegotiate terms, involvement of consumers, existence of mutual trust, sub-categories of contracts, trigger, and cost of altering. This taxonomy was evaluated by interviewing experts.

One of the earliest works on blockchain classification was by Xu et al. [16]. It aimed to address architectural considerations about blockchain and blockchain-based systems. The proposed taxonomy captured major blockchain architectural characteristics. The taxonomy grouped these characteristics into four main dimensions: decentralization, blockchain configuration, storage and computation, and other architectural designs and deployment. Within each dimension, a classification system was structured as a table. A total of 30 nested dimensions and 42 characteristics were identified. Although blockchain examples were used to illustrate the taxonomy, no evaluation was carried out.

Table 1 summarizes the above reviewed taxonomies based on the eight fundamental architectural components described early. As can be seen, none of these taxonomies has considered all these components. Our taxonomy aims to provide a comprehensive blockchain taxonomy based on these eight components.

IV. THE PROPOSED TAXONOMY

An overview of the proposed taxonomy is given in Subsection A. Subsections B to D present a thorough analysis of the blockchain components and their characteristics.

A. OVERVIEW

The proposed taxonomy classifies a blockchain system into five levels, as shown in Figure 2, comprising the *system-level*, the *subsystem-level*, the *component-level*, the *aspect-level*, and finally, the *feature-level*. At the system-level, a blockchain system is decomposed into three subsystems, made up of an *Execution Environment*, an *Internal* and an *External* subsystem. At the subsystem-level, a subsystem is divided into a set of architectural components, based on the eight fundamental components of blockchain systems introduced in Section II. At the component-level, a component is divided into a set of aspects (or *sub-components*). At the aspect-level, each aspect is instantiated by a specific feature (or *characteristic*), selecting from a set of alternative features. Some features may be further divided into sub-features. The top three levels of our taxonomy are shown in Figure 3.

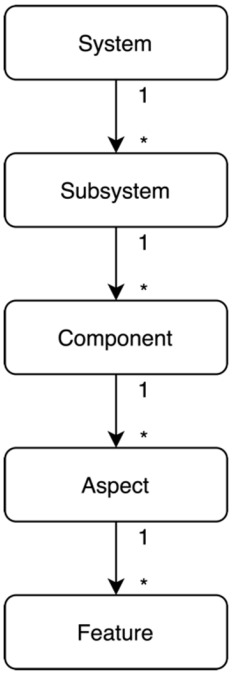

FIGURE 2. The hierarchical decomposition of blockchain systems.

In what follows, we describe our taxonomy for each subsystem and the relationships (Compositional or Alternative) between its various constituents.

B. EXECUTION ENVIRONMENT SUBSYSTEM

The Execution Environment subsystem consists of a Network, one or more Distributed Ledgers, and a Platform. Each of these components is decomposed into several aspects and each aspect is composed of one or more features.

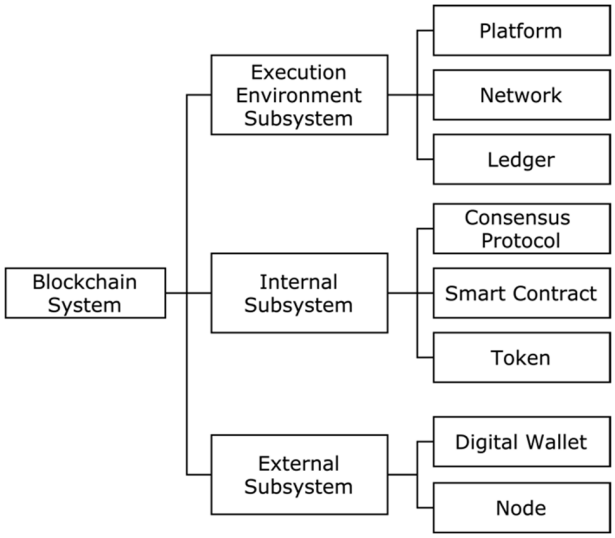

FIGURE 3. The high-level view of the proposed taxonomy.

The classification of this subsystem is depicted in Figure 4 and described as follows.

1) NETWORK

A blockchain system is based on a peer-to-peer (P2P) computer network. This network is characterized by six main aspects: Openness, Chain Structure, Access Control, Membership Entity, Member Registration, and Member Identity. Each of these aspects is characterized by a set of features. For example, the Openness aspect of the network is characterized by the features of Public, Private and Federated. Based on these features, a blockchain network can be either public, private or federated. Public networks are open-access and, typically, all the participating nodes can read from and write to their respective ledgers. Private networks are restricted and owned by an organization, where the permission to read and write is maintained within the organization's members. Federated networks are private networks shared by multiple organizations.

The Chain Structure aspect of the network is characterized by Single Chain and Multi Chain. Based on these features, a blockchain can be composed of a single chain or multiple chains. The remaining three aspects, Membership Entity, Member Registration, and Member Identify, are used to describe different characteristics of the participants of the blockchain applications, such as if a participant is a single organization or joint multiple organizations; if the identity of the participants is known to the network or not, as Figure 4 shows.

2) DISTRIBUTED LEDGER

Distributed ledgers are key to the blockchain technology. They record all the transactions committed by the participants of a blockchain network [28], [29]. The ledger component is characterized by four aspects: Ledger Architecture, Data Structure, Transaction Model, and Access Control. There are three types of ledger architecture [47]: Single-Ledger Based, Multi-Ledger Based, and Interoperability Based. These three architecture types are briefly described here:

The *Single-Ledger Based* architecture was introduced as a public permissionless solution to serve different application industries, such as supply chains and transportation. Two modifications were made to this public architecture to enable private and hybrid settings. First, a handshaking mechanism and a certificate authority are added to the architecture to solve the issue of openness in public settings. Thus, the architecture can fit into private settings. Second, a constellation component was introduced into the public architecture to make it applicable to hybrid settings [47].

The *Multi-Ledger Based* architecture is explicitly introduced for private and federated networks to allow confidential transactions within a consortium or single organization. This architecture involves more encryption and decryption operations than hybrid networks with single-ledger-based architectures. The *Interoperability Based* architecture has been introduced as a solution to the issue of interoperability between incompatible single-ledger-based blockchains. It can also be adapted for multi-ledger-based blockchains [47].

The data structure of a ledger can be classified as Chain-Based or Directed Acyclic Graph (DAG)-Based [28], [38], [48], [49]. Under the Chain-Based structure, a ledger is structured as a list of blocks. DAG-Based structure links data in a directed acyclic graph in a tree instead of a list.

The Transaction Model aspect has three features: Transaction-Based, Account-Based, and Token-Based. Under the stateless Transaction-Based model, tokens are stored in a list of unspent transaction outputs (UTXO). Existing UTXOs are used as inputs for new transactions, which subsequently produce new outputs in their place. Under the stateful Account-Based model, tokens are stored as a balance within an account controlled by a private key or a smart contract. under the stateful Token-Based model, each token has independent data space to store the complete history of the token's ownership.

3) PLATFORM

The blockchain platform provides an operational context and an execution environment of blockchain systems [50]. The platform component contains nine aspects, as Figure 4 shows. The Architectural Design aspect of the platform can be divided into three features: Monolithic, Polymorphic, and Modular. A monolithic platform offers services as a bundle with no distinctive architectural separation. A polymorphic platform provides a clear separation of different functions of the platform. A modular platform offers a mean of customization.

The Supported Solutions aspect characterizes if a blockchain platform is Permissioned or Permissionless.

The Energy Consumption aspect has two features: Power Intensive and Energy Saving. The Incentive Mechanism aspect has two sub-aspects: Fee and Reward. The Fee aspect

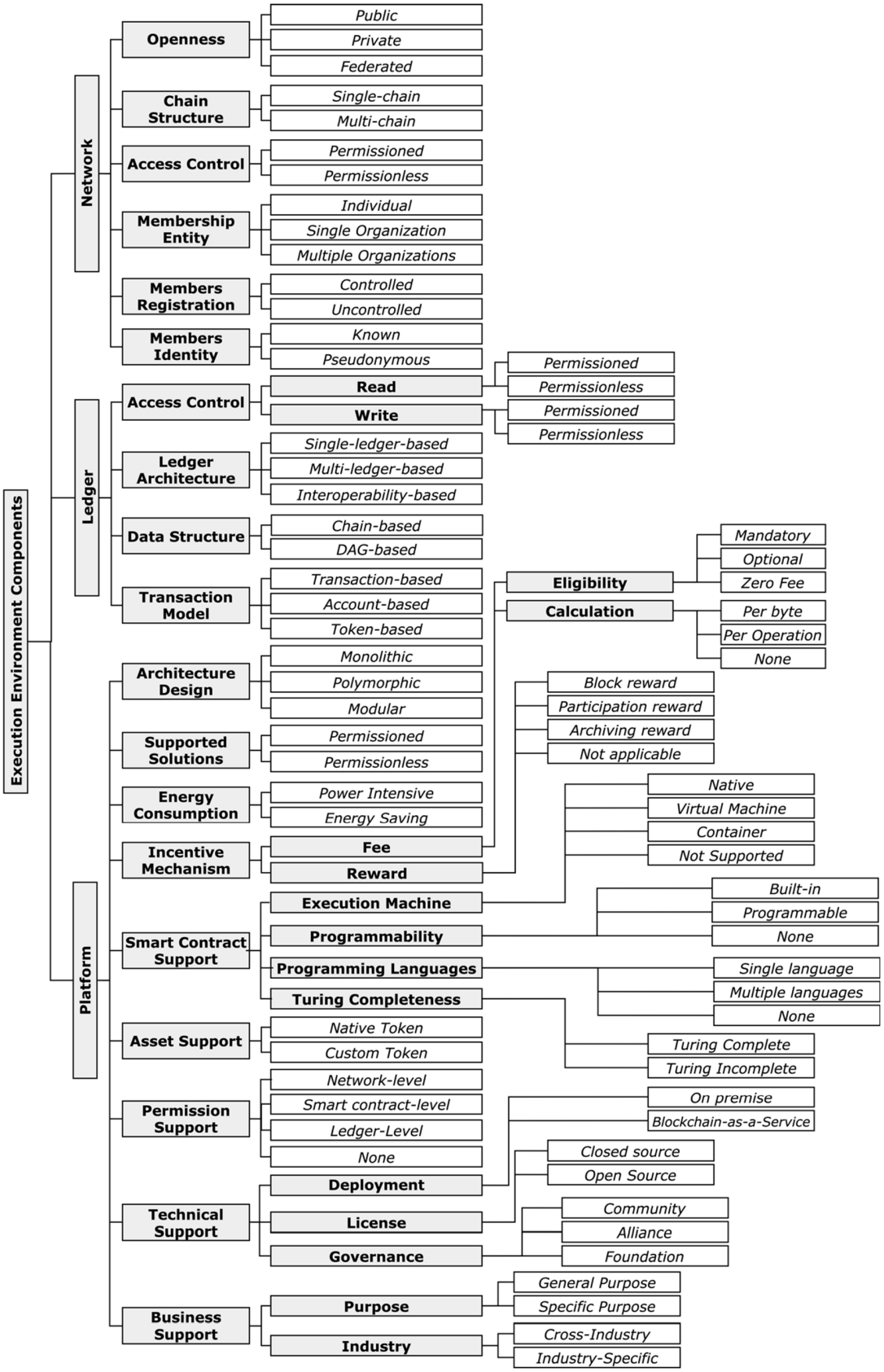

FIGURE 4. Classification of blockchain systems based on their execution environment subsystem components.

has two sub-aspects: Eligibility and Calculation. The Eligibility characterizes if transaction fees on a blockchain platform are Mandatory, Optional, or Zero. The Calculation determines if a fee is calculated Per Byte, Per Operation, or None. The Reward aspect has four features: Block Reward, Participation Reward, Archiving Reward, and Not Applicable.

If the platform supports the execution of smart contracts, it has specifications for the programming language and execution machines. The Smart Contract Support aspect contains four sub-aspects: Execution Machine, Programmability, Programming Languages, and Turing Completeness. The Execution Machine aspect has four features: Native, Virtual Machine, Container, and Not Supported. The Programmability aspect has three features: Built-in, Programmable, and None. The Programming Languages aspect has three features: Single Language, Multiple Languages, and None. The Turing Completeness aspect has two features: Turing Complete and Turing Incomplete.

The Asset Support aspect has two compositional features: Native Token and Custom Token. The Permission Support aspect characterizes different levels of permissions that are supported by a blockchain platform, if any. This aspect has four features: Network-level, Smart Contract-Level, Ledger-level, and None.

The Technical Support aspect consists of three sub-aspects: Deployment, License, and Governance. The Deployment aspect characterizes if a blockchain platform is deployed On Premise or Blockchain-as-a-Service. The License aspect characterizes if a blockchain platform is Open Source or Closed Source. The Governance aspect characterizes if a blockchain platform is supported by a Community, an Alliance, or a Foundation.

The Business Support aspect consists of two sub-aspects: Purpose and Industry. The Purpose aspect has two features: General Purpose and Specific Purpose. The Industry aspect has two features: Cross-Industry and Industry-Specific.

C. INTERNAL SUBSYSTEM

The Internal Subsystem is characterized by three main components: Consensus Protocol, Smart Contract and Token, as Figure 5 shows. These components and their classifications are described as follows.

1) CONSENSUS PROTOCOL

This component is described by four aspects: Consensus Mechanism, Finality, Network Access Control, and Trusted Hardware Utilization. The *Consensus Mechanism* implements the contractual agreement between the network members. This aspect has several features: Computation-Based relying on a node's computation power, Factor-Based relying on a factor to prioritize nodes for mining and validation, Voting-Based relying on the number of votes cast by peer nodes in the network, or Combination-Based built on different mechanisms, such as a combination of Factor-Based and Voting-Based.

The Finality aspect refers to the property in which a valid block cannot be pruned once it is appended to the blockchain. It has two features: Probabilistic and Absolute. Under this aspect, a consensus protocol can be characterized as Probabilistic or Absolute. Typically, the consensus finality is absolute in voting-based protocols, whereas it is probabilistic in computation-based protocols. The finality of the factor- and combination-based protocols is determined based on their implementation.

The Network Access Control aspect has two features: Permissioned and Permissionless. Different consensus mechanisms are applied in public and private network settings [51], [52]. Under this aspect, a compatible blockchain network can be characterized as a permissioned or permissionless.

The Trusted Hardware Utilization aspect indicates whether a specialized hardware is required for implementing the consensus mechanism, such as Intel SGX or ARM TrustZone. It has two features: Required and Not required.

2) SMART CONTRACT

A smart contract source code is deployed on the blockchain by passing its code as transaction data, and the code then becomes immutable. Each contract is represented in the blockchain using a unique address. This address is used to invoke functions inside the contract. These functions can examine conditions, express logic, create new contracts, or terminate their containing contract [28], [38]. A smart contract source code consists of functions and events. When a smart contract is terminated, the contract remains on the blockchain, although the contract no longer responds to transactions. This is evident because of the immutability of the blockchain, where the smart contract code is its data. This immutability of a smart contract gives it the characteristic of being a *firmware* program. For a smart contract to be updated, its current version must be terminated, and then a new version must be deployed on the blockchain.

It is worth mentioning that smart contracts should not be miscategorized as legal contracts. On one hand, a legal contract exists as an agreement between two or more parties. On the other hand, a computer program is a collection of functions and procedures within a source code executed by a computer machine. However, smart contracts are programs that can conditionally transfer digital assets between parties predictably and transparently, which may provide evidence of the ability of smart contracts to facilitate the execution of a legal contract [28].

This component is characterized by five aspects: Hosting Network, Type, Interaction, Events, and License. It is a programming language-agnostic classification that aims to describe concrete smart contract instances. The smart contract execution environment is discussed earlier in this paper as part of the blockchain platform. This taxonomy focuses on classifying concrete instances of smart contracts that deliver business logic to the blockchain systems. The Hosting Network aspect characterizes if a smart contract is deployed

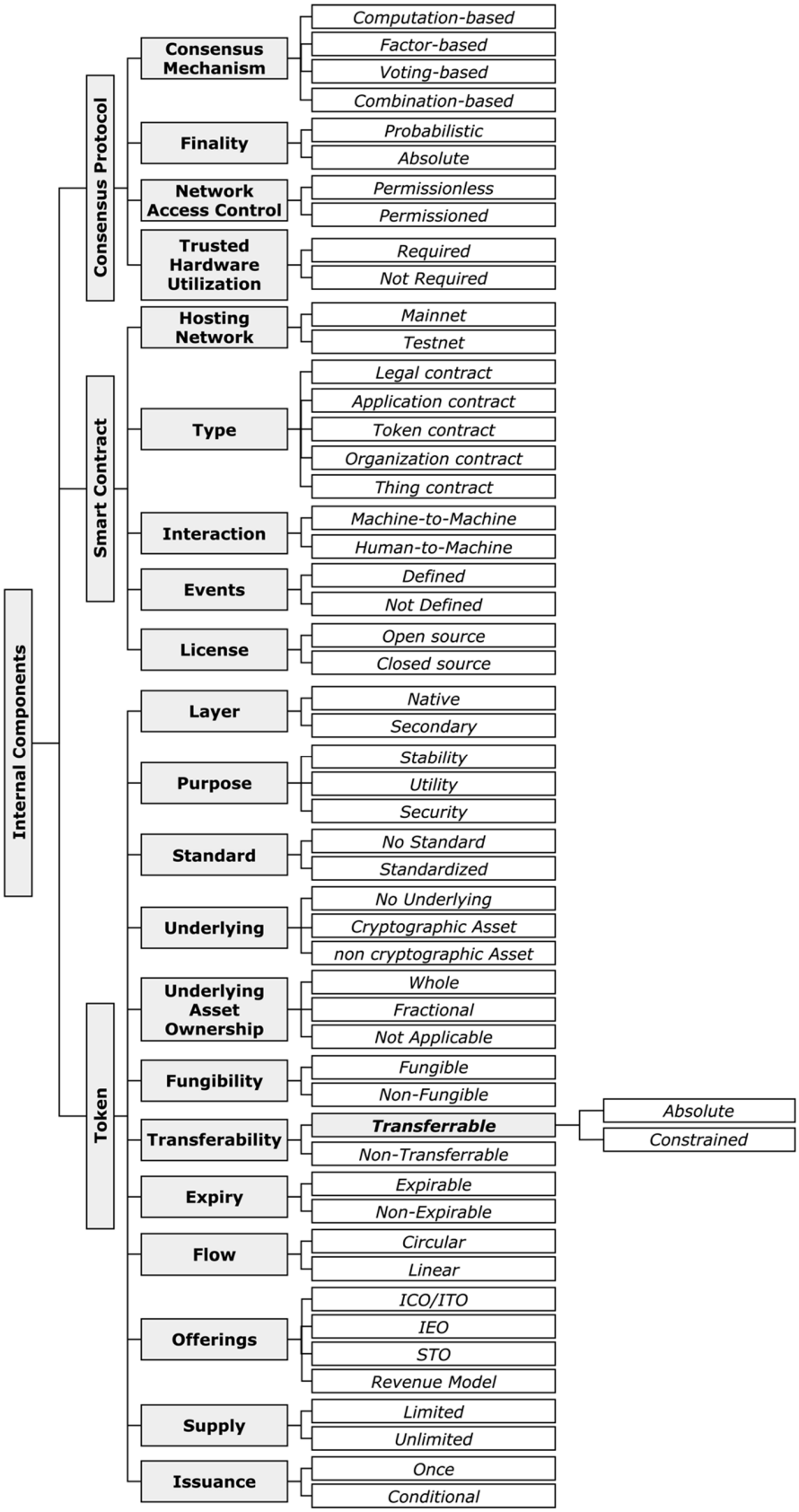

FIGURE 5. Classification of blockchain systems based on their internal subsystem components.

on a Mainnet or Testnet. The Type aspect characterize the business logic implemented in the smart contract. Under this aspect, a smart contract can be characterized as Legal, Application, Token, Organization, or Thing contract. Smart Legal contracts encode legally enforceable agreement terms among multiple parties as smart contracts on the blockchain [53]. Application contracts express the possible business logic and functions of DApps. Organization contracts refer to decentralized autonomous organizations (DAO). They are complex smart contracts that encode an organization's management and operational rules to allow autonomous operation without the need for a central authority [54]. Thing contracts are smart contracts used to register and manage IoT devices [55], [56].

The Interaction aspect has two features: Machine-to-Machine and Human-to-Machine. In the former, smart contracts are invoked and executed by node devices with no human interactions, such as contracting devices in the IoT. In the latter, the invocation of a smart contract involves human interaction, such as in the application and smart legal contracts. These interactions can be recorded in terms of the events.

The Event aspect indicates if there are any events defined by the programmer in the smart contract. It has two features: Defined and Not defined. The License aspect characterizes if a smart contract is open- or closed-source.

3) TOKEN

Blockchain systems offer the distinctive ability to record the existence and transfer of digital assets [28]. These assets are known as tokens and can represent real-world objects [38]. Numerous taxonomies of digital assets have been proposed in the literature as well as in the industry, such as those by the International Token Standardization Association[6] and Enterprise Ethereum Alliance.[7] In addition, a token design decision tree has been proposed to aid the design process of digital tokens when developing blockchain applications [23]. This component is characterized by 13 aspects: Layer, Purpose, Standard, Fungibility, Underlying, Underlying Asset Ownership, Transferability, Expiry, Flow, Offerings, Supply, and Issuance.

The Layer aspect considers the technical foundation of the token. It has two features: Native and Secondary. Native tokens are platform-specific tokens that are implemented as a core component of the platform, mostly as a cryptocurrency, The second-layer tokens support the application functions and are called application tokens. Application tokens can be created if the platform supports programmable smart contracts. In some cases, these tokens are implemented on side-chains rather than on the main chain. Hence, they are called side-chain tokens. An application token or a side-chain token; is a secondary token implemented on top of the blockchain main layer.

The Purpose aspect has three features: Utility, Stability, and Security. The Standard aspect considers if the token is implemented according to a token standard [57], such as Ethereum Request for Comments 20 (ERC20)[8] and Simple Asset Standard [58]. This aspect has two features: Standardized and No Standard. The Fungibility aspect characterizes a token as Fungible or Non-fungible.

The Underlying aspect characterizes the assets represented by a token, if any. It has three features: No Underlying, A Cryptographic Asset, and Non-Cryptographic Asset. In addition, because tokens can represent real-world assets, the ownership of the underlying assets should be considered. The Underlying Asset Ownership aspect refers to the divisibility of the assets represented by the token. For example, multiple people can own a piece of art. Whole ownership means that an asset is owned by one party. By contrast, fractional ownership means that the asset is logically divided into pieces, and a different party owns each piece. Tokens in the blockchain represent this ownership. This aspect has three features: Whole, Fractional, and Not Applicable.

The Transferability, Expiry, and Flow aspects consider the circulation of a token. The Transferability aspect has two features: Transferrable and Non-transferrable. The Transferrable feature has two sub-features: Absolute and Constrained. The Expiry aspect characterizes the lifetime of the token. Under this aspect, a token is characterized as Expirable or Non-Expirable. The Flow aspect has two features: Linear and Circular.

The Offerings, Supply, and Issuance aspects consider economical foundation of a token. For example, some tokens are issued in a limited or unlimited manner, some are issued when a condition is met, such as mining, whereas others are issued all at once in advance [57], [59], [60], [61]. The Offering aspect has four alternative features: Initial Coin—Or *Token*—Offering (ICO/ITO), Initial Exchange Offering (IEO), Security Token Offering (STO), and Revenue Models. The Supply aspect has two alternative features: Limited and Unlimited. The Issuance aspect has two alternative features: Once and Conditional.

D. EXTERNAL SUBSYSTEM

The External Subsystem is characterized by two main components: Node and Digital Wallet, as Figure 6 shows. These components and their classifications are described as follows.

1) NODE

Nodes are the fundamental elements and communication devices in blockchain networks. They retain ledgers, host smart contracts, execute transactions, and generate blocks.

Ideally, a single-blockchain network can have multiple node types. However, logically, a single full node can operate the network. Nodes and clients are closely related. A node is a hardware device (e.g., computer system) in the network whereas a client provides the interface for the node.

[6]https://itsa.global/

[7]https://entethalliance.org/

[8]https://eips.ethereum.org/EIPS/eip-20

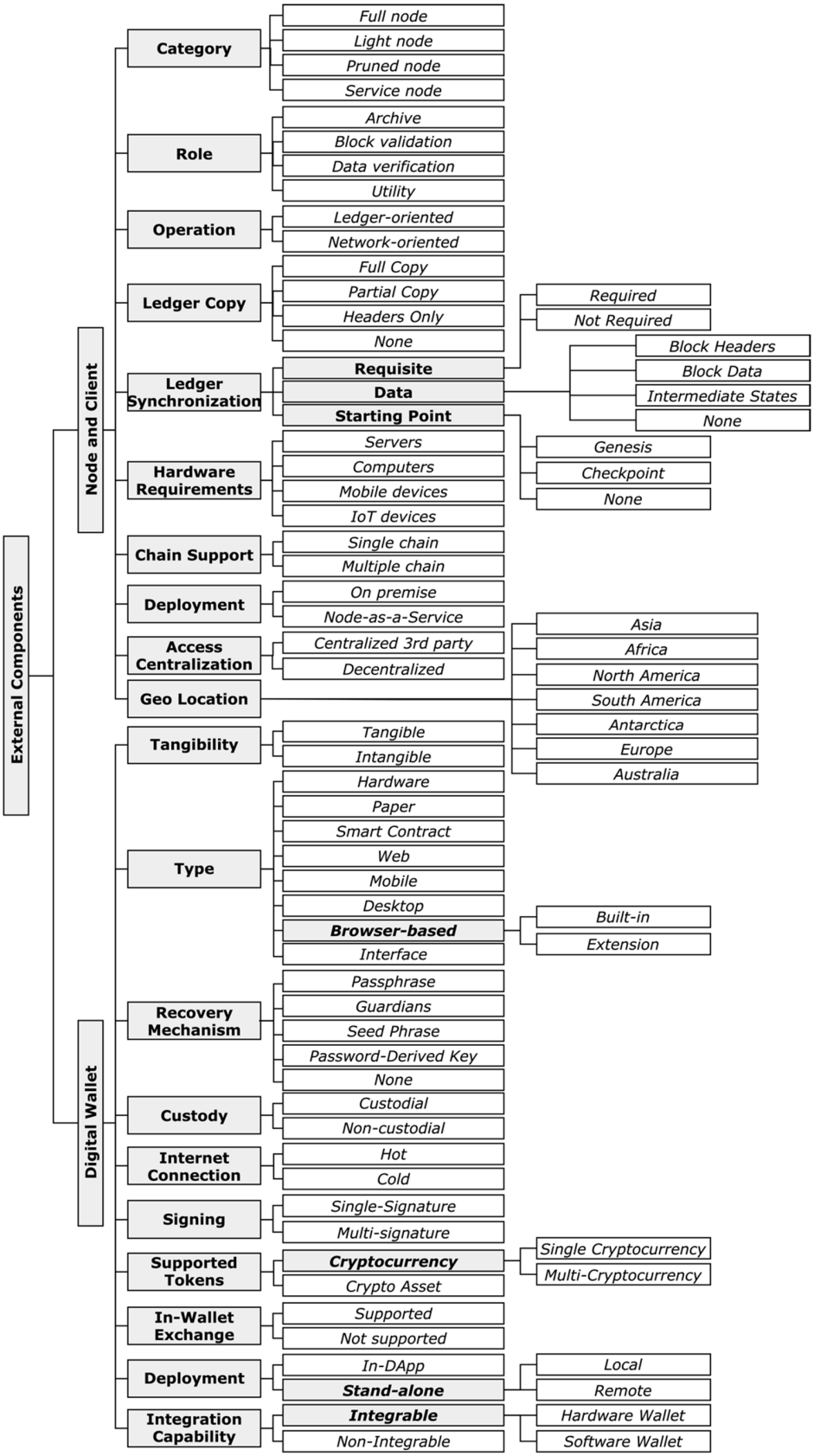

FIGURE 6. Classification of blockchain systems based on their external subsystem components.

For example, Raspberry Pi4 is a node hardware where Geth Ethereum client software can be installed.

This component is characterized by 10 aspects: Category, Role, Operation, Ledger Copy, Ledger Synchronization, Hardware Requirements, Chain Support, Deployment, Access Centralization, and Geolocation.

The Category aspect has four features: Full, Light, Pruned, and Service. The Full nodes are responsible for supporting consensus and verifying transactions or mining, as they store a full copy of the blockchain. The Light nodes rely on full nodes to acquire the required information when communicating with the blockchain. The Pruned nodes store a full copy of a subset of transactions according to a predefined limit and store transaction headers for old transactions. Finally, the Service nodes coordinate the workflow between the network's nodes and do not update the ledger.

The Role aspect has four features: Archive, Block Validation, Data Verification, and Utility. Archive nodes are full nodes that store a complete historical copy of the ledger. Validator nodes are full nodes responsible for consensus and block generation, such as miner and notary nodes. The data verification nodes are light nodes that do not validate blocks. Thus, they need only transaction headers to verify transactions as in simple-payment-verification clients, or possibly no data are synchronized as in minimal verification clients such as incubed (IN3) clients. Finally, service nodes provide utility to the network in aspects other than ledger-oriented operations. For example, proxy nodes in the Klaytn network coordinate the transmission of information between other nodes in the Klaytn network.

The Operation aspect characterizes the operations performed by a node. It has two features: Ledger-Oriented and Network-Oriented. The full, pruned, and light nodes perform ledger-oriented operations that uses and update ledger data.

The Ledger Copy aspect has four features: Full, Partial, Headers only, and None. In contrast to full and pruned nodes, light nodes do not store a copy of blockchains; thus, they cannot validate the blocks. However, they can broadcast transactions and query the blockchain [45], [62]. Although archive and validator nodes store a full copy of the ledger, the data stored in archive nodes include intermediate states to build the history of the blockchain.

Nodes may need to maintain a copy of their ledger according to their role. Thus, they need to be synchronized in different ways [63], [64], [65]. Regardless of the approach adopted, data to be synchronized and synchronization starting point are fundamental aspects from a classification perspective. The Ledger Synchronization aspect has three sub-aspects: Requisite, Data, and Starting Point of synchronization. The Requisite aspect has two features: Required and Not Required. The Data aspect has four aspects: Block Header, Block Data, Intermediate States, and None. The Starting Point aspect has three features: Genesis, Checkpoint, and None.

Nodes can be any electronic device with sufficient power and technical requirements to perform a job on the network. The Hardware Utilization aspect characterizes the machine used for a node. Under this aspect, a node is characterized as Servers, Computers, Mobile Devices, or IoT Devices. The Chain Support aspect consider the number of networks a node can operate. This aspect has two alternative features: Single Chain and Multiple Chains.

Nodes and clients can be deployed locally on a user's qualified machine, remotely on the cloud in a kind of centralization as the cloud provider controls it, or on a decentralized preconfigured node. Multiple nodes of a single network can be in different geographical locations. The Deployment aspect has two features: On Premise and Node-as-a-Service. The Access Centralization aspect has two features: Centralized 3rd Party and Decentralized. The Geolocation aspect has seven features: Asia, Africa, North America, South America, Antarctica, Europe, and Australia.

2) DIGITAL WALLET

A digital wallet is used to manage digital assets in blockchains. This component is characterized by 10 aspects: Tangibility, Type, Recovery Mechanism, Custody, Internet Connection, Signing, Supported Tokens, In-Wallet Exchange, Deployment, and Integration Capability. The Tangibility and Type aspects describe the nature of a digital wallet. The Tangibility aspect has two features: Tangible and Intangible. The Type aspect has eight features: Hardware, Paper, Smart Contract, Web, Mobile, Desktop, Browser-based, and Interface. The Browser-based feature has two sub-features: Built-in and Extension. Tangible digital wallets include hardware and paper wallets. The hardware wallets are devices used to store public and private keys in an air gapped environment. A paper wallet is simply a piece of paper with encrypted digital keys printed on it. Intangible wallets are software programs. Smart contract wallets use smart contracts to store assets, adding a unique capability for security and recovery [66].

A user's private key is the only way to manage their digital assets. If lost, it cannot be recovered unless the anticipated wallet has a mechanism for key recovery. The Recovery Mechanism aspect has five features: Passphrase, Guardian, Seed Phrase, Password-Derived Keys, and None.

The Custody aspect has two features: Custodial and Non-Custodial. Custodial wallets are centralized key stores in which users' private and public keys are stored on a server. Users must provide valid passwords to access their wallets and retrieve their keys to manage their own digital assets. In contrast, non-custodial wallets do not save users' data remotely. Instead, public and private keys are stored in a user-managed storage.

The Internet Connection aspect has two features: Hot and Cold. Whenever a wallet requires an Internet connection to be accessed, it is a hot wallet. This is in contrast with cold wallets, which are air gapped. For example, hardware and paper wallets are cold wallets, whereas software wallets are hot ones. However, there are situations in which software wallets support cold storage when used in an air-gapped environment.

The Signing, Supported Tokens, and In-Wallet Exchange aspects consider the transaction management. Although typical blockchain transactions require only one signature, there are situations in which multiple signatures are required for a single transaction. Hence, a multi-signature (or multi-sig) wallet is required. The Signing aspect has two features: Single-Signature and Multi-Signature.

A digital wallet can support one or more types of digital assets. The Supported Tokens aspect has two features: Cryptocurrency, which is further categorized as Single Cryptocurrency and Multi Cryptocurrency, and Crypto Assets, i.e., non-cryptocurrency tokens. In addition, some wallets are capable of token exchange. The In-Wallet Exchange aspect has two features: Supported and Not Supported.

The Deployment aspect considers where a wallet is deployed. Under this aspect, a digital wallet is characterized as In-App or Stand-alone. Stand-alone wallets are further categorized as Local or Remote. Digital wallets can be integrated with other wallets. The Integration Capability aspect has two features: Integrable and Non-Integrable. The Integrable feature characterizes if the integrated wallet is a Hardware Wallet or Software Wallet.

V. DEMONSTRATING THE PROPOSED TAXONOMY

In this section, we demonstrate our taxonomy by using it to characterize real-world blockchain systems. Based on the design science methodology [67], in the following subsections, we first characterize individual software components in different blockchain systems; we then provide a case study for characterizing a specific blockchain system.

A. CHARACTERIZING INDIVIDUAL BLOCKCHAIN COMPONENTS USING THE PROPOSED TAXONOMY

In this subsection, the proposed taxonomy is used to classify different software components in 80 different blockchain application systems and components we found from the Internet. They are listed in Table 2.

1) CHARATERIZING EXECUTION ENVIRONMENT COMPONENTS

a: NETWORK

From the list of blockchain systems presented in Table 2, we identified 10 state-of-the-art blockchain networks for characterization. The results of the characterization using our taxonomy are presented in Table 3. Table 3 shows that, in terms of the *Openness*, out of the 10 networks, five are public networks (Bitcoin, Etherum, Ardor Parent Chain, Ardor Child Chain, and Klaytn Service Chain), two are purely federated (Corda and Hyperledger Iroha), one is private only (Ardor Private Child Chain), and two allow for both private and federated implementations (Hyperledger Fabric and Klaytn Service Chain). With respect to the *Chain Structure*, three out of the five (3/5) public networks and one out of the four (1/4) federated networks are single chains, whereas the other six networks are multi-chained. For the *Membership Entity*, five networks are open for individuals, three are for single organizations, and four are for multiple organizations. Finally, for the *Access Control*, *Member Registration*, and *Member Identity*, we can see an equal distribution of the features for each network. For example, for the Access Control aspect of the network, five networks require permission whereas another five do not.

b: DISTRIBUTED LEDGER

From the list of blockchain systems presented in Table 2, we identified 10 state-of-the-art blockchain distributed ledgers for characterization. The results of the characterization using our taxonomy are presented in Table 4. For conciseness, the analysis of this table is omitted, but a similar analysis as for Table 3 applies.

c: PLATFORM

From the list of blockchain systems presented in Table 2, we identified 10 state-of-the-art blockchain platforms for characterization. The results of the characterization using our taxonomy are presented in Table 5. In Table 5, for the *Architectural Design* aspect, four platforms are Monolithic, four are Polymorphic, and two are Modular. For example, Hyperledger Sawtooth is a Modular Platform for enterprise solutions.

For the *Supported Solution*, six support the implementation of Permissioned applications, three support Permissionless applications, and one supports both. Furthermore, of the seven platforms supporting permissioned solutions, one supports permissions on all the three permission levels, one platform supports only network-level permissions, and five platforms support network and ledger permissions.

In term of *Energy Use*, eight platforms are based on Power Saving, whereas only two are Power Intensive. For the *Incentive Mechanism*, six platforms calculate transaction fees per operation, where the fees are mandatory in five other platforms. Whereas the Reward mechanism is not applicable to five platforms, block rewards are applicable to four other platforms, participation rewards are applicable to two platforms, and archiving rewards are applicable to only one platform.

For the *Smart Contract Support*, seven platforms support programable smart contracts, of which five execute such contracts in virtual machines, whereas two execute them in containers. For the Asset Support, two platforms support only native tokens, five platforms support only custom tokens, and three platforms support both native and custom tokens. For the *Business Support*, all the 10 platforms are cross-industry, including seven general-purpose and three purpose-specific platforms.

Finally, for the *Technical Support*, nine platforms are open-source and to be deployed on premises, whereas seven of them are governed by a community of developers, one is governed by an alliance, and one is governed by a foundation. IBM Blockchain is the only closed-source platform.

TABLE 2. Blockchain systems and components in the evaluation sample (Y=Yes, N=No).

Blockchain System/Component	Network	Ledger	Platform	Consensus Protocol	Smart Contract	Token	Clients & Nodes	Digital Wallet
Bitcoin (https://bitcoin.org/en/)	Y	Y	N	N	N	N	N	N
Ethereum (https://ethereum.org/)	Y	Y	Y	N	N	Y	N	N
Hyperledger Fabric (https://www.hyperledger.org/use/fabric)	Y	Y	Y	N	N	N	N	N
Corda (https://www.corda.net/)	Y	Y	Y	N	N	N	N	N
Hyperledger Iroha (https://www.hyperledger.org/use/iroha)	Y	N	Y	N	N	N	N	N
Ardor (https://www.jelurida.com/ardor)	Y	N	Y	N	N	N	N	N
Klaytn (https://www.klaytn.foundation/)	Y	Y	Y	N	N	N	N	N
Hyperledger Indy (https://www.hyperledger.org/use/hyperledger-indy)	N	Y	N	N	N	N	N	N
Steem (https://steem.com/)	N	Y	Y	N	N	Y	N	N
Elements (https://elementsproject.org/)	N	Y	N	N	N	N	N	N
everiToken (https://www.everitoken.io/)	N	Y	N	N	N	N	N	N
IOTA Tangle (https://www.iota.org/)	N	Y	Y	N	N	N	N	N
IBM Blockchain (https://www.ibm.com/blockchain/platform)	N	N	Y	N	N	N	N	N
Hyperledger Sawtooth (https://www.hyperledger.org/use/sawtooth)	N	N	Y	N	N	N	N	N
Proof of Work [68]	N	N	N	Y	N	N	N	N
Prime Number Proof of Work [69]	N	N	N	Y	N	N	N	N
Proof of Stake (https://eprint.iacr.org/2020/037.pdf)	N	N	N	Y	N	N	N	N
Delegated Proof of Stake [70]	N	N	N	Y	N	N	N	N
Proof of Authority (https://github.com/ethereum/guide/blob/master/poa.md)	N	N	N	Y	N	N	N	N
Proof of Elapsed Time (https://eprint.iacr.org/2021/086.pdf)	N	N	N	Y	N	N	N	N
Practical Byzantine Fault Tolerance [71]	N	N	N	Y	N	N	N	N
Tendermint [72]	N	N	N	Y	N	N	N	N
Ripple Federated Byzantine Agreement [73]	N	N	N	Y	N	N	N	N
Stellar [74]	N	N	N	Y	N	N	N	N
Proof of Activity [75]	N	N	N	Y	N	N	N	N
Nexo DApp (https://nexo.io/)	N	N	N	N	Y	N	N	N
Meme Factory DApp (https://memefactory.io/)	N	N	N	N	Y	N	N	N
Brave Frontier Heroes DApp (https://bravefrontierheroes.com/)	N	N	N	N	Y	N	N	N
Decentraland DApp (https://decentraland.org/)	N	N	N	N	Y	N	N	N
Broker and NTUA [76]	N	N	N	N	Y	N	N	N
CryptoKitties DApp (https://www.cryptokitties.co/)	N	N	N	N	Y	Y	N	N
Dtube DApp (https://d.tube/)	N	N	N	N	N	Y	N	N
ActiFit DApp (https://actifit.io/)	N	N	N	N	N	Y	N	N
Geon DApp (https://www.geon.network/)	N	N	N	N	N	Y	N	Y
SPiCE VC (https://www.spicevc.com/)	N	N	N	N	N	Y	N	N
Blockimmo DApp (https://blockimmo.ch/)	N	N	N	N	N	Y	N	N
Publicly Accessible Environment Sensor use case (https://in3.readthedocs.io/en/develop/intro.html)	N	N	N	N	N	N	Y	N
Ledger Nano S (https://www.ledger.com/)	N	N	N	N	N	N	N	Y
Trezor One (https://trezor.io/)	N	N	N	N	N	N	N	Y
My Ether Wallet (https://www.myetherwallet.com/)	N	N	N	N	N	N	N	Y
MetaMask (https://metamask.io/)	N	N	N	N	N	N	N	Y
Electrum (https://electrum.org)	N	N	N	N	N	N	N	Y
Coinbase.com (https://www.coinbase.com/)	N	N	N	N	N	N	N	Y
Authereum (https://authereum.com/)	N	N	N	N	N	N	N	Y
Parity signer (https://www.parity.io/technologies/signer/)	N	N	N	N	N	N	N	Y
Argent (https://www.argent.xyz/))	N	N	N	N	N	N	N	Y
MEW App (https://www.mewwallet.com/)	N	N	N	N	N	N	N	Y
Coinbase Wallet (https://wallet.coinbase.com/faq/)	N	N	N	N	N	N	N	Y
Opera Touch (https://www.opera.com/crypto/next)	N	N	N	N	N	N	N	Y

TABLE 3. Characterizing 10 real-world blockchain networks using the proposed taxonomy.

Blockchain Network	Openness†			Chain Structure		Access Control		Membership Entity†			Members Registration		Members Identity	
	Public	Private	Federated	Single chain	Multi-chain	Permissionless	Permissioned	Individual	Single Organization	Multiple Organizations	Uncontrolled	Controlled	Pseudonymous	Known
Bitcoin	Y	N	N	Y	N	Y	N	Y	N	N	Y	N	Y	N
Ethereum	Y	N	N	Y	N	Y	N	Y	N	N	Y	N	Y	N
Hyperledger Fabric	N	Y	Y	N	Y	N	Y	N	Y	Y	N	Y	N	Y
Corda	N	N	Y	N	Y	N	Y	N	N	Y	N	Y	N	Y
Hyperledger Iroha	N	N	Y	Y	N	N	Y	N	N	Y	N	Y	N	Y
Ardor parent chain	Y	N	N	N	Y	Y	N	Y	N	N	Y	N	Y	N
Ardor public child chain	Y	N	N	N	Y	Y	N	Y	N	N	Y	N	Y	N
Ardor private child chain	N	Y	N	N	Y	N	Y	N	Y	N	N	Y	N	Y
Klaytn Main Chain	Y	N	N	Y	N	Y	N	Y	N	N	Y	N	Y	N
Klaytn Service Chain	N	Y	Y	N	Y	N	Y	N	Y	Y	N	Y	N	Y

† Compositional features. Y=Yes, N=No.

TABLE 4. Characterizing 10 real-world distributed ledgers using the proposed taxonomy.

Distributed Ledger	Access Control				Ledger Architecture			Data Structure		Transaction Model		
	Read		Write									
	Permissioned	Permissionless	Permissioned	Permissionless	Single-ledger-based	Multi-ledger-based	Interoperability-based	Chain-based	DAG-based	Transaction-based	Account-based	Token-based
Bitcoin	N	Y	N	Y	Y	N	N	Y	N	Y	N	N
Ethereum	N	Y	N	Y	Y	N	N	Y	N	N	Y	N
Hyperledger Fabric	Y	N	Y	N	N	Y	N	Y	N	N	Y	N
Hyperledger Indy	Y	N	Y	N	N	N	Y	Y	N	N	Y	N
Steem	N	Y	N	Y	Y	N	N	Y	N	N	Y	N
Klaytn	N	Y	N	Y	Y	N	N	Y	N	N	Y	N
Elements	N	Y	N	Y	N	N	Y	Y	N	Y	N	N
everiToken	N	Y	N	Y	Y	N	N	Y	N	N	N	Y
Corda	Y	N	Y	N	N	Y	N	N	Y	Y	N	N
Tangle	N	Y	N	Y	Y	N	N	N	Y	N	Y	N

Y=Yes, N=No.

2) CHARATERIZING INTERNAL COMPONENTS

a: CONSENSUS PROTOCOL

From the list presented in Table 2, we identified 11 state-of-the-art consensus protocols for characterization. The results of the characterization using our taxonomy are presented in Table 6. Of these protocols, two are computation-based with probabilistic finality for permissionless networks, one is combination-based with probabilistic finality for permissionless networks, four are voting-based with absolute finality for permissioned networks, and four are factor-based, including

TABLE 5. Characterizing 10 real-world blockchain platforms using the proposed taxonomy.

Platform	Architecture Design			Supported Solution†		Energy Use		Incentive Mechanism									Smart Contract Support													Asset Support†		Permission Support†				Technical Support								Business Support			
								Fee						Reward†				Execution Machine				Program-ability			Programming Language			Turing Completeness																			
								Eligibility†			Calculation																																						
	Monolithic	Polymorphic	Modular	Permissioned	Permissionless	Power intensive	Power saving	Mandatory	Optional	Zero	Per byte	Per operation	None	Block reward	Participation reward	Archiving reward	Not applicable	Native	Virtual machine	Container	Not supported	Built-in	Programmable	None	Single language	Multiple languages	None	Turing complete	Turing incomplete	Native token	Custom token	Network-level	Smart contract-level	Ledger-level	None	On premises	Blockchain-as-a-Service	Open-source	Closed source	Community	Alliance	Foundation	General	Specific	Cross-industry	Industry-specific
Ethereum	Y	N	N	N	Y	Y	N	Y	N	N	N	Y	N	Y	N	N	N	N	Y	N	N	N	Y	N	Y	N	N	Y	N	Y	Y	N	N	N	Y	Y	N	Y	N	Y	N	N	Y	N	Y	N
Hyperledger Fabric	N	N	Y	Y	N	N	Y	N	N	Y	N	N	Y	N	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	N	Y	N	Y	N
Hyperledger Iroha	N	N	Y	Y	N	N	Y	N	N	N	N	Y	N	N	N	N	Y	Y	N	N	N	N	N	N	N	N	Y	Y	N	N	Y	Y	Y	N	Y	N	Y	N	Y	N	N	Y	N	Y	N
IOTA	Y	N	N	N	Y	Y	N	N	N	Y	N	N	Y	N	N	N	Y	N	N	N	Y	N	N	Y	N	N	Y	N	Y	Y	N	N	N	N	Y	Y	N	Y	N	N	N	Y	N	Y	Y	N
Steem	N	Y	N	N	Y	N	Y	N	N	Y	N	N	Y	Y	Y	N	N	Y	N	N	N	Y	N	N	N	N	Y	Y	N	Y	Y	N	N	N	Y	Y	N	Y	N	Y	N	N	N	Y	Y	N
IBM Blockchain	N	Y	N	Y	N	N	Y	Y	N	N	N	Y	N	N	N	N	N	N	N	Y	N	N	Y	N	N	Y	N	Y	N	N	Y	Y	N	Y	N	N	Y	N	Y	N	N	Y	Y	N	Y	N
Klaytn	Y	N	N	Y	N	N	Y	Y	N	N	N	Y	N	Y	N	N	N	N	N	Y	N	N	Y	N	Y	N	N	N	N	Y	N	Y	N	N	N	Y	N	Y	N	Y	N	N	Y	N	Y	N
Hyperledger Sawtooth	N	Y	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	N	Y	N	Y	N	N	N	Y	N	N	Y	N	Y	N	N	Y	Y	N	Y	N	Y	N	Y	N	Y	N	N	Y	N	Y	N
Corda	Y	N	N	Y	N	N	Y	Y	N	N	N	Y	N	N	N	N	Y	N	Y	N	N	N	Y	N	N	Y	N	Y	N	N	Y	Y	N	Y	N	Y	N	N	Y	Y	N	N	Y	Y	N
Ardor	N	Y	N	Y	Y	N	Y	Y	N	N	N	N	N	Y	Y	Y	N	N	Y	N	N	N	Y	N	N	Y	N	Y	N	Y	Y	Y	N	Y	N	Y	N	Y	N	N	Y	N	Y	N

† Compositional features. Y=Yes, N=No.

one with absolute finality for both permissioned and permissionless networks, and three with probabilistic finality, where two of them are for permissionless networks and one is for both permissioned and permissionless networks. Only one protocol requires the utilization of trusted hardware.

b: SMART CONTRACT

Table 7 presents the characterization of 11 smart contracts listed in Table 2. Of the 11 smart contracts given in Table 7, six are application contracts deployed on a Mainnet for Human-to-Machine interactions, four are token contracts for Human-to-Machine interactions including three are deployed on a Mainnet and one is deployed on a Testnet, and one is a thing contract deployed on a Testnet for Machine-to-Machine interactions. All the 11 smart contracts are open source and include user-defined events.

c: TOKEN

We randomly selected 11 tokens from the systems listed in Table 2. The characterization of these tokens is presented in Table 8. Some of these tokens are briefly introduced here: Ether (ETH) is the cryptocurrency of Ethereum. CryptoKitties CK is the application token of CryptoKitties DApp. STEEM, Steem Power SP, and Steem Dollar tokens are fundamental asset classes of the Steem blockchain. DTC is the application token of Dtube DApp, a Steem-based application for video sharing. AFIT is the application token of ActiFit, a Steem-based mobile DApp for promoting healthy habits. Geon coin GC and Geon token GT are application tokens of Geon, an augmented reality application provides location-based incentives by rewarding its users for physical presence and other real-world location activities. SPiCE is a security token provided by SPICE VC, a venture capital firm that uses the blockchain technology to solve the liquidity problem. IMMO is the application token of Blockimmo, a decentralized marketplace for tokenized property in Switzerland.

Of these 11 tokens, five are Native Tokens, of which three are for utility, one for stability, and one for security. The remaining six tokens are Secondary, including four for utility, one for stability, and one for security. Of the 11 tokens, eight are Standardized Fungible Tokens, two are Non-Standardized, and one is a Standardized Non-Fungible token. Six tokens are not the representations of an underlying asset, whereas two represent wholly ownable cryptographic assets, two represent wholly ownable non-cryptographic assets, and one represents fractionally ownable non-cryptographic assets. In terms of the Expiry aspect, nine tokens are non-expirable where eight are transferable in a circular flow and one is constrainedly transferable in circular flow. Two of the 11 tokens are expirable and transferable in circular flow absolutely for one and constrainedly for the other.

For the four tokens offered via ICO\ITO, one token has a limited supply issued all at once, and three tokens have an unlimited supply issued conditionally. All three tokens offered via a revenue model are issued conditionally, where two tokens have limited supply and one token has an unlimited supply. The two tokens offered via STO have a limited supply, where one is issued conditionally, and the other is issued all at once. The last two tokens are offered via IEO and issued conditionally with a limited supply for one token and an unlimited supply for the other token.

3) CHARATERIZING EXTERNAL COMPONENTS

a: NODE

A blockchain network usually contains multiple nodes. Based on the client environment, each node may have different characteristics. Thus, nodes as individual components are

TABLE 6. Characterizing 11 consensus protocols in real-world systems using the proposed taxonomy.

Consensus Protocol	Consensus Mechanism				Finality		Network Access Control†		Trusted Hardware Utilization	
	Computation-based	Factor-based	Voting-based	Combination-based	Probabilistic	Absolute	Permissioned	Permissionless	Required	Not required
Proof of Work	Y	N	N	N	Y	N	N	Y	N	Y
Prime Number Proof of Work	Y	N	N	N	Y	N	N	Y	N	Y
Proof of Stake	N	Y	N	N	Y	N	N	Y	N	Y
Delegated Proof of Stake	N	Y	N	N	Y	N	N	Y	N	Y
Proof of Authority	N	Y	N	N	N	Y	Y	Y	N	Y
Proof of Elapsed Time	N	Y	N	N	Y	N	Y	Y	Y	N
Practical Byzantine Fault Tolerance	N	N	Y	N	N	Y	Y	N	N	Y
Tendermint	N	N	Y	N	N	Y	Y	N	N	Y
Ripple Federated Byzantine Agreement	N	N	Y	N	N	Y	Y	N	N	Y
Stellar	N	N	Y	N	N	Y	Y	N	N	Y
Proof of Activity	N	N	N	Y	Y	N	N	Y	N	Y

† Compositional features. Y=Yes, N=No.

typically classified by their owners, as the owners have the knowledge of the client environment. In this paper, we characterize the network nodes through the Publicly Accessible Environment Sensor Use Case from IN3 Network.[9] This use case describes the use of different nodes and clients in IoT applications in the city of Stuttgart. The characterization of the primary nodes in the use case is given in Table 9.

Of the four nodes, one performs ledger-oriented operations, two perform network-oriented operations, and one performs both operations. For the chain support, one node supports a single chain which is a light node, and three nodes support multiple chains including two service nodes and one full node. Of the four nodes, two are servers, one is a computer, and the other is an IoT device. One node requires synchronization of the block data from the genesis block to participate in block validation. Another node requires synchronization of block headers from a checkpoint to participate in data verification. The remaining two nodes do not require ledger data. On premises and node-as-a-service deployments are in an equal measure. The access to all the four nodes is decentralized.

b: DIGITAL WALLET

An example of applying the digital wallet features to a set of state-of-the-art digital wallets is presented in Table 10. Of the 13 wallets, two are tangible hardware and 11 are intangible. Of the 11 intangible wallets, three are mobile app wallets, two are web app wallets, two are smart contract wallets, and one wallet for each other types. For the recovery mechanism, the seed phrase has a significant number of eight wallets, three wallets have no recovery mechanism, and the other mechanisms have equal measures of one wallet each. Most of the wallets are non-custodial, totaling 12 out of the 13 wallets. Out of 13, seven wallets support both multi-currency and crypto-assets, five wallets support only cryptocurrencies including one wallet supports only a single cryptocurrency, and one wallet supports only crypto-assets. Only one of the 13 wallets supports multiple signatures. The in-wallet exchange is supported by seven wallets out of the 13. For the wallet deployment, eight wallets support local deployment, two wallets support remote deployment, and two wallets support both deployments. For the integration capability, six out of the 13 wallets are non-integrable, six are integrable with other software wallets, and five wallets are integrable with other hardware wallets.

B. CASE STUDY: CHARACTERIZING AN ENTIRE SYSTEM USING THE PROPOSED TAXONOMY

In this case study, we use our taxonomy to characterize the CryptoKitties application system,[10] a blockchain based computer game for breed-able virtual cats known as CryptoKitties (CK) tokens. The game was developed to support blockchain technology education through gamification, as its key mechanism is tied to crypto-assets and smart contracts. CryptoKitties is a featured decentralized gaming application and has been used in several studies [48], [77], [78]. Here

[9]https://in3.readthedocs.io/en/develop/intro.html

[10]https://www.cryptokitties.co/

TABLE 7. Characterizing 11 smart contracts in real-world systems using the proposed taxonomy.

Smart Contract	Hosting Network		Type†					Interaction		Events		License	
	Mainnet	Testnet	Legal contract	Application contract	Token contract	Organization contract	Thing contract	Machine-to-Machine	Human-to-Machine	Defined	Not Defined	Open-source	Closed source
Kitty Core	Y	N	N	N	Y	N	N	N	Y	Y	N	Y	N
Siring Clock Auction	Y	N	N	Y	N	N	N	N	Y	Y	N	Y	N
Sale Clock Auction	Y	N	N	Y	N	N	N	N	Y	Y	N	Y	N
Gene Science	Y	N	N	Y	N	N	N	N	Y	Y	N	Y	N
Offer	Y	N	N	Y	N	N	N	N	Y	Y	N	Y	N
Nexo Token	Y	N	N	N	Y	N	N	N	Y	Y	N	Y	N
Meme Token	Y	N	N	N	Y	N	N	N	N	Y	N	Y	N
BFH Daily Action V1	Y	N	N	Y	N	N	N	N	Y	Y	N	Y	N
LAND Proxy	Y	N	N	Y	N	N	N	Y	N	Y	N	Y	N
Broker	N	Y	N	N	N	N	Y	Y	N	Y	N	Y	N
NTUA Token	N	Y	N	N	Y	N	N	N	Y	Y	N	Y	N

† Compositional features. Y=Yes, N=No.

we characterize CryptoKitties in terms of its fundamental components, aspects and features.

1) EXECUTION ENVIRONMENT COMPONENTS

The Execution Environment of CryptoKitties consists of the Ethereum Mainnet (see Table 3), Ethereum distributed ledger (see Table 4), and Ethereum platform (see Table 5).

2) INTERNAL COMPONENTS

The application's internal components consist of PoW as a consensus protocol (see Table 6); five smart contracts: Kitty Core, Siring Clock Auction, Sale Clock Auction, Gene Science, and Offer: (see Table 7); and two tokens: ETH and CK (see Table 8).

3) EXTERNAL COMPONENTS

The application has no built-in wallet. Therefore, players must use a stand-alone wallet that supports cryptocurrencies and crypto-assets to use the DApp (see Table 9). Players do not need full nodes to play. However, if a player wants to participate in block validations and, hence, in the overall security of the chain, there is a need to deploy an Ethereum node. The CryptoKitties frontend is a web-based client-server application. It uses relational databases to replicate on-chain data related to the events emitted by the smart contracts. Players must purchase ETH to start buying, selling, and breeding kitties. It requires users to create accounts for the first time. They can then log in using their digital wallets.

CK is non-fungible and represents unique virtual kitties with different visual appearances at varying levels of rarity. These kitties can be sold, bought, gifted, and never die. Ethereum requires fees to be paid in ETH per transaction on its network. Thus, players must pay fees to place a bid or cancel an offer. This has implications for the game logic of CryptoKitties. On the one hand, an innovative auction mechanism is designed to minimize on-chain transactions and achieve a better user experience. It is a descending clock auction where sellers pay gas fees to initiate an auction, and buyers pay gas fees, in addition to the offer value, only when they complete a purchase of a CK.

However, CryptoKitties requires players to pay additional fees to cover the cost of transactions performed for them as part of the game logic. For example, a birthing fee is required each time the players breed their own kitties. The fee is paid by the player, collected by CryptoKitties smart contracts, and paid to the network miners when a new CK is generated and written to the blockchain. Because the application is based on a distributed ledger with a single-ledger-based architecture, there is a potential for an increasing number of game transactions to crowd out and be delayed by other businesses that use Ethereum.

When CryptoKitties was first released in 2017, the Gene Science contract was a closed-source, whereas the other four contracts were open-source. In 2019, the Gene Science contract was open sourced. The game logic is split between the application backend and frontend. Complicated game functions that require periodically calling of the smart contracts, such as generating new CK tokens (i.e., kitty birth), are handled by the blockchain, whereas other simpler functions, such as rendering images of kitties according to their identified genetic makeup in the smart contract, are handled by the frontend. The frontend handles mapping from CK genotypes

TABLE 8. Characterizing 11 tokens using the proposed taxonomy.

Token	Layer		Purpose			Standard		Underlying			Underlying Ownership			Fungibility		Transferability			Expiry		Flow		Offerings				Supply		Issuance	
																Transferable														
	Native	Secondary	Stability	Utility	Security	Standardized	No standard	No underlying	Cryptographic asset	Non-cryptographic asset	Whole	Fractional	Not applicable	Fungible	Non-fungible	Absolute	Constrained	Non-Transferable	Expirable	Non-expirable	Circular	Linear	ICO/ITO	IEO	STO	Revenue Model	Limited	Unlimited	Once	Conditional
Ether ETH	Y	N	N	Y	N	Y	N	Y	N	N	N	N	Y	Y	N	Y	N	N	N	Y	Y	N	Y	N	N	N	N	Y	N	Y
CryptoKitties CK	N	Y	N	Y	N	Y	N	Y	N	N	N	N	Y	N	Y	Y	N	N	N	Y	Y	N	N	N	N	Y	Y	N	N	Y
Steem	Y	N	N	Y	N	Y	N	Y	N	N	N	N	Y	Y	N	Y	N	N	N	Y	Y	N	Y	N	N	N	N	Y	N	Y
Steem Power SP	Y	N	N	Y	N	Y	N	N	Y	N	Y	N	N	Y	N	N	Y	N	N	Y	Y	N	N	N	N	Y	Y	N	N	Y
Steem Dollar SBD	Y	N	Y	N	N	Y	N	N	Y	N	Y	N	N	Y	N	Y	N	N	N	Y	Y	N	Y	N	N	N	N	Y	N	Y
Dtube DTC	N	Y	N	Y	N	N	Y	Y	N	N	N	N	Y	Y	N	Y	N	N	N	Y	Y	N	N	Y	N	N	N	Y	N	Y
AFIT	N	Y	N	Y	N	Y	N	Y	N	N	N	N	Y	Y	N	Y	N	N	N	Y	Y	N	N	Y	N	N	Y	N	N	Y
Geon coin GC	N	Y	Y	N	N	N	Y	Y	N	N	N	N	Y	Y	N	Y	N	N	N	Y	Y	N	N	N	N	Y	N	Y	N	Y
Geon token GT	N	Y	N	Y	N	Y	N	N	N	Y	N	N	Y	Y	N	Y	N	N	Y	N	N	Y	Y	N	N	N	Y	N	Y	N
SPiCE token	Y	N	N	N	Y	Y	N	N	N	Y	Y	N	N	Y	N	Y	N	N	N	Y	Y	N	N	N	Y	N	Y	N	Y	N
Blockimmo IMMO	N	Y	N	N	Y	Y	N	N	N	Y	N	Y	N	Y	N	N	Y	N	Y	N	Y	N	N	N	Y	N	Y	N	N	Y

Y=Yes, N=No.

(i.e., token Ids) stored on the blockchain to phenotypes stored in the relational database. Transaction data are sent from the frontend to Ethereum via the Web3 API, whereas on-chain and off-chain data are retrieved via a RESTful interface. To this end, and at a high-level of abstraction, we can conclude that the architecture of CryptoKitties consists of an on-chain backend, an off-chain backend, and a frontend. Linking the CryptoKitties backend to its frontend components offers several insights, as discussed in Section VI.

VI. DISCUSSION

Although we have demonstrated the applicability of the proposed taxonomy through diverse examples found in real-world systems, the taxonomy has not been validated by blockchain systems developers. While validating the taxonomy will be our ongoing research, this section discusses our general observations obtained from using the taxonomy to characterize the blockchain systems and offers our suggestions for some key decisions for the design of blockchain systems.

A. OBSERVATIONS

Blockchain platforms are largely *general-purpose, open-source platforms and are governed by communities of developers*. In agreement with [79], incentive mechanisms should consider rewarding full nodes, such as archival nodes, despite being miners. In addition, full nodes that solve computational problems and do not win block rewards consume sufficient storage and power with no reward. Blockchain systems are not fully decentralized. Although their execution environment is intended to be decentralized, there are other components that affect the decentralization of the final application, such as custody of the digital wallet and centralized frontends.

Moreover, three expertise areas are essential for developing blockchain applications. The first area of expertise is ***smart contract development***, which focuses on the programming and testing of smart contracts. This area requires a new thinking vector that differs from conventional software development [80], [81]. More importantly, smart contracts are deployed as transactions in the blockchain. Thus, deployed smart contracts are immutable, irreversible, and non-modifiable. It is crucial to test smart contracts, ideally on test networks, before actual production of blockchain systems. The second area of expertise is ***infrastructure setting***. This area focuses on configuring and establishing a blockchain system execution environment. It involves working with blockchain platforms and making decisions concerning architectural design such as scalability and data privacy [82], [83]. The last area of expertise is ***front-end development***. This area generally focuses on conventional software development, including the UI and user experience design. It also focuses on establishing communication services to the application's backend.

B. DESIGN DECISIONS

From a software developer's perspective, there are some design decisions that should be considered when developing blockchain application systems:

1) At least three licenses should be considered for a single blockchain application. The first is the license of the blockchain platform, the second is the license of smart

TABLE 9. Characterizing the network nodes found the publicly accessible environment sensor use case using the proposed taxonomy.

Node/Client	Category				Role				Operation†		Ledger Copy				Ledger Synchronization									HW Requirement				Chain Support		Deployment		Access		Geo Location						
															Requisite		Data				Sync Point																				
	Full node	Light node	Pruned node	Service node	Archive	Block validation	Data verification	Utility	Ledger-oriented	Network-oriented	Full	Partial	Headers only	None	Required	Not required	Block headers	Block data	Intermediate states	None	Genesis	Checkpoint	None	Servers	Computers	Mobile devices	IoT devices	Single chain	Multiple chains	On premise	Node-as-a-service	Centralized	Decentralized	Asia	Africa	North America	South America	Antarctica	Europe	Australia
IN3 node	Y	N	N	N	N	Y	N	Y	Y	Y	Y	N	N	N	Y	N	Y	Y	N	N	Y	N	N	Y	N	N	N	N	Y	N	Y	N	Y	N	N	N	N	Y	N
IN3 client	N	N	N	Y	N	N	N	Y	N	Y	N	N	N	Y	N	Y	N	N	N	Y	N	N	Y	N	N	N	Y	N	Y	Y	N	N	Y	N	N	N	N	Y	N
Watchdogs	N	N	N	Y	N	N	N	Y	N	Y	N	N	N	Y	N	Y	N	N	N	Y	N	N	Y	Y	N	N	N	N	Y	N	Y	N	Y	N	N	N	N	Y	N
PBCD	N	Y	N	N	N	N	Y	N	Y	N	N	N	Y	N	Y	N	Y	N	N	N	N	Y	N	N	Y	N	N	Y	N	Y	N	N	Y	N	N	N	N	Y	N

† Compositional features. Y=Yes, N=No.

TABLE 10. Characterizing 13 digital wallets using the proposed taxonomy.

Wallet	Tangibility		Type									Recovery Mechanism†					Custody		Internet Connection†		Signing		Supported Tokens†			In-Wallet Exchange		Deployment†			Integration Capability		
									Browser-based														Crypto Currency					Stand-alone*		Integrable†		
	Tangible	Intangible	HW	Paper	Smart Contract	Web	Mobile	Desktop	Built-in	Extension	Interface	Passphrase	Guardians	Seed phrase	Password-derived key	None	Custodial	Non-custodial	Hot	Cold	Single signature	Multi-Signature	Single currency	Multi currency	Crypto-Asset	Supported	Not supported	In-DApp	Local	Remote	HW wallet	SW wallet	Non-Integrable
Ledger Nano S	Y	N	Y	N	N	N	N	N	N	N	N	N	N	Y	N	N	N	Y	N	Y	Y	N	N	Y	Y	N	Y	N	Y	N	N	Y	N
Trezor One	Y	N	Y	N	N	N	N	N	N	N	N	Y	N	Y	N	N	N	Y	N	Y	Y	N	N	Y	N	N	Y	N	Y	N	N	N	Y
My Ether Wallet	N	Y	N	N	N	Y	N	N	N	N	N	N	N	N	N	Y	N	Y	Y	N	Y	N	N	Y	N	Y	N	N	N	Y	Y	Y	N
MetaMask	N	Y	N	N	N	N	N	N	N	Y	N	N	N	Y	N	N	N	Y	Y	N	Y	N	N	Y	Y	Y	N	N	Y	N	Y	Y	N
Electrum	N	Y	N	N	N	N	N	Y	N	N	N	N	N	Y	N	N	N	Y	Y	Y	N	Y	Y	N	N	N	Y	N	Y	N	Y	N	N
Coinbase.com	N	Y	N	N	N	Y	N	N	N	N	N	N	N	Y	N	N	Y	N	Y	N	Y	N	N	Y	N	Y	N	N	N	Y	N	N	Y
Authereum	N	Y	N	N	Y	N	N	N	N	N	N	N	N	N	Y	N	N	Y	Y	N	Y	N	N	Y	Y	Y	N	N	Y	Y	N	Y	N
Parity signer	N	Y	N	N	N	N	Y	N	N	N	N	N	N	Y	N	N	N	N	N	Y	Y	N	N	Y	Y	N	Y	N	Y	N	N	N	Y
Argent	N	Y	N	N	Y	N	N	N	N	N	N	N	Y	N	N	N	N	Y	Y	N	Y	N	N	Y	Y	Y	N	N	Y	Y	Y	Y	N
MEW App	N	Y	N	N	N	N	Y	N	N	N	N	N	N	N	N	Y	N	Y	Y	N	Y	N	N	Y	N	Y	N	N	Y	N	Y	Y	N
Coinbase Wallet	N	Y	N	N	N	N	Y	N	N	N	N	N	N	Y	N	N	N	Y	Y	N	Y	N	N	Y	Y	Y	N	N	Y	N	N	N	Y
Opera Touch	N	Y	N	N	N	N	N	N	Y	N	N	N	N	Y	N	N	N	Y	Y	N	Y	N	N	Y	Y	N	Y	N	Y	N	N	N	Y
Geon DApp	N	Y	N	N	N	N	N	N	N	N	Y	N	N	N	N	Y	N	Y	Y	N	Y	N	N	N	Y	N	Y	Y	N	N	N	N	Y

† Compositional features. Y=Yes, N=No.

contracts, and the last is the license of the frontend application.

2) For a single blockchain application, four deployments should be considered. The first is the deployment of the blockchain platform whether on a local node or as a service; the second is the deployment of the smart contracts, whether on a production or test network and whether on a single chain or multiple chains; the third is the deployment of the frontend application whether local on users' devices or remote on web servers; and the last is the deployment of the digital wallet whether embedded within the DApp, local on users' devices, or remote on web servers.
3) For a single blockchain application, there are at least four access-control levels. In addition to the network, ledger, and smart contracts permissions, the frontend can be open access or restricted, for example, to a specific geolocation.
4) Membership supported by the backend network is different from the membership supported by the frontend application. For example, a DApp may constrain the members' registration to a certain group of people although the underlying blockchain network is open and its membership is not controlled.

VII. CONCLUSION

In this paper we have made three major contributions to the research and development of blockchain systems:

First, we propose a novel taxonomy for blockchain systems. The taxonomy is comprehensive, characterizing a blockchain system using 3 subsystems, 8 fundamental components, 83 aspects, and 198 features. Such a detailed characterization can better inform software developers in their design and implementation of blockchain systems. Based on our literature review, we believe this is the first comprehensive taxonomy for blockchain systems.

Second, we show that the taxonomy is flexible and adaptable to the characterization of individual software components from a wide range of real-world blockchain systems. Using this taxonomy, we can gain a systematic understanding of a blockchain system or their components.

Finally, we offer our observations and insights to blockchain systems developers. We intend to provide this taxonomy as a service, making it accessible by the blockchain researchers and practitioners, so that it can be validated and used in practice. This may also open up collaboration opportunities to extend and standardize the taxonomy. Based on this taxonomy, our future work will research and develop reference software architectures for blockchain systems.

REFERENCES

[1] J. Yli-Huumo, D. Ko, S. Choi, S. Park, and K. Smolander, "Where is current research on blockchain technology?—A systematic review," *PLoS ONE*, vol. 11, no. 10, Oct. 2016, Art. no. e0163477.

[2] U. Bodkhe, S. Tanwar, K. Parekh, P. Khanpara, S. Tyagi, N. Kumar, and M. Alazab, "Blockchain for Industry 4.0: A comprehensive review," *IEEE Access*, vol. 8, pp. 79764–79800, 2020.

[3] F. Casino, T. Dasaklis, and C. Patsakis, "A systematic literature review of blockchain-based applications: Current status, classification and open issues," *Telematics Inform.*, vol. 36, pp. 55–81, Mar. 2018.
[4] A. Alammary, S. Alhazmi, M. Almasri, and S. Gillani, "Blockchain-based applications in education: A systematic review," *Appl. Sci.*, vol. 9, no. 12, p. 2400, Jun. 2019.
[5] G. Chen, B. Xu, M. Lu, and N.-S. Chen, "Exploring blockchain technology and its potential applications for education," *Smart Learn. Environ.*, vol. 5, no. 1, p. 1, Dec. 2018.
[6] A. Tandon, A. Dhir, A. K. M. N. Islam, and M. Mäntymäki, "Blockchain in healthcare: A systematic literature review, synthesizing framework and future research agenda," *Comput. Ind.*, vol. 122, Nov. 2020, Art. no. 103290.
[7] P. K. Sharma and J. H. Park, "Blockchain based hybrid network architecture for the smart city," *Future Gener. Comput. Syst.* vol. 86, pp. 650–655, Sep. 2018.
[8] A. Khatoon, P. Verma, J. Southernwood, B. Massey, and P. Corcoran, "Blockchain in energy efficiency: Potential applications and benefits," *Energies*, vol. 12, no. 17, p. 3317, Aug. 2019.
[9] F. Antonucci, S. Figorilli, C. Costa, F. Pallottino, L. Raso, and P. Menesatti, "A review on blockchain applications in the agri-food sector," *J. Sci. Food Agricult.*, vol. 99, no. 14, pp. 6129–6138, Nov. 2019.
[10] M. Pournader, Y. Shi, S. Seuring, and S. C. L. Koh, "Blockchain applications in supply chains, transport and logistics: A systematic review of the literature," *Int. J. Prod. Res.*, vol. 58, no. 7, pp. 2063–2081, Apr. 2020.
[11] S. Huckle, R. Bhattacharya, M. White, and N. Beloff, "Internet of Things, blockchain and shared economy applications," *Proc. Comput. Sci.*, vol. 98, pp. 461–466, Sep. 2016.
[12] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. IEEE Int. Congr. Big Data (BigData Congr.)*, Jun. 2017, pp. 557–564, doi: 10.1109/BigDataCongress.2017.85.
[13] P. Tasca and C. J. Tessone, "A taxonomy of blockchain technologies: Principles of identification and classification," *Ledger*, vol. 4, 2019, doi: 10.5195/ledger.2019.140.
[14] L. Pawczuk, J. Holdowsky, R. Massey, and B. Hansen, "Deloitte's 2020 global blockchain survey," Deloitte Center Integr. Res., New York, NY, USA, Tech. Rep., 2020.
[15] S. Wieninger, G. Schuh, and V. Fischer, "Development of a blockchain taxonomy," in *Proc. IEEE Int. Conf. Eng., Technol. Innov.*, Jun. 2019, pp. 1–9.
[16] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba, "A taxonomy of blockchain-based systems for architecture design," in *Proc. IEEE Int. Conf. Softw. Archit.*, Apr. 2017, pp. 243–252.
[17] T. Schulze, S. Seebacher, and F. Hunke, "Conceptualizing the role of blockchain technology in digital platform business," in *Exploring Service Science*. Cham, Switzerland: Springer, 2020, pp. 150–163.
[18] E. Bellini, Y. Iraqi, and E. Damiani, "Blockchain-based distributed trust and reputation management systems: A survey," *IEEE Access*, vol. 8, pp. 21127–21151, 2020.
[19] S. M. Sarkintudu, H. H. Ibrahim, and A. B. Abdwahab, "Taxonomy development of blockchain platforms: Information systems perspectives," in *Proc. AIP Conf.*, 2016, Art. no. 020130.
[20] S. Tönnissen and F. Teuteberg, "Towards a taxonomy for smart contracts," in *Proc. 26th Eur. Conf. Inf. Syst. (ECIS)*, Portsmouth, U.K., 2018. [Online]. Available: https://aisel.aisnet.org/ecis2018_rp/12
[21] T. Ankenbrand, D. Bieri, R. Cortivo, J. Hoehener, and T. Hardjono, "Proposal for a comprehensive (crypto) asset taxonomy," in *Proc. Crypto Valley Conf. Blockchain Technol.*, 2020, pp. 16–26.
[22] H. Arslanian and F. Fischer, "A high-level taxonomy of crypto-assets," in *The Future of Finance: The Impact of Fintech, AI, and Crypto on Financial Services*. Cham, Switzerland: Springer, 2019, pp. 139–156.
[23] L. Oliveira, L. Zavolokina, I. Bauer, and G. Schwabe, "To token or not to token: Tools for understanding blockchain tokens," in *Proc. Int. Conf. Inf. Syst. (ICIS)*, San Francisco, CA, USA, 2018, doi: 10.5167/uzh-157908.
[24] R. C. Nickerson, U. Varshney, and J. Muntermann, "A method for taxonomy development and its application in information systems," *Eur. J. Inf. Syst.*, vol. 22, no. 3, pp. 336–359, 2013.
[25] D. Efanov and P. Roschin, "The all-pervasiveness of the blockchain technology," *Proc. Comput. Sci.*, vol. 123, pp. 116–121, Jan. 2018.
[26] Y. Xinyi, Z. Yi, and Y. He, "Technical characteristics and model of blockchain," in *Proc. Int. Conf. Commun. Softw. Netw.*, 2018, pp. 562–566.
[27] W. F. Silvano and R. Marcelino, "IOTA tangle: A cryptocurrency to communicate Internet-of-Things data," *Future Gener. Comput. Syst.*, vol. 112, pp. 307–319, Nov. 2020.
[28] X. Xu, I. Weber, and M. Staples, *Architecture for Blockchain Applications*, 1st ed. Cham, Switzerland: Springer, 2019.
[29] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran, and S. Chen, "The blockchain as a software connector," in *Proc. 13th Working IEEE/IFIP Conf. Softw. Archit.*, Jun. 2016, pp. 182–191.
[30] J. S. Arun, J. Cuomo, and N. Gaur, *Blockchain for Business*. Reading, MA, USA: Addison-Wesley, 2019.
[31] J. Mendling, I. Weber, W. V. D. Aalst, J. V. Brocke, C. Cabanillas, F. Daniel, S. Debois, C. D. Ciccio, M. Dumas, and S. Dustdar, "Blockchains for business process management-challenges and opportunities," *ACM Trans. Manage. Inf. Syst.*, vol. 9, no. 1, p. 4, 2018.
[32] W. Viriyasitavat and D. Hoonsopon, "Blockchain characteristics and consensus in modern business processes," *J. Ind. Inf. Integr.*, vol. 13, pp. 32–39, Mar. 2019.
[33] B. Singhal, G. Dhameja, and P. S. Panda, "How blockchain works," in *Beginning Blockchain: A Beginner's Guide to Building Blockchain Solutions*. Berkeley, CA, USA: Apress, 2018, pp. 31–148.
[34] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran, "An overview on smart contracts: Challenges, advances and platforms," *Future Gener. Comput. Syst.*, vol. 105, pp. 475–491, Dec. 2020.
[35] V. Morabito, *Business Innovation Through Blockchain*. Cham, Switzerland: Springer, 2017.
[36] X. Xu, C. Pautasso, L. Zhu, Q. Lu, and I. Weber, "A pattern collection for blockchain-based applications," in *Proc. 23rd Eur. Conf. Pattern Lang. Programs*, 2018, p. 3.
[37] X. Xu, F. Rahman, B. Shakya, A. Vassilev, D. Forte, and M. Tehranipoor, "Electronics supply chain integrity enabled by blockchain," *ACM Trans. Design Autom. Electron. Syst.*, vol. 24, no. 3, pp. 1–25, Jun. 2019.
[38] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 7, pp. 1366–1385, Jul. 2018.
[39] H. Jin, X. Dai, and J. Xiao, "Towards a novel architecture for enabling interoperability amongst multiple blockchains," in *Proc. Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 1203–1211.
[40] L. Deng, H. Chen, J. Zeng, and L.-J. Zhang, "Research on cross-chain technology based on sidechain and hash-locking," in *Proc. Edge Comput. (EDGE)*, 2018, pp. 144–151.
[41] V. Buterin, "Chain interoperability," R3 Res. Paper, 2016, vol. 9. [Online]. Available: https://theblockchaintest.com/uploads/resources/R3%20-%20Chain%20Interoperability%20-%202016%20-%20Sep.pdf
[42] T. Koens and E. Poll, "Assessing interoperability solutions for distributed ledgers," *Pervas. Mobile Comput.*, vol. 59, Oct. 2019, Art. no. 101079.
[43] E. J. Scheid, T. Hegnauer, B. Rodrigues, and B. Stiller, "Bifröst: A modular blockchain interoperability API," in *Proc. IEEE 44th Conf. Local Comput. Netw. (LCN)*, Oct. 2019, pp. 332–339, doi: 10.1109/LCN44214.2019.8990860.
[44] W. Zhang and X. Lu, "System and method for universal blockchain interoperability," Google Patents 10 901 983, Jan. 26, 2020.
[45] A. Palai, M. Vora, and A. Shah, "Empowering light nodes in blockchains with block summarization," in *Proc. 9th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Feb. 2018, pp. 1–5, doi: 10.1109/NTMS.2018.8328735.
[46] T. Ali Syed, A. Alzahrani, S. Jan, M. S. Siddiqui, A. Nadeem, and T. Alghamdi, "A comparative analysis of blockchain architecture and its applications: Problems and recommendations," *IEEE Access*, vol. 7, pp. 176838–176869, 2019.
[47] L. Ismail and H. Materwala, "A review of blockchain architecture and consensus protocols: Use cases, challenges, and solutions," *Symmetry*, vol. 11, no. 10, p. 1198, Sep. 2019.
[48] H. Pervez, M. Muneeb, M. U. Irfan, and I. U. Haq, "A comparative analysis of DAG-based blockchain architectures," in *Proc. 12th Int. Conf. Open Source Syst. Technol. (ICOSST)*, Dec. 2018, pp. 27–34, doi: 10.1109/ICOSST.2018.8632193.
[49] H.-Y. Paik, X. Xu, D. Bandara, S. Lee, and S. K. Lo, "Analysis of data management in blockchain-based systems: From architecture to governance," *IEEE Access*, vol. 7, pp. 186091–186107, 2019.
[50] Z. Moezkarimi, R. Nourmohammadi, S. Zamani, F. Abdollahei, Z. Golmirzaei, and A. Arabsorkhi, "An overview on technical characteristics of blockchain platforms," in *Proc. High-Perform. Comput. Big Data Anal.*, 2018, pp. 265–278.

[51] H. S. Jennath and S. Asharaf, "Survey on blockchain consensus strategies," in *Proc. ICDSMLA*, 2019, pp. 637–654.
[52] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *Proc. Int. Workshop Open Problems Netw. Secur.*, 2015, pp. 112–125.
[53] (2018). *Smart Contracts Alliance, Smart Contracts: Is the Law Ready? The Chamber of Digital Commerce*. [Online]. Available: https://digitalchamber.s3.amazonaws.com/Smart-Contracts-Whitepaper-WEB.pdf
[54] S. Wang, W. Ding, J. Li, Y. Yuan, L. Ouyang, and F.-Y. Wang, "Decentralized autonomous organizations: Concept, model, and applications," *IEEE Trans. Computat. Social Syst.*, vol. 6, no. 5, pp. 870–878, Oct. 2019.
[55] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with IoT. Challenges and opportunities," *Future Gener. Comput. Syst.*, vol. 88, pp. 173–190, Nov. 2018.
[56] Y. Hanada, L. Hsiao, and P. Levis, "Smart contracts for machine-to-machine communication: Possibilities and limitations," in *Proc. IEEE Int. Conf. Internet Things Intell. Syst. (IOTAIS)*, Nov. 2018, pp. 130–136, doi: 10.1109/IOTAIS.2018.8600854.
[57] M. Shirole, M. Darisi, and S. Bhirud, "Cryptocurrency token: An overview," in *Proc. IC-BCT*, 2019, pp. 133–140.
[58] *Simple Assets Standard*, Standard, CryptoLions, Ukrain, 2021.
[59] S. Voshmgir, *Token Economy: How the Web3 Reinvents the Internet*, 2nd ed. Berlin, Germany: BlockchainHub, 2020.
[60] F. Hartmann, G. Grottolo, X. Wang, and M. I. Lunesu, "Alternative fundraising: Success factors for blockchain-based vs. conventional crowdfunding," in *Proc. IEEE Int. Workshop Blockchain Oriented Softw. Eng. (IWBOSE)*, Feb. 2019, pp. 38–43, doi: 10.1109/IWBOSE.2019.8666515.
[61] A. Panin, K.-K. Kemell, and V. Hara, "Initial coin offering (ICO) as a fundraising strategy: A multiple case study on success factors," in *Proc. Int. Conf. Softw. Bus.*, 2019, pp. 237–251.
[62] J. Evans. (2020). *Blockchain Nodes: How They Work (All Types Explained)*. [Online]. Available: https://nodes.com/
[63] R. Matzutt, B. Kalde, J. Pennekamp, A. Drichel, M. Henze, and K. Wehrle, "How to securely prune bitcoin's blockchain," 2020, *arXiv:2004.06911*.
[64] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Vienna, Austria, Oct. 2016, pp. 3–16.
[65] Q. Hu, M. Xu, S. Wang, and S. Guo, "Sync or fork: Node-level synchronization analysis of blockchain," in *Proc. Int. Conf. Wireless Algorithms, Syst., Appl.*, 2020, pp. 170–181.
[66] B. Bollen and J. Goldberg, "Recovering access to a digital smart contract wallet," Google Patents 16 804 782, Sep. 3, 2020.
[67] K. Peffers, M. Rothenberger, T. Tuunanen, and R. Vaezi, "Design science research evaluation," in *Proc. Int. Conf. Design Sci. Res. Inf. Syst.*, 2012, pp. 398–410.
[68] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research perspectives and challenges for bitcoin and cryptocurrencies," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 104–121, doi: 10.1109/SP.2015.14.
[69] S. King, "Primecoin: Cryptocurrency with prime number proof-of-work," *July 7th*, vol. 1, no. 6, Jul. 2013.
[70] D. Larimer, "Delegated proof-of-stake (DPoS)," *Bitshare Whitepaper*, vol. 81, p. 85, Jun. 2014.
[71] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. OsDI*, 1999, pp. 173–186.
[72] J. Kwon, *Tendermint: Consensus Without Mining*, Draft V. 0.6, Fall 1.11, 2014. [Online]. Available: https://www.weusecoins.com/assets/pdf/library/Tendermint%20Consensus%20without%20Mining.pdf
[73] D. Schwartz, N. Youngs, and A. Britto. (2014). *The Ripple Protocol Consensus Algorithm*. [Online]. Available: https://ripple.com/files/ripple_consensus_whitepaper.pdf
[74] D. Mazieres, "The stellar consensus protocol: A federated model for internet-level consensus," *Stellar Develop. Found.*, vol. 32, pp. 1–45, Jul. 2015.
[75] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract] y," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 3, pp. 34–37, 2014.
[76] G. Papadodimas, G. Palaiokrasas, A. Litke, and T. Varvarigou, "Implementation of smart contracts for blockchain based IoT applications," in *Proc. 9th Int. Conf. Netw. Future (NOF)*, Nov. 2018, pp. 60–67, doi: 10.1109/NOF.2018.8597718.
[77] S. Eskandari, S. Moosavi, and J. Clark, "SoK: Transparent dishonesty: Front-running attacks on blockchain," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2019, pp. 170–189.
[78] Y. Zhou, D. Kumar, S. Bakshi, J. Mason, A. Miller, and M. Bailey, "Erays: Reverse engineering Ethereum's opaque smart contracts," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)*, 2018, pp. 1371–1385.
[79] Y. Liu, Z. Fang, M. H. Cheung, W. Cai, and J. Huang, "Economics of blockchain storage," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–6, doi: 10.1109/ICC40277.2020.9148934.
[80] G. Zheng, L. Gao, L. Huang, and J. Guan, "Develop secure contract," in *Ethereum Smart Contract Development in Solidity*. Singapore: Springer, 2021, pp. 215–249.
[81] M. Bartoletti and L. Pompianu, "An empirical analysis of smart contracts: Platforms, applications, and design patterns," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, Cham, Switzerland, 2017, pp. 494–509.
[82] X. Xu, C. Pautasso, L. Zhu, Q. Lu, and I. Weber, "A pattern collection for blockchain-based applications," in *Proc. 23rd Eur. Conf. Pattern Lang. Programs*, Jul. 2018, p. 3.
[83] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba, "A taxonomy of blockchain-based systems for architecture design," in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, Apr. 2017, pp. 243–252.

FOUZIA E. ALZHRANI received the M.S. degree in advanced computer science from Swansea University, Swansea, U.K., in 2016. She is currently pursuing the Ph.D. degree in software engineering with the School of Engineering, The University of Manchester, Manchester, U.K.

Since 2012, she has been working at King Abdulaziz University in different roles. First, she was the Software Management Supervisor at the Deanship of Information Technology. Then, she became a Teaching Assistant at the Faculty of Computing and Information Technology (FCIT). She is currently a Lecturer with the Information Systems Department, FCIT. She is also a Certified Blockchain Solution Architect. Her research interests include software engineering with intersection to software design patterns, blockchain, and business process management.

Ms. Alzhrani is a member of the IEEE Task Force on Process Mining. Her awards and honors include the Best Performance by a Student in Advanced Computer Science (Swansea University).

KAWTHER A. SAEEDI (Member, IEEE) received the Ph.D. degree from the School of Computing, The University of Manchester, U.K. She is currently an Assistant Professor with the Information System Department, King Abdulaziz University. She is interested in applied research that involves new technology trends. Her core focuses include software engineering, blockchain, and artificial intelligent. With regards to her professional experience, she has worked as a Technology Specialist at ING Financial Market, Amsterdam. She also worked as a Sales Executive Engineer for IT integrated solutions at Aljafali, Jeddah. She worked and managed projects of different scales and technologies. She is a Certified Professional Scrum Master, a Senior Software Engineer, and a System Architect. She is an Active Team Player and have led the IS Department for four years.

LIPING ZHAO is currently an Associate Professor with the Department of Computer Science, The University of Manchester. Her research interests include the areas of software engineering (SE) and requirements engineering (RE), with intersection to conceptual modeling, software design patterns, service-oriented computing, natural language processing (NLP), machine learning (ML), and deep learning (DL).

Chapter 4

An Architectural Pattern Language for Blockchain Application Development

Research Objective

Objective 3: To identify architectural patterns for blockchain application development.

Thesis Context

This chapter uses the results from Chapter 3 to analyze the Industry-developed applications dataset. The analysis is supported by the findings from analyzing the Academia-researched applications dataset. Both datasets are described in Chapter 2. This chapter, i.e., Chapter 4 is based on a paper that has been submitted to Information and Software Technology. The full reference of this paper is as follows:

F. Alzhrani, K. Saeedi, and L. Zhao, "An Architectural Pattern Language for Blockchain Application Development," *Information and Software Technology*, Under Review.

CRediT authorship contribution statement

Fouzia Alzhrani: Conceptualization, Data Curation, Methodology, Writing - Original Draft, Writing - Review & Editing. **Kawther Saeedi**: Conceptualization, Supervision, Writing - Review & Editing. **Liping Zhao**: Conceptualization, Supervision, Writing - Review & Editing.

An Architectural Pattern Language for Blockchain Application Development

Fouzia Alzhrani[a,b], Kawther Saeedi[a] and Liping Zhao[b,*]

[a]King Abdulaziz University, Jeddah, Saudi Arabia
[b]The University of Manchester, Manchester, United Kingdom

ARTICLE INFO

Keywords:
Blockchain Technology
Blockchain Application Development
Software Architecture
Blockchain Application Architecture
Blockchain Architectural Patterns

ABSTRACT

Context: Blockchain technology has recently gained popularity in diverse applications far beyond finance transactions, but the development of blockchain application systems is hard, facing the stringent demand on quality requirements, such as performance, scalability, reliability, portability, interoperability, privacy, and security. Software architecture has long been recognized to be highly critical to realizing key quality requirements. However, as blockchain application development is still a new field, little work has been carried out on software architectures for blockchain applications.
Objective: This paper aims to make a contribution in blockchain application software architectures by proposing a set of interconnected architectural patterns.
Method: These patterns, which are collectively called a pattern language, are identified from 400 real-world blockchain applications. These patterns cover three aspects or viewpoints of blockchain application architectures: Structure, Interaction and Transaction. This paper describes these patterns and demonstrates their applicability by reverse-engineering three real-world blockchain applications.
Results: This paper identifies 12 interconnected architectural patterns for blockchain applications. These patterns can aid in the design of blockchain application architectures from three different perspectives, comprising structural, international and transactional. The patterns can also be used to review and detect quality issues when designing blockchain applications.
Conclusions: This paper makes a novel contribution to blockchain application architecture research and shows the importance of architectural patterns for supporting application-specific quality requirements.

1. Introduction

A blockchain is a secured transaction management system in which business transactions and digital assets are stored as chains of blocks [1]. Originally developed to record cryptocurrency transactions [2], blockchains are considered highly secure due to their anonymity, auditability, persistency, and decentralization. Blockchain technology has been steadily gaining popularity, with applications in diverse domains that extend far beyond finance and digital payments [3, 4, 5, 6]. These domains include voting, healthcare, supply chain management, and the Internet of Things (IoT) [7].

However, developing blockchain applications faces a large number of technical challenges [7] and the stringent demand on non-functional requirements (NFRs), particularly quality requirements, such as performance, scalability, reliability, portability, interoperability, privacy, and security [8].

Software architecture has long been recognized as a critical construct in software development. A good architecture can aid in the design of a software system that will satisfy key quality requirements, but a bad one can be disastrous [9]. As blockchain application systems are still new, work on software architectures for these systems is rather limited. To date, most work has primarily been focused on computing networks of blockchain hardware systems using the *Peer-to-Peer (P2P)* architecture [10, 8]. In this paper, we aim to make a contribution in this area by proposing a set of interconnected software architectural patterns as design constructs for blockchain application development. We call this set of architectural patterns *a pattern language*, a terminology commonly used in the software patterns community [11, 12].

In particular, our pattern language consists of 12 architectural patterns, extracted from 400 real-world blockchain applications hosted on different online blockchain platforms [13]. These patterns cover three aspects or viewpoints of blockchain application architectures: *structural*, *interactional* and *transactional*. They aim to help blockchain application architects and developers to make design decisions about how to structure key application components and their interactions. Such decisions can then impact on the fulfillment of application-level quality requirements.

The rest of the paper reports our pattern language, which is organized into the following sections: Section 2 reviews related work on blockchain architectures, whereas Section 3 gives an overview and background of blockchain applications. Section 4 describes a set of core blockchain applications components which serve as the foundations for the patterns in our pattern language. Section 5 presents our pattern language. Section 6 evaluates this pattern language by using it to describe the architectures of three real-world blockchain application systems. Section 7 discusses our research limitation. Finally, Section 8 concludes the paper.

*Corresponding author
feaalzhrani@kau.edu.sa (F. Alzhrani); ksaeedi@kau.edu.sa (K. Saeedi); liping.zhao@manchester.ac.uk (L. Zhao)
ORCID(s): 0000-0002-5780-0469 (F. Alzhrani)

2. Related Work

Literature shows that some efforts have been made to document reusable design patterns for blockchain applications. Some of the proposed patterns focus on designing generic applications [14] [14, 15, 16, 17, 18, 19], whereas others are collected for specific contexts [20, 21, 22, 23]. The selection of appropriate design patterns is supported by seven decision models proposed by Xu et al. [24]. The models support design decisions related to data management, performance, security, oracle, and smart contract. However, there is a significant lack of research on the architectural patterns for blockchain applications.

Liu et al. [25] proposed a reference architecture for governance of blockchain systems, focusing on how governance fits in the development and use of blockchain. The proposed reference architecture consists of four layers: infrastructure, platform, API, and user layers. Within each layer, the study suggested a set of design patterns to address governance-related issues in blockchain systems. The architecture was evaluated using two real-world blockchain platforms.

In the context of Multi-Tenant Blockchain-Based Systems, Weber et al. [26] proposed a platform architecture to ensure data integrity of permissioned blockchains while maintaining privacy and performance isolation. This was achieved via anchoring the consensus state of each permissioned blockchain periodically to a public blockchain. The proposed architecture was implemented in a proof-of-concept prototype based on Ethereum.

A study by Wöhrer et al. [27] discussed design guidance for blockchain integration. It focused on operational and integration aspects of public permissionless blockchain-based applications. The study elaborated a number of design options related to decentralization, identity provisioning, transaction handling, key management, transaction state synchronization, blockchain connection, frontend provisioning, application logic, off-chain interaction, rich querying, and confidential storage.

On the other hand, few attempts have explicitly focused on analyzing Industry-developed blockchain applications. The survey in [28] investigated a specific type of blockchain application: digital games. This survey aimed to statistically analyze blockchain game trends. It identified four fundamental benefits of blockchains in the digital game industry: rule transparency, asset ownership, asset reusability, and user-generated content. In addition, the survey provided a high-level block-diagram architecture for blockchain-based games. This architecture provides a high-level view of the interactions between game players and blockchain networks through smart contracts. Although the dataset in [28] consisted of 23 game applications from the Bitcoin, Ethereum, EOS, TRON, and Nebulas platforms, the study mainly analyzed Ethereum and EOS games.

Another descriptive analysis was presented in [29]. It mainly focuses on analyzing transaction data from Ethereum applications. Specifically, the study investigated the popularity of blockchain applications, the openness of their source codes, and the costs of deploying and executing smart contracts. The findings in this study offer an overview of the Ethereum applications market and provide implications for its end-users and developers. The architecture presented in this study describes high-level interactions between clients and the smart contracts of Ethereum applications. Furthermore, it identifies three interaction patterns: direct, indirect, and mixed. The architecture comprises five layers: the DApp client, smart contract, DApp service, DApp server, and Ethereum blockchain. The study's dataset consists of 995 blockchain applications on the Ethereum platform.

At the repository level, the study in [30] presented a quantitative analysis of five public repositories of blockchain applications. The study examined consistency in terms of schema and content across these repositories. In this regard, the study defined a blockchain application as a decentralized application consisting of a front-end and smart contract. A basic three-layer architecture (application, contract, and service layers) was proposed for blockchain applications. It is a refinement of the two-layer architecture presented in [31]. Although the dataset used in [30] included 4760 blockchain applications, only the top 10-ranked applications on Ethereum were selected for smart contract analysis.

The reviewed literature either proposed design patterns or presented comprehensive statistical analyses of Ethereum applications. In contrast, this study addresses the architectural concerns of blockchain applications via an in-depth analysis of existing blockchain applications built on top of nine different blockchain platforms. This paper presents an architectural pattern language for blockchain applications. Although the dataset in this study contains a smaller number of blockchain applications than those in [29] and [30], it offers a broader range of blockchain platforms representing all the typical blockchain types: public, private, and consortium.

3. Types of blockchain applications

Our analysis of 63 research papers [13] on blockchain business applications has identified the following three broad application types. This categorization of blockchain business applications allows understanding application-specific architectures.

1. *Consumer-Centric applications*. They are use cases where independent entities would like to collaborate without relying confidentially and electronically on trusted intermediaries. Such use cases involve individuals' interactions and thus require high transparency. Various blockchain use cases from different business domains have been designed as Consumer-Centric applications [7, 32, 33, 34, 35]. For example, cryptocurrency exchange allows individuals to carry out currency transactions in a secure, trustworthy, and transparent environment [36, 37]. Furthermore, tokenization enables the exchange of values and data in the same manner as cryptocurrency exchanges. The idea of tokenization provides endless opportunities for innovative applications.

2. *Enterprise-Centric applications*. These applications offer the ability of trusted peer-to-peer exchange and automated execution of business contracts. This ability eliminates the need for third parties and ensures the integrity of the processes and data with security and cryptography. The elimination of intermediaries drives a new ecosystem of players and fosters the creation of new profit pools, microeconomics, competitors, consumers, and distributed ecosystems [38]. Enterprise-Centric blockchain applications are use cases designed for inter- and intra-organizational procedures such as auditing, monitoring, enterprise authentication and authorization, enterprise integration, and regulatory requirements. For example, demand forecasting in supply chain management is a collaborative and coordinated approach to forecasting future demand by supply chain members. A crucial prerequisite for demand forecasting is data exchange between the supply chain members. Blockchain offers a permissioned and trusted exchange system between supply chain members to ensure data integrity and privacy [39].
3. *IoT-Based blockchain applications*. The IoT ecosystems enable a diverse range of smart physical geolocated devices to communicate over the Internet to accomplish specific tasks. These digital transactions between smart devices demand high security for sensitive data and interactions, which is challenging to meet in traditional centralized development. Blockchain technology can effectively secure these transactions, ensure their safety and efficiency, save costs, automate IoT complex workflows, and promote resource sharing [7, 34, 40]. IoT-Based blockchain applications are use cases that incorporate machine-to-machine interaction. For example, a smart grid combines electric power transmission and information technology to form a power supply network consisting of distributed users (e.g., consumers and producers) and smart devices (e.g., smart meters, energy resources, and smart appliances). Therefore, smart grid solutions must support security, privacy, and transparency [7, 40, 41]. Blockchain has the potential to transform smart grids into decentralized and transparent processes. It offers opportunities for energy trading between participants, dynamic energy pricing based on availability and consumption, demand response for balancing energy demand and supply [34, 35, 42].

4. Blockchain application components as basis for our architectural patterns

To identify the architectural patterns, we manually selected 400 blockchain applications between September 2019 and February 2020 from these nine different blockchain platforms: Ethereum, Steem, EOS, Blockstack, Klaytn, POA, Hive, Corda, and Hyperledger Fabric. These applications and their sources are listed in Zenodo [43]. We mapped the applications to the three application categories. Table 1 shows their distributions on the nine different platforms. The selection of these applications is based on the following inclusion and exclusion criteria:

- Include applications with smart contract functionality, i.e. Blockchain 2.0 and Blockchain 3.0
- Include applications built on open source blockchain platform
- Include applications built on cross-industry blockchain platform
- Exclude Blockchain 1.0 applications
- Exclude applications with no supporting documents
- Exclude other applications that are related to protocols, application programming interfaces, and software development kits

From these 400 applications, we first identified a set of nine core components and organized them into a taxonomy [44]; we then mapped these components and their relationships onto the known software architecture patterns [45], [46] to identify a set of architectural patterns for blockchain applications. The mapping of these components to the identified architectural patterns in this pattern language. In what follows, we briefly describe these nine core components:

1. *Smart Contracts.* These are programmable commands that can conditionally transfer digital assets between parties predictably and transparently [47]. A smart contract is written in a blockchain platform-specific language. For example, Solidity is a standard programming language for smart contracts based on EVM, such as Ethereum and POA. Go, Java, and JavaScript are programming language choices for smart contracts in Hyperledger Fabric. Some platforms offer the ability only to command pre-built smart contract modules. In this case, application-specific smart contracts are tied to predefined functions in the platform's modules, such as Hyperledger Iroha. However, some blockchain platforms, such as EOS, provide pre-built smart contracts and allow user-defined contracts. In this case, the pre-built modules are used for core operations. Accordingly, some blockchain platforms execute smart contracts on specialized machines, whereas others run them natively within the platform architecture.
2. *Tokens.* Digital assets are known as tokens in the blockchain ecosystem. They are cryptographic assets that may represent cryptocurrencies or real-world objects [1, 48]. Tokens are primarily native – protocol - or application tokens. Native tokens are tied to a blockchain platform, whereas application tokens are linked to a blockchain application. For example, an Ethereum-based application must use ETH for transaction fees. However, it can also implement its

Table 1
An overview of the analyzed blockchain applications in this study

Application Category	Blockchain Platform	Network Type	Number of Applications	Source
Consumer-Centric Applications	Blockstack	Public	1	State of the DApps[1]
	EOS	Public	29	State of the DApps
	Ethereum	Public	171	State of the DApps
	Hive	Public	5	State of the DApps
	Klaytn	Public	14	State of the DApps
	POA	Public	2	State of the DApps
	Steem	Public	18	State of the DApps
Enterprise-Centric Applications	Corda	Consortium	67	R3 Marketplace[2]
	Ethereum	Public	56	State of the DApps
	Hyperledger Fabric	Private	12	IBM Code Patterns[3]
	Klaytn	Public	3	State of the DApps
	POA	Public	3	State of the DApps
	Steem	Public	2	State of the DApps
IoT-Based Applications	Corda	Consortium	8	R3 Marketplace
	Ethereum	Public	3	State of the DApps
	Hyperledger Fabric	Private	5	IBM Code Patterns
	Klaytn	Public	1	State of the DApps

[1] https://www.stateofthedapps.com, last accessed: July 2022.
[2] https://marketplace.r3.com, last accessed: November 2020.
[3] https://marketplace.r3.com, last accessed: November 2020.

application-specific tokens through smart contracts. Application tokens are application-specific tokens that are implemented as smart contracts. These are second-layer tokens defined and programmed by smart contract developers. There are several standards for application tokens. For instance, ERC20 and ERC721 are standards for fungible and non-fungible EVM-based application tokens, respectively [49].

3. *Digital Wallets.* A digital wallet stores and manages digital keys and crypto assets in a blockchain system. Digital wallets can involve cold or air-gapped storage, which is used offline, or hot software applications, which may offer in-wallet exchanges of cryptocurrencies. Digital wallets have various forms. The high-level classification of digital wallets is either tangible or intangible. Another classification is based on how a digital wallet is implemented. Hardware, paper, desktop, mobile, web, browser-based, and smart contract wallets are implemented independently of a blockchain application. Hardware wallets are separate devices used by end-users to manage their digital keys and crypto-assets. Software wallets can either be embedded within the blockchain application as a primary service or a stand-alone software program, such as desktop, mobile, or web applications. It is also possible that a digital wallet can be another decentralized application [44].
4. *Permission Management.* This component is responsible for authorizing access to other blockchain components, such as joining the network, invoking smart contracts, and reading from or writing to ledgers. A permission component is essential in the consortium and private networks to encapsulate transactions and data within the consortia, and to maintain data privacy between consortia members themselves, as each member should have access to specific data only [1, 48].
5. *Incentive Mechanism.* Primarily, this mechanism applies to public blockchains where peer nodes are incentivized to contribute to the blockchain network, such as mining blocks [1]. The incentive mechanism has two components: fees and rewards. The system fee is payable by the users of a blockchain application to the system, whereas the reward fee is payable by the system to the miners.
6. *Distributed Ledgers.* These digital ledgers serve as record keepers for all transactions that occur in the network. They are synchronized between the involved peers [1, 48, 50, 51]. Blockchain platforms adopt different architectures of distributed ledgers. There are mainly two architectures of the distributed ledgers: single- and multi-ledger. Not all distributed ledgers are chains of blocks, although most are. For example, Corda is a chain of transactions. A valid transaction in Corda is confirmed and attached in real-time to the chain of transactions it depends on, instead of bundling a set of unrelated transactions in a block. Referencing the dataset, eight out of nine blockchains are chain-based. Moreover, not all distributed ledgers are chain-based. There are other distributed ledgers where a set of transactions or blocks are linked in a Direct Acyclic Graph DAG [52]. An example of a DAG-based distributed ledger is IoTA's ledger known as the Tangle [44]. Thus, distributed ledgers could be a chain of blocks, a chain of transactions, a DAG of blocks, or

a DAG of transactions. World state databases are used to store the state of the digital ledgers.

7. *Consensus.* These mechanisms are used to agree on the validity of transactions between trustless entities. These protocols have a significant impact on the performance of a blockchain network [53]. Consensus is a mechanism to achieve agreement among distributed peers on a single value or state. Different consensus mechanisms have been applied in public and private network settings [54]. For example, computation-based protocols, such as PoW, rely heavily on a node's computation power. Voting-based protocols rely on the number of votes cast by peer nodes in the network, such as PBFT. Many other algorithms, such as Proof of Stake (PoS) and DPoS, attempt to overcome the power-intensiveness limitation of the computation-based algorithms in the public networks. Although most consensus algorithms are hardware-independent, there is an exception such as Proof of Elapsed Time (PoET) which requires trusted hardware such as SGX or ARM TrustZone.
8. *Peer-to-Peer (P2P) Networks.* A blockchain network is a communication medium that allows participants to communicate in a P2P manner [1, 10, 48]. There are three main types of blockchain networks: public, private, and consortium. Public blockchains are permissionless networks that offer equal rights to all peers to read from and write to the distributed ledger. Private blockchains encapsulate transactions and data within a single organization. Read and write privileges are managed using a permission management component. An extension of this private setting allows multiple organizations to participate as a consortium.
9. *Peer Nodes.* These nodes are the hardware and software representations of users participating in the blockchain network. A node can be any electronic device with the required technical specifications to perform its job within a blockchain network, including servers, laptops, mobile phones, and IoT devices [44]. Three main types of nodes perform ledger-oriented operations: full, pruned, and lightweight nodes. A significant difference is the ability of a node to participate in the transaction validation and block generation. These features require a full copy of the distributed ledger, as in full nodes, or at least a full copy of transaction headers, plus a partial detailed copy of some transactions, as in pruned nodes. However, lightweight nodes store only transaction headers and do not participate in consensus and block generation processes. Lightweight nodes represent the nodes of an application's end users. Although full and pruned nodes perform the same job within a blockchain network, they are distinguished because full nodes are mandatory components for the existence and operation of a blockchain network. However, pruned nodes are a good choice for other peers with insufficient storage capacity to store the full history of the network. The implementation of these nodes is platform-independent. Any hardware device with minimum requirements for hosting and operating a full or pruned node can be used. In addition, a blockchain network may use service nodes for network-oriented operations to coordinate the general workflow between other nodes, such as proxy servers.

5. The architectural pattern language

Based on our understanding of core blockchain appliation compoments and their interactions, this section proposes an architectural pattern language for blockchain application development. The pattern language contains 12 patterns, classified into three categories, consisting of seven *structural* patterns, one *interactional* and four *transactional* patterns. Each pattern category represents a specific architectural view that helps blockchain application developers to focus on one design aspect. Table 2 summarizes these patterns and show their related patterns and their relationships. These patterns are described in detail in the following subsections.

5.1. Structural view

This view abstracts the logical and physical dissemination of the blockchain application components. From this perspective, a blockchain application is viewed as a number of distributed components among multiple tiers in a networked ecosystem. It addresses the concerns of how the components of a blockchain application are decoupled from and interact with each other, with a focus on their physical arrangement. There are seven patterns related to this view.

5.1.1. Distributed client-server pattern

Summary: This pattern provides a global view of a blockchain application as a constituent of distributed client and server nodes.

Context: A blockchain application is a software application that delivers data and services from its backend servers to its frontend clients as multiple client-server relationships.

Solution: A blockchain application can be depicted from a high level of abstraction into a client-server architecture, as shown in Figure 1. The client is a lightweight node comprising the application front-end served by the server-side components. The server side is the backend of the blockchain application. By design, a single archival full blockchain node should be able to serve multiple client peer nodes. Oppositely, multiple full nodes can serve a client peer node. This pattern introduces a multiple client-server relationship. Any node in a blockchain application can be interchangeably a client, a server, or both. Thus, when some nodes are clients, other nodes become servers for those clients. This allows for various architectural patterns for the server and client sides.

Known Uses: - Dtube is a video-sharing application. Its client-side consists of the web application, and its server-side consists of centralized and distributed logic and data servers. When the client-side run as a full node, it can serve other clients on Steem blockchain, and thus, it becomes a server.

Table 2
An overview of the proposed architectural pattern language for blockchain application development

Architectural View	Pattern	Related Pattern	Relationship
Structural	Distributed Client-Server	Distributed Backend	Composition
		Layered Client Application	Composition
	On-Chain/Off-Chain	Blockchain Broker	Complement
	Distributed Backend	Distributed Client-Server	Composition
		P2P On-Chain Backend	Composition
		Distributed Off-Chain Backend	Composition
		Centralized Off-chain Backend	Composition
	P2P On-Chain Backend	Distributed Backend	Composition
		Distributed Off-Chain Backend	Complement
		Centralized Off-chain Backend	Complement
	Centralized Off-Chain Backend	Distributed Backend	Composition
		P2P On-Chain Backend	Complement
		Distributed Off-Chain Backend	Alternative
	Distributed Off-Chain Backend	Distributed Backend	Composition
		Centralized Off-chain Backend	Alternative
		P2P On-Chain Backend	Composition
	Blockchain Broker	On-Chain/Off-Chain	Complement
Interactional	Layered Client Application	Distributed Client-Server	Composition
Transactional	On-Chain Pipe-Filter	Chain-of-Blocks	Complement
	Replicated Repository	Chain-of-Blocks	Instantiation
	Chain-of-Blocks	Chain Fork	Variant
		On-Chain Pipe-Filter	Complement
		Replicated Repository	Instantiation
	Chain Fork	Chain-of-Blocks	Variant

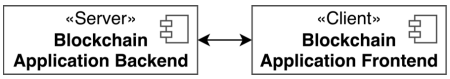

Figure 1: The Distributed client-server pattern for blockchain applications.

- PUML is a health and fitness loyalty program. Its client-side consists of a mobile application, and its server-side consists of centralized and distributed logic and data servers. Since mobile devices are not adequate to store the full copy of the ledger, it is unlikely to be a server.
- CryptoFlip Cars is a card-trading game. Its client-side consists of a web application, and its server-side consists of centralized and distributed logic and data servers. When the client-side run as a full node, it can serve other clients on Ethereum blockchain, and thus, it becomes a server.

Related Patterns: Communication between the server and client components is achieved through the Blockchain Broker pattern. The server-side can be realized using Distributed Backend pattern as a combination of P2P On-Chain Backend pattern, Distributed Off-Chain Backend pattern, and Centralized Off-Chain Backend pattern. The client side can be realized using the Layered Client Application pattern.

5.1.2. On-Chain/Off-chain pattern

Summary: This pattern addresses the physical organization of blockchain application components as a multi-tier system.
Context: A software application utilizes blockchain technology.

Solution: This architecture provides a global view of blockchain applications. A blockchain application is separated into five logical and physical tires, as shown in Figure 2. First, the presentation tier hosts the application front-end and handles the static and dynamic presentation of the user interface. It can be a remote web server or a local device such as smartphones and wearable. Second, the off-chain logical tier coordinates the interaction between the off-chain data and presentation tiers. It performs off-chain computation and business logic on data collected from the presentation tier, and passes data from and to the off-chain data tier. Source code files can be written in conventional server-side languages and run in dedicated frameworks. Third, the off-chain data tier is where off-chain data are stored and retrieved from, to the off-chain logic tier and eventually to the presentation tier. It can be a database or a file system. Fourth, smart contracts perform on-chain computations and business logic through the on-chain logical tier. Finally, the on-chain data tier stores on-chain data in distributed ledgers. This architectural pattern is sophisticated in that each tire has several variant architectures.

The blockchain application components can be deployed in a plug-and-play manner. There are three main deployment approaches for the different tiers: pure local, pure remote, and a combination of both. In pure local deployment, all tiers are deployed to a local machine and connected to the test network of the blockchain platform. This approach is suitable for pre-production and testing environments to evaluate application features and assess smart contract execution. In

pure remote deployment, all tiers are deployed on remote machines and are accessed through RPCs. For example, the off-chain tier is deployed on a remote web server, and the on-chain tier is deployed to a remote blockchain node as a service, such as DappNode. Usually, this approach is used in a production environment in which the application is deployed on the production network of the blockchain platform. Hybrid deployment can occur in several ways.

For example, a remote off-chain local on-chain deployment occurs when a blockchain node hosts the on-chain tier on a local computer machine connected to the blockchain network, and the off-chain tier is hosted on remote servers. In contrast, when the off-chain tier is a mobile app installed on users' phone devices connected to the blockchain network, the on-chain tier is deployed to remote nodes as a local off-chain remote on-chain approach. The hybrid deployment is a customized approach that allows the articulation of different deployment styles for each tier.

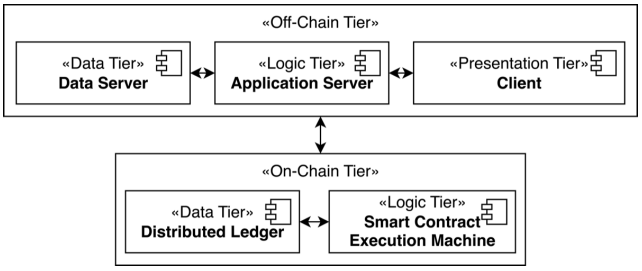

Figure 2: The On-Chain/Off-chain pattern for blockchain applications.

Known Uses: - Dtube is a video-sharing application. It consists of presentation and off-chain logic tiers as a web application hosted on a centralized server, an off-chain data tier that adopts centralized and distributed third-party data stores, and on-chain logic and data tiers hosted on the Steem and Hive blockchains.
- PUML is a health and fitness loyalty program. It consists of presentation and off-chain logic tiers combined in a mobile application hosted on users' mobile devices and an off-chain data tier that adopts third-party data stores. PUML's on-chain logic and data tiers are hosted on the EOS blockchain and PUML's private side chain.
- CryptoFlip Cars is a card-trading game application. It consists of a presentation, off-chain data, and off-chain logic tiers combined as a web application hosted on a centralized server. The application's on-chain logic and data tiers are hosted on the Ethereum blockchain.
Related Patterns: Communication between the off-chain and on-chain components is achieved through the Blockchain Broker pattern.

5.1.3. Distributed backend pattern

Summary: This pattern articulates the backend of a blockchain application as a constituent of on-chain and off-chain server components.
Context: Articulating the backend components of a blockchain application as a distributed Client-Server application.
Solution: The server-side of a blockchain application consists of off-chain and on-chain backend components, as shown in Figure 3. The off-chain backend consists of application and data servers that provide services related to the external world (of a blockchain system) to the application frontend, such as off-chain business logic processing and data management. The on-chain backend consists of peer nodes that consensually provide blockchain-related services such as immutable data storage and transparent business logic processing. These services can be on single or multiple chains.

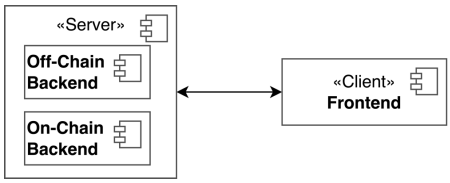

Figure 3: The Distributed backend pattern for blockchain applications.

Known Uses: - Dtube is a video-sharing application. Its off-chain backend consists of application and data servers, and its on-chain backend is the Steem blockchain.
- PUML is a health and fitness loyalty program. Its off-chain backend consists of application and data servers, and its on-chain backend is the EOS blockchain and PUML sidechain.
- CryptoFlip Cars is a card-trading game application. Its off-chain backend consists of application and data servers, and its on-chain backend is the Ethereum blockchain.
Related Patterns: This pattern can be realized using a combination of P2P On-Chain Backend pattern, Distributed Off-Chain Backend pattern, and Centralized Off-Chain Backend pattern. The communication between the off-chain and on-chain components is achieved through the Blockchain Broker pattern.

5.1.4. P2P on-chain backend pattern

Summary: This pattern addresses concerns regarding the decentralized communication of backend components within the internal environment of the blockchain.
Context: A software application utilizes blockchain technology.
Solution: This is the typical architecture of blockchain systems, as shown in Figure 4. A blockchain network consists of multiple decentralized peer nodes, in which each node has the same capabilities and responsibilities. For example, blockchain participants have equal reading and writing rights to the blockchain distributed ledger. However, permission components can manage these rights in certain cases. The propagation of blocks and transactions typically follows a gossip strategy. There are different protocols adopted by blockchain networks to manage the propagation of data between peers. For example, Ethereum Mainnet nodes run different protocols, such as the Light Ethereum Subprotocol (LES) used by lightweight clients and the Ethereum Wire Protocol (ETH) used by full nodes. Hyperledger Fabric

implements a data dissemination protocol based on gRPC and protocol buffers.

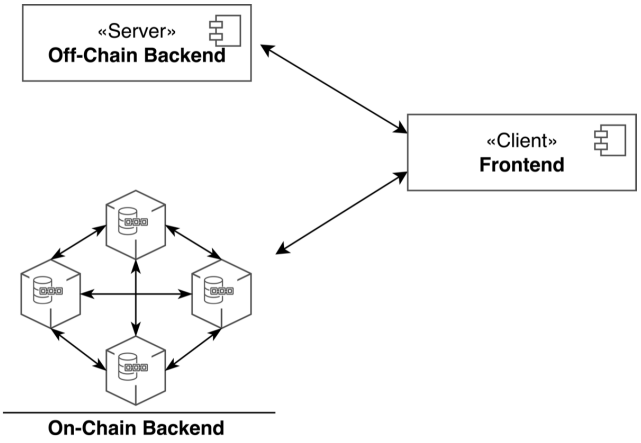

Figure 4: The P2P on-chain backend pattern for blockchain applications.

Known Uses: - Dtube is a video-sharing application. Its on-chain backend is the Steem blockchain, which is a P2P network.
- PUML is a health and fitness loyalty program. Its on-chain backend is the EOS blockchain and PUML sidechain, which are P2P networks.
- CryptoFlip Cars is a card trading game application. Its on-chain backend is the Ethereum blockchain, which is a P2P network.
Related Patterns: This pattern should be complemented by the Distributed Off-Chain Backend pattern, Centralized Off-Chain Backend pattern, or a combination of both.

5.1.5. Distributed off-chain backend pattern

Summary: This pattern addresses concerns regarding the decentralized communication of backend components within the external environment of the blockchain.
Context: A blockchain application utilizes off-chain services as a distributed system.
Solution: Off-chain backend servers provide their services through distributed servers to distribute access to web apps and off-chain data, as shown in Figure 5. For example, a blockchain application may use an off-chain P2P file system, such as the BitTorrent File System (BTFS) and Interplanetary File System (IPFS), for web hosting and data storage. Instead, the application may utilize a P2P file-sharing system for either web hosting or data storage and use a centralized server for the other as a three-tier client-server architecture.
Known Uses: - Aragon is an Ethereum-based service for creating Decentralized Autonomous Organizations (DAOs) that uses IPFS for their data.
- Augur is an Ethereum-based marketplace that uses IPFS for client application hosting and distribution.
- Dtube is a Steem-based video sharing platform that uses IPFS and BTFS for users' video hosting.

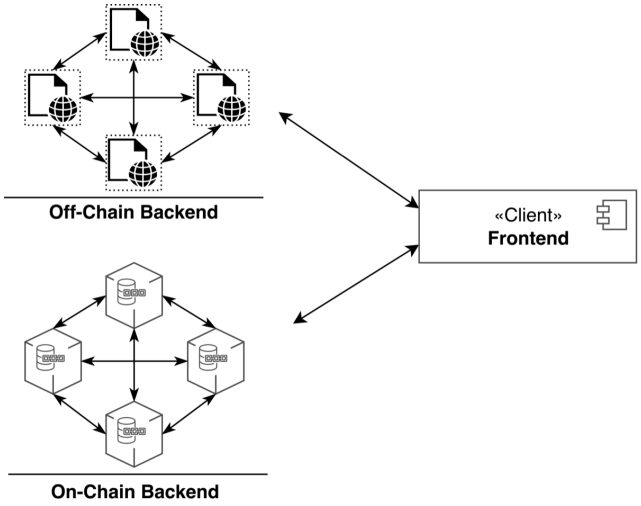

Figure 5: The distributed off-chain backend pattern for blockchain applications.

Related Patterns: This pattern should complement the P2P On-Chain Backend pattern. It can also be accompanied by the Centralized Off-Chain Backend pattern.

5.1.6. Centralized off-chain backend pattern

Summary: This pattern addresses concerns regarding the centralized communication of backend components within the external environment of the blockchain.
Context: A blockchain application utilizes off-chain services as a centralized system.
Solution: Off-chain backend servers provide their services through centralized server machines, as shown in Figure 6. For example, a blockchain application may use a single server for the off-chain application frontend and data or can use separate servers for each as a three-tier client-server architecture.

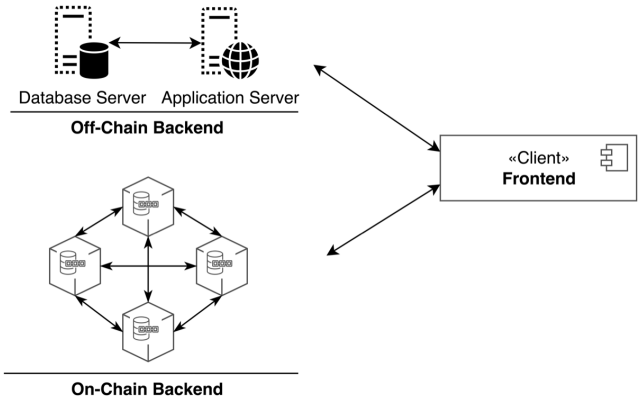

Figure 6: The centralized off-chain backend pattern for blockchain applications.

Known Uses: - CryptoFlip Cars is an Ethereum-based game that uses centralized servers for web hosting and data storage.

- Dtube uses centralized servers for web hosting and data storage.
- Geon is a POA-based mobile game hosted locally by mobile device users.

Related Patterns: This pattern should be complemented by the P2P On-Chain Backend pattern. It can also be accompanied by the Distributed Off-Chain Backend pattern.

5.1.7. Blockchain broker pattern

Summary: This pattern addresses concerns about communicating the decoupled components of a blockchain application as a distributed system.

Context: A blockchain application consists of decoupled off-chain and on-chain components.

Solution: The Blockchain Broker enables the communication between the decoupled components in distributed systems. As shown in Figure 7, there are two types of blockchain broker:

1. Side-chain broker: By design, blockchains are heterogeneous and not able to communicate with each other since each blockchain has its own set of rules and tools that makes communication difficult. Side-chain broker allows blockchain interoperability through cross-chain bridges. It enables the transfer of assets between one blockchain (on-chain) and another (side-chain). Typically, a side-chain broker involves locking or burning tokens through a smart contract on the source chain and unlocking or minting equivalent tokens on the destination chain through another smart contract.
2. Off-chain broker: This type of service communicate blockchain and client applications. A client API handles the input from the client UI and passes it to a dedicated blockchain endpoint. The blockchain endpoint node passes the input data to the blockchain and returns output data to the client API, which in turn passes the data to the UI. The client API and blockchain endpoint API are service broker components that connect off-chain and on-chain components.

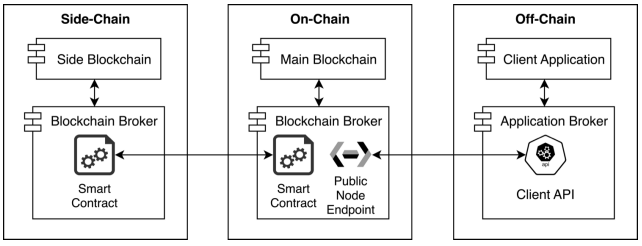

Figure 7: The Blockchain broker pattern for blockchain applications.

Known Uses: - Dtube is a Steem-based video-sharing application. It uses the JSON-RPC web service and the RPC endpoint of Steem's public service node to communicate the off-chain components with the Steem blockchain.

- CryptoFlip Cars is an Ethereum-based game application. It uses the JSON-RPC web service and the RPC endpoint of Ethereum public nodes to communicate the off-chain components with the Ethereum blockchain.
- PUML is a health and fitness loyalty program. It uses the JSON-RPC web service and the RPC endpoint of EOS public nodes to communicate the off-chain components with the EOS blockchain and the PUML side chain.

Related Patterns: This pattern shows hoe the different tiers in the On-Chain/Off-Chain pattern are connected.

5.2. Interactional view

This view abstracts how the individual components of a blockchain application can exchange messages while retaining their autonomy. From this perspective, a blockchain application is viewed as a set of independent components that interact with each other within an application context. It addresses the concerns of how the components of a blockchain application are decoupled from and interact with each other, with a focus on the logical arrangement. A layered pattern is related to this view, as described below.

5.2.1. Layered client application pattern

Summary: This pattern addresses the decomposition of blockchain application components into interacting parts as complex, heterogeneous entities.

Context: A software application utilizes blockchain technology.

Solution: A blockchain application can be represented in a generic layered architecture, as shown in Figure 8. A typical blockchain application architecture consists of the following layers:

- *Application layer:* This layer represents the use cases of blockchain applications such as intelligent transportation, supply chain traceability, artificial intelligence, robotics, unmanned aerial vehicles, intelligent manufacturing, business process management, enterprise transformation, insurance processing, risk management, cryptocurrency exchange, games, gambling, social networking, community reputation systems, rights management, and data ownership.
- *Presentation layer:* This layer is related to front-end components that can interact with the blockchain layer via client APIs. This layer consists of a typical UI and digital wallet. It serves the purpose of providing an intuitive way for end-users to use blockchain applications.
- *Support layer:* This layer provides supportive services for off-chain computation and data. Hosting services are required for hosting and operating client applications. External storage services can also be provided through this layer. This layer is extended with an IoT service layer in IoT applications. It consists of components that provide IoT-related services, such as processing and analyzing sensor data and transferring them to other layers.

- *Client Interface layer:* This layer consists of web services and API gateway components. The blockchain RPC endpoints receive client API requests from the external layers, transfer them to the internal layers, and then send the response to the external layers. The client interface layer is developed with libraries and APIs in conventional UI programming languages, such as JavaScript.
- *Business logic layer:* This layer handles the on-chain computations of the blockchain application. An essential component is a smart contract. The other components within this layer are smart contract wallets and application tokens.
- *Consensus layer:* This layer comprises the necessary components to coordinate on-chain operations and ensure data integrity. It consists of a consensus protocol, transaction management, and incentive mechanisms. Transaction management refers to the mechanisms used to manage on-chain workflows, such as transaction pools, reward pools, notaries, witnesses, and miners.
- *Distributed ledger layer:* This layer is the data layer. It consists of the design and implementation of the digital ledger. It is a design decision to determine which data should be on the main chain and which should be on the side-chain or off-chain. For example, some application data can be stored off-chain, but the blockchain ledger is either on-chain or side-chain.
- *Network layer:* This layer implements the P2P communication between the users of the blockchain application. It allows the messaging and ledger data transmission between nodes to keep the ledger synchronized among all the participants.
- *Infrastructure layer:* This layer is the foundation layer for blockchain applications. It consists of blockchain platforms and their native tokens. The platforms can be viewed as the execution environment of blockchain applications. The P2P network, consensus, and ledger are soft components usually managed using a blockchain platform.
- *Physical layer:* This layer represents the primary hardware components of other layers. It consists of different types of nodes: full, pruned, lightweight, and service nodes. It also includes a smart contract execution machine and world state database servers. Fog/Edge computing nodes can be included in IoT-based applications.
- *Security and privacy layer:* This layer provides security services to the applications and users. This layer includes user authentication, membership management, access control, and permission management components.

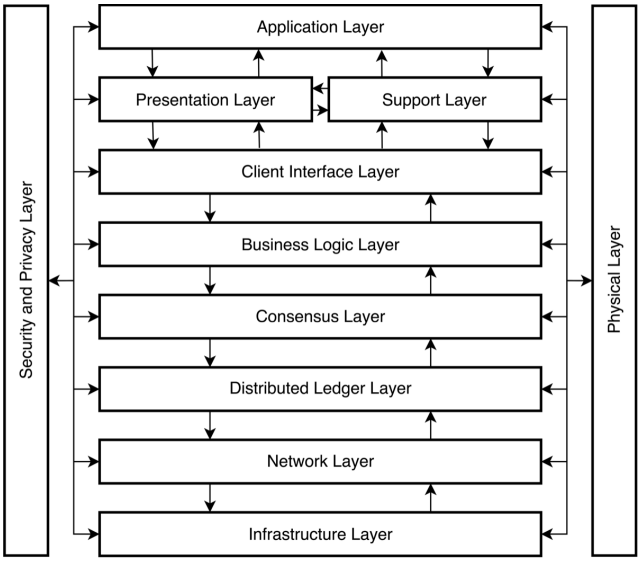

Figure 8: The Layered client application pattern for blockchain applications.

There are three concrete examples of this pattern: consumer-centric, enterprise-centric, and IoT-based applications. The following text describes each architecture. The differences in the elements between the architectures are indicated in their corresponding figures.

1. *Consumer-Centric application*: Blockchain technology allows independent entities to collaborate confidentially without relying on central intermediaries. The architecture of Consumer-Centric applications consists of 11 logical layers: application, presentation, support, client interface, business logic, consensus, distributed ledger, network, infrastructure, security and privacy, and physical layers. The components within each layer are shown in Figure 9. This type of application benefits highly from the public blockchain networks.
2. *Enterprise-Centric application*: Blockchain technology can ensure the integrity of the processes and data with security and cryptography by offering a trusted peer-to-peer exchange and automated execution of business contracts. The architecture of Enterprise-Centric applications consists of 11 logical layers: application, presentation, support, client interface, business logic, consensus, distributed ledger, network, infrastructure, security and privacy, and physical layers. The components within each layer are shown in Figure 10. Enterprise businesses can benefit highly from permission management components and multichannel networks supported by private and consortium blockchain platforms. One of the notable components of Enterprise-Centric applications is an integrated legacy system. This type of application can exchange data with an organization's existing systems, such as Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM).

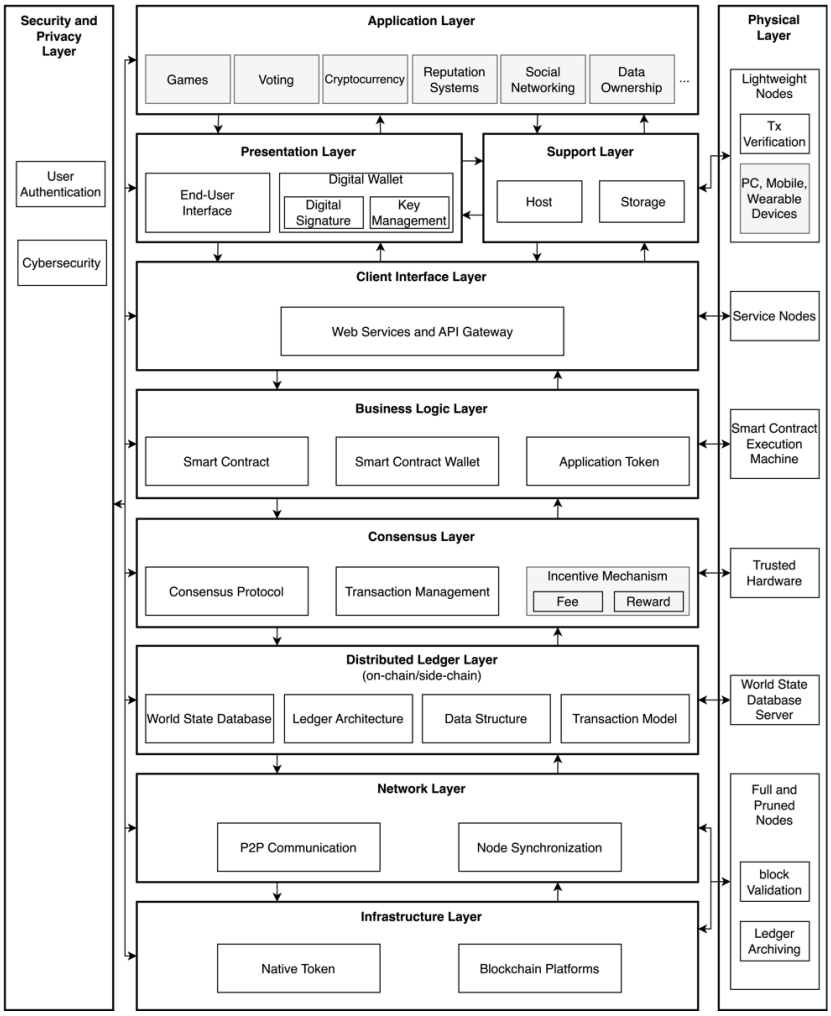

Figure 9: The layered client pattern for consumer-centric application.

3. *IoT-Based application pattern*: Blockchain technology can effectively secure machine-to-machine transactions, ensure their safety and efficiency, reduce costs, promote resources sharing, and automate IoT complex workflow. The architecture of IoT-Based applications consists of 12 logical layers: application, presentation, support, IoT service, client interface, business logic, consensus, distributed ledger, network, infrastructure, security and privacy, and physical layers. The components within each layer are shown in Figure 11. The IoT service layer is an enabler of this pattern. It can collect and transmit the processed data from sensor devices to the application support layer via cloud-computing services. One of the notable components of IoT-Based applications is thing smart contracts. This type of contract coordinates on-chain machine-to-machine transactions. The design of the application logic determines the necessity of such contracts. For example, machine transactions can be processed off-chain, and the resulting data are passed to the blockchain. Thus, the thing smart contract becomes inessential. This type of application benefits from the Fog/Edge computing services and nodes. Smart objects such as data sensors, smartphones, and wearable are essential components. More granular permissions may be required for different machines and user accounts.

Known Uses: - Dtube is an example of Consumer-Centric applications. It is a Steem-based video-sharing application that rewards users for sharing their content, commenting on, and upvoting others' content.
- KYC-Chain is an example of Enterprise-Centric applications. It is an Ethereum-based customer onboarding application for verifying customers' identities and managing the customer lifecycle.
- Hawk E-Scooter is an example of IoT-Based applications. It is a Klaytn-based decentralized, shared scooter travel platform.
Related Patterns: This pattern is a realization of the Distributed Client-Server pattern.

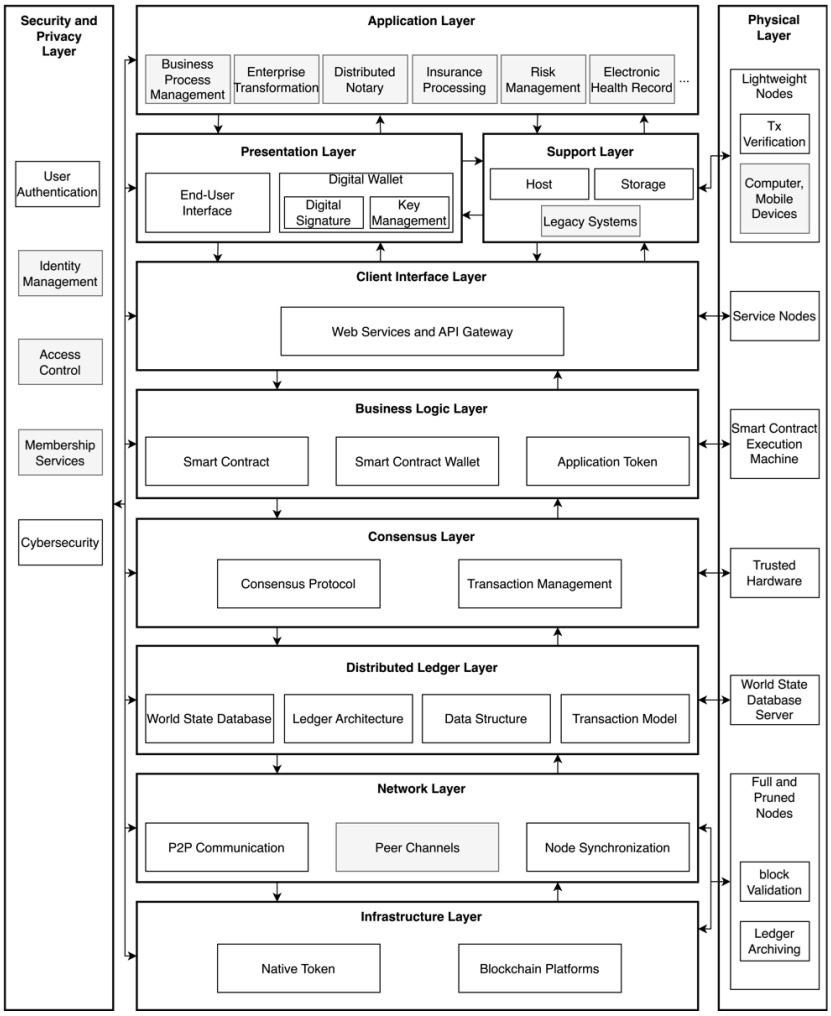

Figure 10: The layered client pattern for enterprise-centric application.

5.3. Transactional view

This view abstracts how blockchain application components successively process transactions as a persistent, immutable, shared data. In this view, a blockchain application is viewed as a number of subsequent transfers of the input transaction data. It addresses the concerns of how transactions on blockchain are committed and ordered. It also addresses how distributed ledgers are accessed, updated, and distributed. Four patterns are related to this view, as described below.

5.3.1. On-chain pipe-filter pattern

Summary: This pattern addresses concerns regarding how data are transferred from a client to a shared ledger while retaining data integrity and confidentiality.

Context: A software application utilizes blockchain services as a distributed ledger.

Solution: Data in the blockchain are processed in the form of transactions. A transaction is a single instruction constructed and cryptographically signed by a client external to the scope of the blockchain. The process of adding new transactions to the ledger is a Pipe-Filter architecture, as shown in Figure 12. A transaction passes several filters (validators/miners) from the client (source) to the ledger (sink). The number and types of filters differ among blockchain systems. Initially, a transaction must be cryptographically signed at the client application using the user's private key. The signed transactions then enter a pending pool. They are then filtered by validators/miners. Subsequently, a certain number of valid transactions are bundled into a block. Block size determines the number of transactions included within a block. The block size also differs between the blockchain systems. Finally, a valid block is added to the ledger and synchronized with all participants.

Known Uses: - Transactions of Ethereum-based applications are submitted to Ethereum's pending transaction pool and then validated by miners, who are rewarded for generating new blocks.

- Transactions of Hyperledger Fabric-based applications are validated by endorsing, ordering, and committing peers.

- Transactions of Steem-based applications are submitted to Steem's pending transaction pool and then validated by witnesses, who are rewarded for producing new blocks.

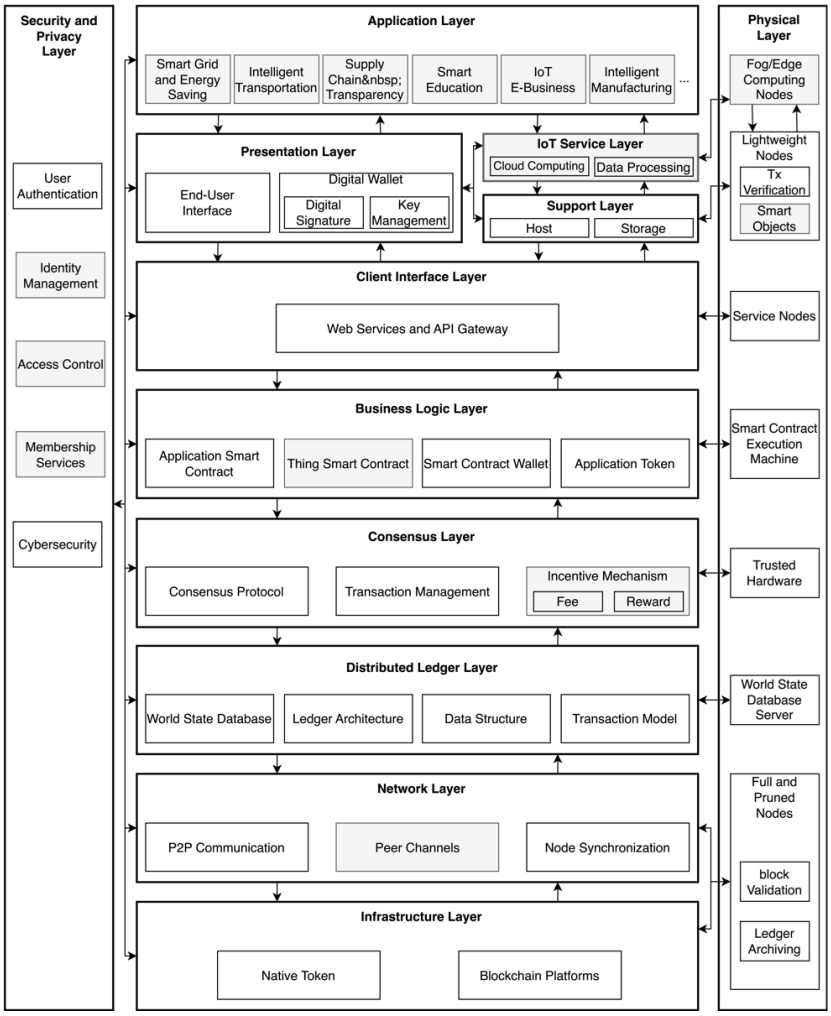

Figure 11: The layered client pattern for IoT-based application.

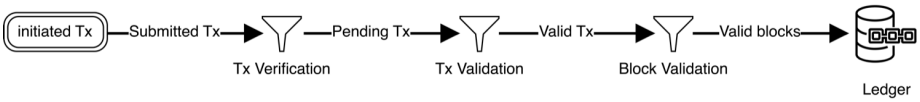

Figure 12: The on-chain pipe-filter pattern for blockchain applications.

Related Patterns: The generated blocks can be linked as in the Chain-of-Blocks pattern

5.3.2. Replicated repository pattern

Summary: This pattern addresses concerns regarding distributing on-chain data as a replicated ledger.

Context: A software application utilizes blockchain services as a distributed ledger.

Solution: A blockchain is a digital archive of transactions that are replicated and broadcast across the blockchain network nodes. Every time a new block is committed, a copy of that block is added to the ledgers of the other participants, as shown in Figure 13. Because each participant, except light clients, has its copy, ledgers on the blockchain are replicated rather than shared. Although nominated public nodes share their replica with other nodes and clients in permissionless networks, any newly added block is replicated and broadcast among all network participants. Reading from and writing to these replicas are granted equally to all nodes in the network. In permissioned networks, broadcasting is performed for a subset of the network participants based on their granted permissions. This permission-based replication allows complex ledger structures in enterprise applications. In such cases, each member in a consortium has a digital ledger, and any subset of members of the consortium can have a mutual replica. Finally, there is a master replica among all the consortium members. The replicated records are also at varying transparencies to ensure data confidentiality in the private environment.

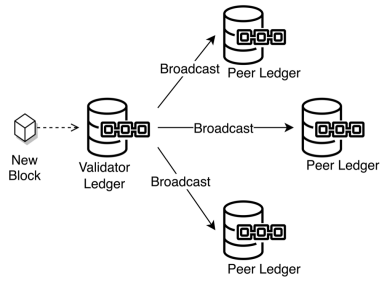

Figure 13: The replicated repository pattern for blockchain applications.

Known Uses: - The Ethereum network is permissionless, and its digital ledger is replicated among all Ethereum nodes as a single-ledger-based architecture.
- The Hyperledger Fabric network is permissioned, and its digital ledger is replicated per channel, allowing for a multi-ledger-based architecture.
- The Corda network is permissioned and allows for shared ledger structures.
Related Patterns: The data in a replicated ledger can be linked as in the Chain-of-Blocks pattern.

5.3.3. Chain-of-blocks pattern

Summary: This pattern addresses concerns regarding structuring on-chain data.
Context: A software application utilizes blockchain services as a distributed ledger.
Solution: Confirmed transactions in the blockchain are grouped into blocks. Each block has a unique number, known as the block height. A block typically consists of two parts common to all blockchains: a block header and block data. The block header contains, among other information, a link to its preceding block, which allows the blocks to be chained in a strict order. Such a link is typically a hash of the preceding block. A block hash is cryptographically derived from the block's data. The block data contain a list of confirmed transactions. Thus, a block can be searched for by its block number or hash. Additional parts are customized per blockchain platform for the block structure. Usually, these data are not included in the block hash calculation. The time interval, or block interval, is the time allowed to add a new block to the chain. This time differs between blockchain systems based on the blockchain protocol specifications. For example, Steem produces blocks every three seconds. Figure 14 illustrates this pattern.

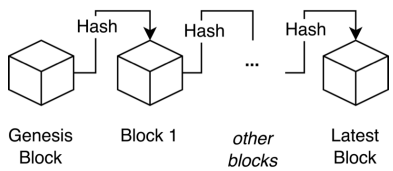

Figure 14: The chain-of-blocks pattern for blockchain applications.

Known Uses: - Ethereum is chain-based, and its block structure consists of a block header, block data, and a list of ommers. The blocks are connected to a single chain.
- Hyperledger Fabric is chain-based, and its block structure consists of a block header, data, and metadata. The blocks are connected to a restricted shared chain.
- Klaytn is chain-based, and its block structure consists of a block header, block data, and a list of uncles. The blocks are connected to a single chain.
Related Patterns: The chain of blocks in the blockchain is synchronized between the network participants as a Replicated Repository pattern. When more than one block has the same block height, that is, they are mined at nearly the same time, there could be a Chain Fork pattern.

5.3.4. Chain fork pattern

Summary: This pattern addresses concerns regarding structuring on-chain data on conflict.
Context: A software application utilizes blockchain services as a distributed ledger.
Solution: To create a blockchain, the chain of blocks receives information from the blocks that preceded them. The block relationship can be viewed as a parent-child relationship, where each parent block has one child. The hash of the parent block is included in the header of the child block. Blockchain participants must follow common rules to maintain the history of the blockchain. When the participants are not in agreement or the rules are changed, alternative chains may emerge as chain forks. A chain fork occurs by splitting a chain of blocks into two paths. There are two main methods for such a split, as shown in Figure 15.

The first occurs unintentionally at the consensus level when different blockchain participants disagree on the block validity. For example, when two blocks are mined simultaneously, they include the hash of their parent block and produce two different hashes for their blocks' data. This affects the validity of the chain. Therefore, only one block must be integrated into the chain. To decide which child block to include, the participants are allowed a temporary split with both conflicting child blocks. The miners then continue producing blocks based on each conflicting block. The conflicting block with the longest chain is then accepted. The other conflicting block becomes a stale/ommer block, and all its dependent blocks are discarded and returned to the pending transaction pool for validation to be included in the chain. The effect of a consensus-level chain fork on a blockchain application is that it affects referential integrity with transaction finality.

The second occurs intentionally at the protocol level when the blockchain is upgraded or changes its rules. This change causes a permanent split in the chain. When a blockchain changes its rules, the existing chain is considered invalid in the new network. Thus, an alternative chain should be created according to the new rules. Therefore, the validator nodes must be upgraded to the new network. If some validator nodes insist on following the old rules, a

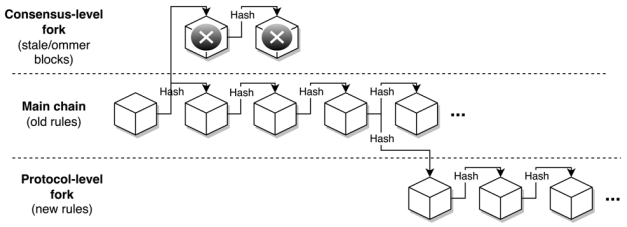

Figure 15: The chain fork pattern for blockchain applications.

permanent divergence version of the chain is created. One path follows the new version of the blockchain, and the other follows the old one. The effect of a protocol-level chain fork on a blockchain application is that the application may remain on the old chain, or move completely to the forked chain, or can be available on both chains.

Known Uses: - Ethereum was forked into Ethereum Classic (old) and Ethereum (new) as a protocol level split.
- Hive is forked from Steem as a protocol level split.
- Dtube is a Steem-based video-sharing application runs on both blockchains: Steem and its forked chain Hive.

Related Patterns: The forked chain is a Chain-of-Blocks synchronized between the network participants as a Replicated Repository pattern.

6. Evaluating the proposed pattern language

In this section, we evaluate our proposed pattern language by using it to describe the architectures of three real-world blockchain applications. These three applications are: Cryptokitties [55], which is a consumer-centric gaming application on Ethereum; IBM Food Trust (IFT) [56], which is a enterprise-centric Software-as-a-Service (SaaS) solution; Actifit [57], which is an IoT-based Steem-based fitness tracker mobile application aimed at rewarding physical activity.

6.1. Using the pattern language to describe the architecture of Cryptokitties

CryptoKitties is a popular marketplace game for buying, selling, and breeding virtual cats called CryptoKitties. The architecture of CryptoKitties can be described using the proposed architectural pattern language for blockchain applications as follows:

6.1.1. Structural view of CryptoKitties architecture

At a high level of abstraction, CryptoKitties is a Distributed Client-Server application whose server-side is built as a combination of the P2P On-Chain Backend and Centralized Off-Chain Backend patterns. CryptoKitties's P2P on-chain backend is the Ethereum Mainnet. The application's centralized off-chain is realized by hosting the CryptoKitties frontend web application on a centralized hosting service and storing the off-chain data in a relational database server.

The CryptoKitties's presentation tier compromises a web application hosted on Cloud Flare servers. Game logic is distributed between the off-chain and on-chain logic tiers. Off-chain data correspond to the off-chain logic and the client application. These data are stored in PostgreSQL. On-chain data correspond to the on-chain logic are stored in the Ethereum distributed ledger. CryptoKitties can be deployed using various approaches. The off-chain tier is deployed remotely from a game player's perspective. It compromises the presentation, off-chain logic, and off-chain data tiers. The on-chain logic tier is deployed on the EVM.

The deployment of the on-chain data tier depends on the player's choice. If the player wants to interact with CryptoKitties as a lightweight client, it is not required to deploy its own Ethereum node. However, if the player is an Ethereum miner, it must have its own Ethereum node. However, it is possible to deploy its full node as a remote Node-as-a-Service instead of being deployed locally. The former is a pure remote deployment, and the latter is a hybrid approach in the form of remote off-chain local on-chain.

This analysis realizes the application of the On-Chain/Off-Chain pattern on the CryptoKitties application. The off-chain and on-chain tiers communicate via RPCs initiated from the CryptoKitties client application to the Ethereum Mainnet public node endpoints using JSON-RPC web services. This communication realizes the Blockchain Broker pattern.

6.1.2. Interactional view of CryptoKitties architecture

CryptoKitties is a gaming application; thus, its architecture is mapped to the layered client application pattern of the consumer-centric application. Figure 16 illustrates the mapping of CryptoKitties architecture to the architecture of Consumer-Centric applications.

6.1.3. Transactional view of CryptoKitties architecture

Game transactions are initiated by players at the CryptoKitties' web front. These transactions involve buying, selling, transferring, or generating CK tokens. The client first signs the transaction by using the player's digital keys via their digital wallet. The signed transaction is then broadcasted to Ethereum mempool, a pool for pending transactions, and made available for miners to be validated. Miners are not necessarily players in CryptoKitties. Usually, they are application-independent and contribute to the Ethereum network through block validations. The miner then selects the transactions for validation. Usually, those with higher gas fees are selected because they turn into higher mining rewards. Subsequently, a set of validated transactions is bundled to generate a block. Once this block is validated, it is added to the miner's ledger. The miner is then rewarded with the deserved amount of Ether for the validated blocks. There are no separate transaction pipes for the different applications in Ethereum.

Ethereum has a single ledger for all transactions that occurred on its Mainnet, and miners select the set of transactions to validate; thus, CryptoKitties transactions may be bundled with transactions from other applications within a single block. Every time a new block is generated, it is attached to its preceding blocks in the Ethereum Mainnet in the

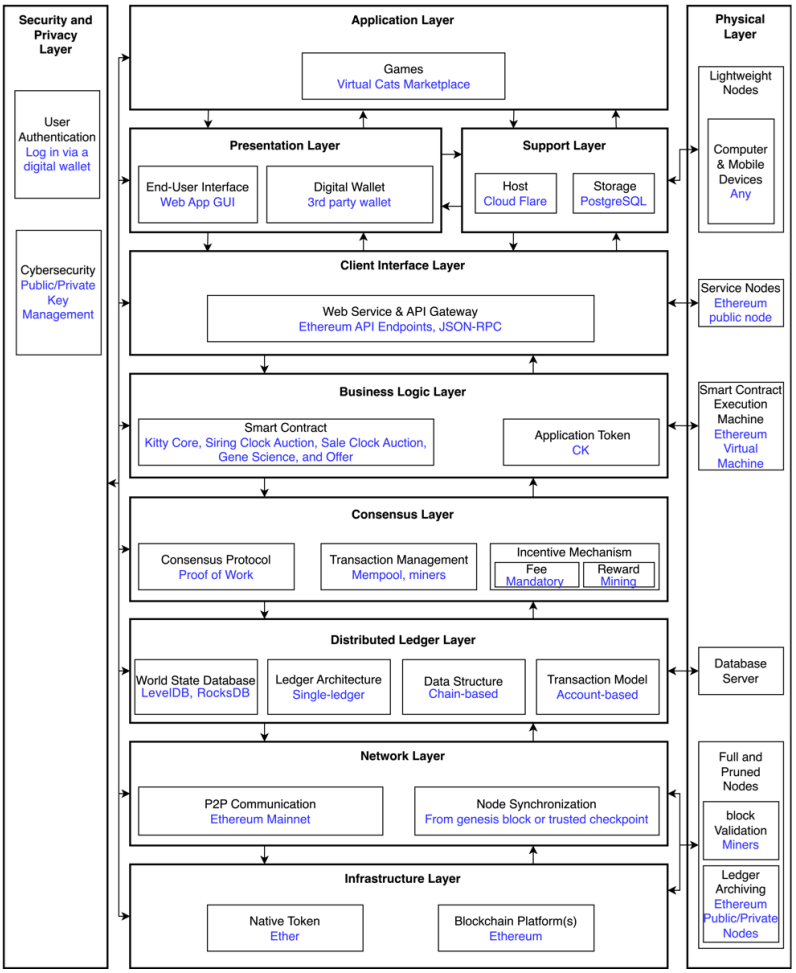

Figure 16: The Mapping of CryptoKitties architecture to the layered pattern of Consumer-Centric application.

form of the Chain-of-Blocks pattern. It is then broadcasted to all Ethereum participants, even though these nodes are not CryptoKitties players, as in the Replicated Repository pattern.

6.2. Using the pattern language to describe the architecture of IBM Food Trust

IBM Food Trust (IFT) is a Software-as-a-Service (SaaS) solution built on Hyperledger Fabric and powered by the IBM Blockchain, a Platform-as-a-Service (PaaS) for Hyperledger Fabric. This application is aimed at facilitating food supply chain management (IBM Food Trust 2020). It provides controlled information sharing and convenient data publishing through a permission-based shared view of food information. The architecture of the IFT can be described using the proposed architectural pattern language for blockchain applications as follows:

6.2.1. Structural view of IBM Food Trust architecture

At a high level of abstraction, IFT is a Distributed Client-Server application where its server-side is built as a combination of the P2P On-Chain Backend and Distributed Off-Chain Backend patterns. The IFT's P2P on-chain backend is the Hyperledger Fabric network. The application's distributed off-chain backend is realized by hosting the IFT frontend on IBM Cloud, providing the solution as a service.

The IFT's presentation tier comprises a web application hosted in the IBM Cloud. The application logic is distributed between the off-chain and on-chain logic tiers. Off-chain data correspond to data from ERP systems. These data are stored in the external data stores of an organization. On-chain data correspond to the on-chain transactions and are stored in the Hyperledger Fabric restricted shared ledger of the food supply chain network. Because the IFT is a SaaS powered by a PaaS, it is deployed in a purely remote deployment approach. The application's off-chain components are accessed via the IBM Cloud portal. The IFT's on-chain components are deployed on the IBM Blockchain.

This analysis realizes the application of the On-Chain/Off-chain pattern on the IFT application. The off-chain and on-chain tiers communicate via RPCs initiated from the IFT client application to the Hyperledger Fabric network using Fabric Gateway, IFT REST APIs, and Fabric gRPC web services. This communication realizes the Blockchain Broker pattern.

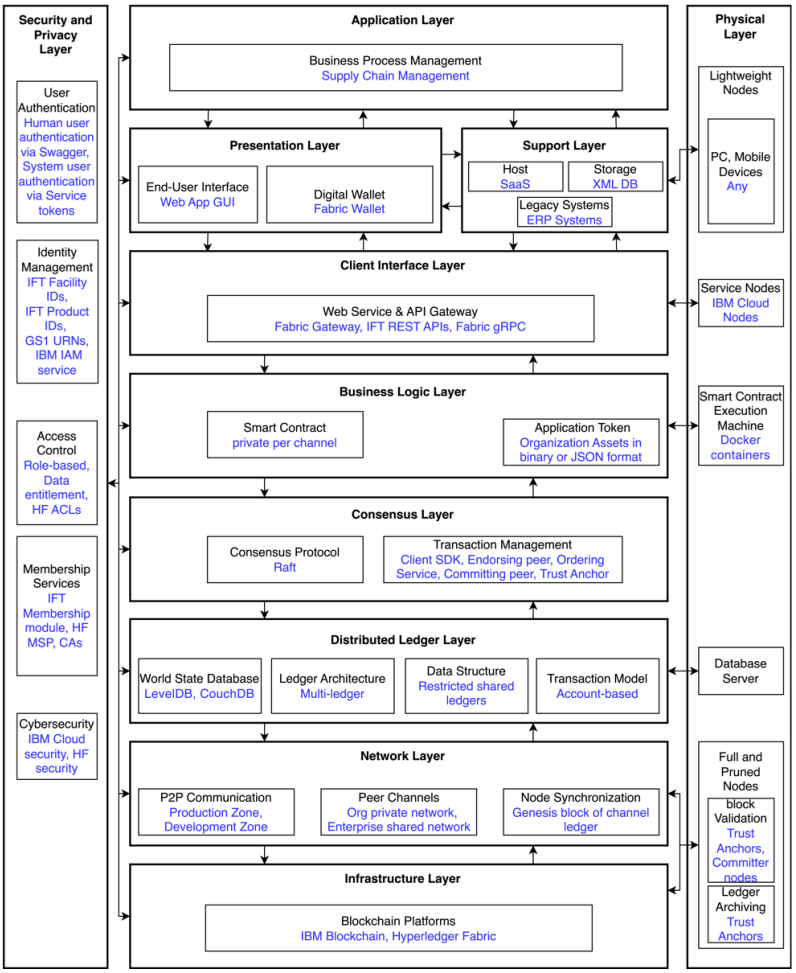

Figure 17: The Mapping of IBM Food Trust architecture to the layered pattern of Enterprise-Centric application.

6.2.2. Interactional view of IBM Food Trust architecture

The IFT is a food supply chain management application; thus, its architecture is mapped to the layered client application pattern of the enterprise-centric application. Figure 17 illustrates the mapping of IFT application architecture to the architecture of Enterprise-Centric applications.

6.2.3. Transactional view of IBM Food Trust architecture

The IFT transactions are initiated by organization members of the food supply chain through the IFT's web front. When the user initiates a transaction, the IFT client application first verifies the transaction using client SDK. It uses the user's cryptographic keys to sign the transaction and broadcasts this transaction proposal to the organization channel's endorser peers. Next, the endorsing peers verify each transaction proposal for the form of the transaction, submission newness, validity of the signature, and authorization of the submitter to perform a write operation on the channel. Approved transactions are signed by the endorsing peers, accompanied by read sets from the current state database, and sent back to the client. The IFT client verifies endorser peers' signatures and compares the received responses.

The application then composes the transaction proposal and the response into a transaction message. This message and channel ID are broadcast to the ordering service. The ordering service receives transactions from all channels in the food supply chain network. For each channel, the ordering service chronologically orders transactions and creates blocks of transactions. The transactions within a block are validated and tagged as valid or invalid. These blocks are sent to all the participants in the channel. The peers append and replicate the block to the channel's chain. Finally, write sets are committed to the current state database for each valid transaction in the block.

Each organizational member in the supply chain has its channel. Every time a new block is generated in the channel, it is attached to its preceding blocks in the same channel in the form of the Chain-of-Blocks pattern. It is then broadcast to the privileged participants in a channel-based Replicated Repository pattern.

6.3. Using the pattern language to describe the architecture of Actifit

Actifit is a Steem-based fitness tracker mobile application aimed at rewarding physical activity. It incentivizes people to become active and healthy. It introduces the "Proof of Activity" concept for reward on Steem. Users must provide proof of their daily movements and receive rewards accordingly. Actifit is available as Android and iOS mobile applications with support for Fitbit and Apple smartwatches [58]. The architecture of Actifit can be described using the proposed architectural pattern language for blockchain applications as follows.

6.3.1. Structural view of Actifit architecture

At a high level of abstraction, Actifit is a Distributed Client-Server application where its server-side is built as a combination of P2P On-Chain Backend pattern and Centralized Off-Chain Backend pattern. Actifit's p2p on-chain backend is the Steem Mainnet. The application's centralized off-chain is realized by hosting the Actifit frontend mobile application and its associated off-chain logic and data on centralized mobile devices.

The Actifit's presentation tier compromises a fitness-tracking mobile app for Android and iPhone OS smartphones. The application logic is distributed between the off-chain and on-chain logic tiers. Off-chain data correspond to off-chain computation and the client applications. These data mainly track the users' physical steps collected using the device's sensors and paired wearables. Depending on the collected data, the application calculates the rewards. The application stores its data using SQLiteDB. The application uses AWS to process sensor data and update the user's account on the Steem blockchain accordingly to the calculated rewards. On-chain data correspond to on-chain transactions, such as managing the user's account balance and redeeming the reward tokens stored on the Steem distributed ledger.

Actifit is deployed in a hybrid approach as a local off-chain remote on-chain. The off-chain tier is deployed on the users' mobile devices. It comprises the presentation, off-chain logic, and off-chain data tiers. On-chain logic is executed natively on the Steem platform. Thus, the on-chain logic and data tiers are deployed on Steem public nodes. This analysis realizes the application of the On-Chain/Off-chain pattern on the Actifit application. The off-chain and on-chain tiers communicate via RPCs initiated from the Actifit client application to the Steem Mainnet public node endpoints using JSON-RPC web services. This communication realizes the Blockchain Broker pattern.

6.3.2. Interactional view of Actifit architecture

Actifit is a sensor-based health-service application; thus, its architecture is mapped to the layered client application pattern of the IoT-based application. Figure 18 illustrates the mapping of Actifit Android app architecture to the architecture of IoT-Based applications.

6.3.3. Transactional view of Actifit architecture

User activities are recorded automatically using the sensors on the user's mobile device. Alternatively, activity data can be automatically synchronized using a paired smartwatch. The application generates activity report cards daily. The user can post the card to the Steem network to get rewarded via the mobile app interface. The user can also read, comment, and upvote on others' reports, where each is a transaction. In either case, the submitted transactions are signed using the user's private key. Then, the transaction is broadcast to the transaction mempool on the Steem network, pending validation by Steem witnesses.

A witness is a node elected by the community of Steem and is responsible for transaction and block validation. The witness selects a set of pending transactions for validation. The user is rewarded for its validated activities from an assigned reward pool. The reward for the daily activity report is auto-evaluated according to a multitude of factors: activity count from 5000 to 10000 (25%), report card content quality (20%), community engagement (20%), moderator review on the report card (10%), and the user rank from 0 to 100 (25%). The rewards are in HIVE, STEEM, and AFIT tokens, plus partner tokens such as SPORT and APX, tokens of the Sport Talk Social platform. The validated transactions are bundled into one block, signed by the witness's private key. The valid block is then appended to the witness's ledger. There are no separate transaction pipes for different applications in Steem.

Steem has a single ledger for all transactions that occurred on its network, and it is up to the witnesses to select the set of transactions to validate; thus, Actifit transactions may be bundled with transactions from other applications within a single block. Every time a new block is generated, it is attached to its preceding blocks in the Steem ledger in the form of the Chain-of-Blocks pattern. It is then broadcast to all Steem participants, even though these nodes are not Actifit users, as in the Replicated Repository pattern. Actifit adapts its existence on Steem and Hive blockchains simultaneously by mirroring users' accounts and rewards on both chains as a result of the Chain Fork pattern of Steem and Hive blockchains.

7. Research limitation

As blockchain technology is constantly evolving, new patterns may emerge from new developments. Consequently, the pattern language presented in this paper is a work in progress. Furthermore, the patterns in this language will inevitably evolve, where some patterns may be replaced by new patterns.

Another limitation is concerned with the evaluation of the pattern language, as we have only selected three blockchain applications to demonstrate the language. We have not involved industries in evaluation.

Finally, a major limitation of our pattern language is that none of the proposed patterns describes the design of the architectural components that support security and trust. Our future work will focus on this aspect.

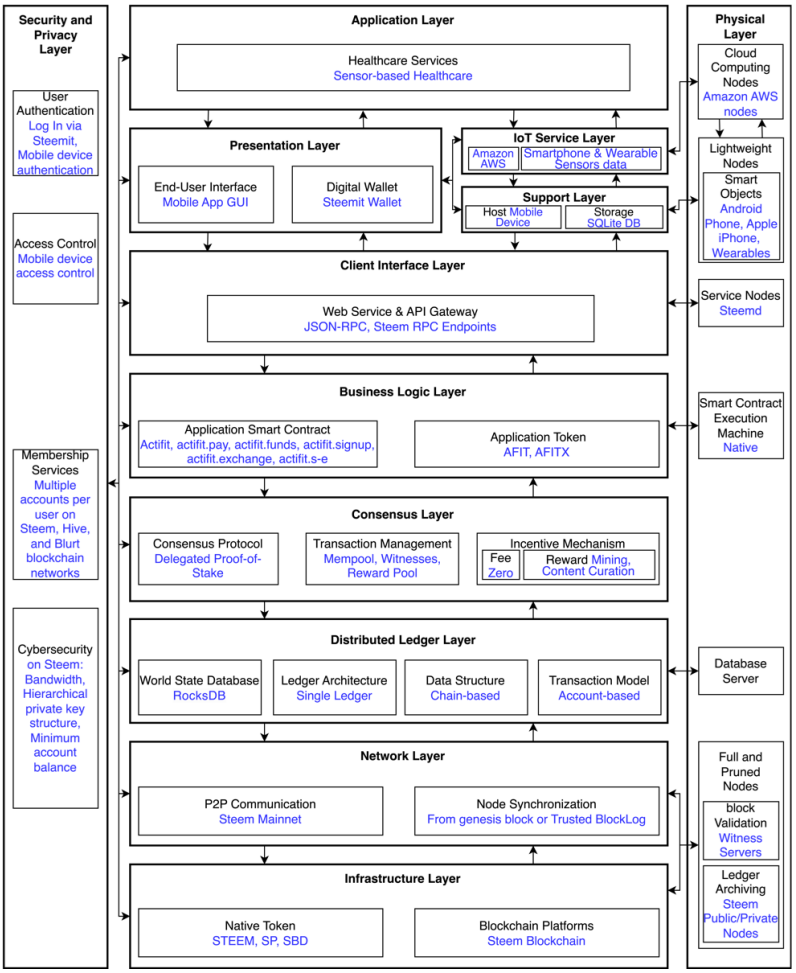

Figure 18: The Mapping of Actifit architecture to the layered pattern of IoT-Based application.

8. Conclusion

This paper proposed an architectural pattern language for the design of blockchain applications. The 12 patterns in this pattern language were derived from 400 industrial blockchain applications. The pattern language describes the architecture of a blockchain application from three architectural views. Within each view, this pattern language provides a set of architectural patterns and their relationships. The proposed patterns are intended to be general and applicable to a broad range of blockchain applications. The feasibility of this pattern language was demonstrated through three case studies by illustrating how it can be used to describe the architecture of real-world blockchain applications.

Declaration of competing interest

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

Data availability

Data used for this research are available in Zenodo [43].

Acknowledgements

Fouzia Alzhrani wishes to gratefully acknowledge a scholarship she received from King Abdulaziz University that enables her to pursue a Ph.D. degree at the University of Manchester.

References

[1] X. Xu, I. Weber, M. Staples, Architecture for blockchain applications, Springer, 2019. doi:10.1007/978-3-030-03035-3.

[2] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Decentralized Business Review (2008) 21260.

[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. doi:https://doi.org/10.1016/j.tele.2018.11.006.

[4] U. Bodkhe, S. Tanwar, K. Parekh, P. Khanpara, S. Tyagi, N. Kumar, M. Alazab, Blockchain for industry 4.0: A comprehensive review, IEEE Access 8 (2020) 79764–79800. doi:10.1109/ACCESS.2020.2988579.

[5] P. K. Sharma, J. H. Park, Blockchain based hybrid network architecture for the smart city, Future Generation Computer Systems 86 (2018) 650–655. doi:10.1016/j.future.2018.04.060.

[6] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng, V. C. M. Leung, Decentralized applications: The blockchain-empowered software system, IEEE Access 6 (2018) 53019–53033. doi:10.1109/ACCESS.2018.

2870644.

[7] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, H. Wang, Blockchain challenges and opportunities: A survey, International journal of web and grid services 14 (2018) 352–375.

[8] A. A. Monrat, O. Schelén, K. Andersson, A survey of blockchain from the perspectives of applications, challenges, and opportunities, IEEE Access 7 (2019) 117134–117151. doi:10.1109/ACCESS.2019.2936094.

[9] D. Garlan, Software architecture: a roadmap, in: Proceedings of the Conference on the Future of Software Engineering, 2000, pp. 91–101.

[10] T. A. Syed, A. Alzahrani, S. Jan, M. S. Siddiqui, A. Nadeem, T. Alghamdi, A comparative analysis of blockchain architecture and its applications: Problems and recommendations, IEEE access 7 (2019) 176838–176869.

[11] C. Alexander, A pattern language: towns, buildings, construction, Oxford university press, 1977.

[12] L. Zhao, L. Macaulay, J. Adams, P. Verschueren, A pattern language for designing e-business architecture, Journal of Systems and Software 81 (2008) 1272–1287.

[13] F. Alzhrani, A Collection of Papers on Blockchain Business Applications, 2023. URL: https://doi.org/10.5281/zenodo.7564584. doi:10.5281/zenodo.7564584.

[14] R. Mühlberger, S. Bachhofner, E. Castelló Ferrer, C. Di Ciccio, I. Weber, M. Wöhrer, U. Zdun, Foundational oracle patterns: Connecting blockchain to the off-chain world, in: A. Asatiani, J. M. García, N. Helander, A. Jiménez-Ramírez, A. Koschmider, J. Mendling, G. Meroni, H. A. Reijers (Eds.), Business Process Management: Blockchain and Robotic Process Automation Forum, Springer International Publishing, Cham, 2020, pp. 35–51.

[15] N. Six, C. Negri Ribalta, N. Herbaut, C. Salinesi, A blockchain-based pattern for confidential and pseudo-anonymous contract enforcement, in: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2020, pp. 1965–1970. doi:10.1109/TrustCom50675.2020.00268.

[16] M. Wöhrer, U. Zdun, Design patterns for smart contracts in the ethereum ecosystem, in: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2018, pp. 1513–1520. doi:10.1109/Cybermatics_2018.2018.00255.

[17] M. Wohrer, U. Zdun, Smart contracts: security patterns in the ethereum ecosystem and solidity, in: 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE), 2018, pp. 2–8. doi:10.1109/IWBOSE.2018.8327565.

[18] X. Xu, C. Pautasso, L. Zhu, Q. Lu, I. Weber, A pattern collection for blockchain-based applications, in: Proceedings of the 23rd European Conference on Pattern Languages of Programs, EuroPLoP '18, Association for Computing Machinery, New York, NY, USA, 2018. URL: https://doi.org/10.1145/3282308.3282312. doi:10.1145/3282308.3282312.

[19] M. Bartoletti, L. Pompianu, An empirical analysis of smart contracts: Platforms, applications, and design patterns, in: M. Brenner, K. Rohloff, J. Bonneau, A. Miller, P. Y. Ryan, V. Teague, A. Bracciali, M. Sala, F. Pintore, M. Jakobsson (Eds.), Financial Cryptography and Data Security, Springer International Publishing, Cham, 2017, pp. 494–509.

[20] Q. Lu, X. Xu, H. D. Bandara, S. Chen, L. Zhu, Patterns for blockchain-based payment applications, in: 26th European Conference on Pattern Languages of Programs, EuroPLoP'21, Association for Computing Machinery, New York, NY, USA, 2022. URL: https://doi.org/10.1145/3489449.3490006. doi:10.1145/3489449.3490006.

[21] X. Yang, Blockchain-based supply chain finance design pattern, in: 2021 13th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2021, pp. 200–203. doi:10.1109/IHMSC52134.2021.00053.

[22] M. Müller, N. Ostern, M. Rosemann, Silver bullet for all trust issues? blockchain-based trust patterns for collaborative business processes, in: A. Asatiani, J. M. García, N. Helander, A. Jiménez-Ramírez, A. Koschmider, J. Mendling, G. Meroni, H. A. Reijers (Eds.), Business Process Management: Blockchain and Robotic Process Automation Forum, Springer International Publishing, Cham, 2020, pp. 3–18.

[23] W. Yánez, R. Bahsoon, Y. Zhang, R. Kazman, Architecting internet of things systems with blockchain: A catalog of tactics, ACM Trans. Softw. Eng. Methodol. 30 (2021). doi:10.1145/3442412.

[24] X. Xu, H. Dilum Bandara, Q. Lu, I. Weber, L. Bass, L. Zhu, A decision model for choosing patterns in blockchain-based applications, in: 2021 IEEE 18th International Conference on Software Architecture (ICSA), 2021, pp. 47–57. doi:10.1109/ICSA51549.2021.00013.

[25] Y. Liu, Q. Lu, G. Yu, H.-Y. Paik, L. Zhu, Bgra: A reference architecture for blockchain governance, 2022. URL: https://arxiv.org/abs/2211.04811. doi:10.48550/ARXIV.2211.04811.

[26] I. Weber, Q. Lu, A. B. Tran, A. Deshmukh, M. Gorski, M. Strazds, A platform architecture for multi-tenant blockchain-based systems, in: 2019 IEEE International Conference on Software Architecture (ICSA), 2019, pp. 101–110. doi:10.1109/ICSA.2019.00019.

[27] M. Wöhrer, U. Zdun, Architectural design decisions for blockchain-based applications, in: 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2021, pp. 1–5. doi:10.1109/ICBC51069.2021.9461109.

[28] T. Min, H. Wang, Y. Guo, W. Cai, Blockchain games: A survey, in: 2019 IEEE Conference on Games (CoG), 2019, pp. 1–8. doi:10.1109/CIG.2019.8848111.

[29] K. Wu, Y. Ma, G. Huang, X. Liu, A first look at blockchain-based decentralized applications, Software: Practice and Experience 51 (2021) 2033–2050. doi:https://doi.org/10.1002/spe.2751.

[30] I. A. Qasse, J. Spillner, M. Abu Talib, Q. Nasir, A study on Đapps characteristics, in: 2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS), 2020, pp. 88–93. doi:10.1109/DAPPS49028.2020.00010.

[31] Y. Zeng, Y. Zhang, Review of research on blockchain application development method, Journal of Physics: Conference Series 1187 (2019) 052005. doi:10.1088/1742-6596/1187/5/052005.

[32] G. Chen, B. Xu, M. Lu, N.-S. Chen, Exploring blockchain technology and its potential applications for education, Smart Learning Environments 5 (2018) 1. doi:10.1186/s40561-017-0050-x.

[33] T.-T. Kuo, H.-E. Kim, L. Ohno-Machado, Blockchain distributed ledger technologies for biomedical and health care applications, Journal of the American Medical Informatics Association 24 (2017) 1211–1220. doi:10.1093/jamia/ocx068.

[34] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, F. Wang, Blockchain-enabled smart contracts: Architecture, applications, and future trends, IEEE Transactions on Systems, Man, and Cybernetics: Systems 49 (2019) 2266–2277. doi:10.1109/TSMC.2019.2895123.

[35] J. Al-Jaroodi, N. Mohamed, Blockchain in industries: A survey, IEEE Access 7 (2019) 36500–36515. doi:10.1109/ACCESS.2019.2903554.

[36] I. Konstantinidis, G. Siaminos, C. Timplalexis, P. Zervas, V. Peristeras, S. Decker, Blockchain for business applications: A systematic literature review, in: W. Abramowicz, A. Paschke (Eds.), Business Information Systems, Springer International Publishing, Cham, 2018, pp. 384–399.

[37] B. A. Tama, B. J. Kweka, Y. Park, K.-H. Rhee, A critical review of blockchain and its current applications, in: 2017 International Conference on Electrical Engineering and Computer Science (ICECOS), 2017, pp. 109–113. doi:10.1109/ICECOS.2017.8167115.

[38] N. Gaur, J. Cuomo, J. S. Arun, Blockchain for Business, 1 ed., Addison-Wesley Professional, 2019.

[39] D. Dujak, D. Sajter, Blockchain Applications in Supply Chain, Springer International Publishing, Cham, 2019, pp. 21–46. URL: https://doi.org/10.1007/978-3-319-91668-2_2. doi:10.1007/978-3-319-91668-2_2.

[40] S. Aggarwal, R. Chaudhary, G. S. Aujla, N. Kumar, K.-K. R. Choo, A. Y. Zomaya, Blockchain for smart communities: Applications, challenges and opportunities, Journal of Network and Computer Applications 144 (2019) 13–48. doi:https://doi.org/10.1016/j.jnca.2019.06.018.

[41] A. S. Musleh, G. Yao, S. M. Muyeen, Blockchain applications in smart grid–review and frameworks, IEEE Access 7 (2019) 86746–86757. doi:10.1109/ACCESS.2019.2920682.

[42] J. Bao, D. He, M. Luo, K.-K. R. Choo, A survey of blockchain applications in the energy sector, IEEE Systems Journal 15 (2021) 3370–3381. doi:10.1109/JSYST.2020.2998791.

[43] F. Alzhrani, A collection of industry-developed blockchain-based applications, 2020. URL: https://doi.org/10.5281/zenodo.4268953. doi:10.5281/zenodo.4268953.

[44] F. E. Alzhrani, K. A. Saeedi, L. Zhao, A taxonomy for characterizing blockchain systems, IEEE Access 10 (2022) 110568–110589. doi:10.1109/ACCESS.2022.3214837.

[45] P. Avgeriou, U. Zdun, Architectural patterns revisited - A pattern language, in: A. Longshaw, U. Zdun (Eds.), EuroPLoP' 2005, Tenth European Conference on Pattern Languages of Programs, Irsee, Germany, July 6-10, 2005, UVK - Universitaetsverlag Konstanz, 2005, pp. 431–470.

[46] M. Jaiswal, Software architecture and software design, International Research Journal of Engineering and Technology (IRJET) e-ISSN (2019) 2395–0056.

[47] 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. doi:https://doi.org/10.1016/j.future.2019.12.019.

[48] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, J. Wang, Untangling blockchain: A data processing view of blockchain systems, IEEE Transactions on Knowledge and Data Engineering 30 (2018) 1366–1385. doi:10.1109/TKDE.2017.2781227.

[49] M. Di Angelo, G. Salzer, Identification of token contracts on ethereum: standard compliance and beyond, International Journal of Data Science and Analytics (2021). doi:10.1007/s41060-021-00281-1.

[50] H. Pervez, M. Muneeb, M. U. Irfan, I. U. Haq, A comparative analysis of dag-based blockchain architectures, in: 2018 12th International Conference on Open Source Systems and Technologies (ICOSST), 2018, pp. 27–34. doi:10.1109/ICOSST.2018.8632193.

[51] H.-Y. Paik, X. Xu, H. M. N. D. Bandara, S. U. Lee, S. K. Lo, Analysis of data management in blockchain-based systems: From architecture to governance, IEEE Access 7 (2019) 186091–186107. doi:10.1109/ACCESS.2019.2961404.

[52] N. Kannengießer, S. Lins, T. Dehling, A. Sunyaev, Trade-offs between distributed ledger technology characteristics, ACM Comput. Surv. 53 (2020). doi:10.1145/3379463.

[53] L. Ismail, H. Materwala, A review of blockchain architecture and consensus protocols: Use cases, challenges, and solutions, Symmetry 11 (2019). doi:10.3390/sym11101198.

[54] H. S. Jennath, S. Asharaf, Survey on blockchain consensus strategies, in: A. Kumar, M. Paprzycki, V. K. Gunjan (Eds.), ICDSMLA 2019, Springer Singapore, Singapore, 2020, pp. 637–654.

[55] CryptoKitties, Cryptokitties | collect and breed digital cats!, 2022. URL: https://www.cryptokitties.co/.

[56] I. F. Trust, About IBM Food Trust, Report, IBM Corporation, 2020. URL: https://www.ibm.com/downloads/cas/EX1MA1OX.

[57] Actifit, Actifit - rewarding your everyday activity, 2022. URL: https://actifit.io/.

[58] Actifit, Actifit: rewarding your everyday activity, Whitepaper v2.0.1, Actifit, 2021. URL: https://actifit.io/whitepaper/Actifit_White_Paper.pdf.

Chapter 5

A Process-Aware Framework to Support Process Mining from Blockchain Applications

Research Objective

Objective 4: To develop a framework to support process mining from blockchain applications

Thesis Context

This chapter uses the results in Chapter 3 to characterize a process-aware blockchain application. The implementation of the proposed framework in this chapter results in *BELA*, which is then used to examine the Industry-developed applications and produce the event logs datasets in Chapter 2. This chapter is based on a paper that has been submitted to Engineering Applications of Artificial Intelligence. The full reference of this paper is as follows:

F. Alzhrani, K. Saeedi, and L. Zhao, "A Process-Aware Framework to Support Process Mining from Blockchain Applications," *Engineering Applications of Artificial Intelligence*, Under Review.

CRediT authorship contribution statement

Fouzia Alzhrani: Conceptualization, Data Curation, Methodology, Software, Writing - Original Draft, Writing - Review & Editing. **Kawther Saeedi**: Conceptualization, Supervision, Writing - Review & Editing. **Liping Zhao**: Conceptualization, Supervision, Writing - Review & Editing.

A Process-Aware Framework to Support Process Mining from Blockchain Applications

Fouzia Alzhrani[a,b], Kawther Saeedi[a] and Liping Zhao[b,*]

[a]King Abdulaziz University, Jeddah, Saudi Arabia
[b]The University of Manchester, Manchester, United Kingdom

ARTICLE INFO

Keywords:
Blockchain
Machine Learning
Process Mining
Event Data Analytics
Decision Support
Rule-Based System
Business Process Intelligence

ABSTRACT

Blockchain technology offers the ability of a trusted peer-to-peer exchange and automated execution of business contracts through smart contracts. This offer needs to be examined on the operational level to gain objective insights of workflow embedded within smart contracts. Several studies have been conducted to demonstrate the application of Process Mining (PM) techniques to blockchain application event data. However, research on process awareness of blockchain applications seems to be lacking. This study proposes a framework to support mining business processes from blockchain applications. The framework consists of two modules: Process Awareness Recognizer (PAR) and Event Log Generator (ELG). PAR is a rule-based classifier to assess the process awareness of a given blockchain application. ELG is an automated batch processing model consisting of three methods: 1) Extractor: an algorithm for event data retrieval from blockchain networks, 2) Decoder: a data decoding algorithm to transform the extracted event data to a human-readable format, and 3) Formatter: an algorithm to produce event log files in a format that is compatible with PM tools. The framework supports Ethereum-compatible applications. It was validated by implementing a proof-of-concept application with an input set of 201 real-world applications. The results approved the feasibility and applicability of the framework. This study aims to accelerate PM practices on blockchain event data by providing a methodology to assist in selecting, retrieving, and transforming event data of blockchain applications.

1. Introduction

Blockchain technology has the potential to revolutionize e-business by delivering a new way to enforce trust among distrusted business partners. It brings new opportunities for organizations to re-imagine business models by enhancing current models and creating new ones (Mendling et al., 2018; Arun et al., 2019). As an enabler of cross-organizational processes, blockchain applications can be integrated with traditional, non-blockchain applications (Sousa and Corentin, 2019). Blockchain technology can transform the centralized structure of traditional businesses into a distributed and shared one. This helps to eliminate the complex and hierarchical management of business transactions. Business processes can be encoded into smart contracts to ensure the integrity of the processes and data with security and cryptography (Arun et al., 2019; Viriyasitavat and Hoonsopon, 2019).

This potential needs to be examined on the operational level to gain objective insights of workflow embedded within smart contracts. Process Mining (PM) enables a fact-based analysis through processing event logs by state-of-the-art algorithms, techniques, and tools (Burattin, 2015; Aalst, 2012; Reinkemeyer, 2020). PM arose as a result of merging techniques from process science and data science to enable evidence-based business process management (van der Aalst, 2016; Aalst, 2012; Burattin, 2015; dos Santos Garcia et al., 2019). It is a data-driven process-centric approach aiming to accelerate process improvements by automatically extracting knowledge from captured process traces available in a system's log files (Sousa and Corentin, 2019).

In 2012, IEEE Task Force on Process Mining published a manifesto to encourage the topic of PM. van der Aalst et al. (2012) defines PM as: "techniques, tools, and methods to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs commonly available in today's (information) systems". There are three techniques of PM: process discovery, conformance checking, and process enhancement. PM techniques use event data to discover processes, compare process variants, check conformance, analyze bottlenecks, and suggest improvements (Reinkemeyer, 2020; van der Aalst et al., 2012).

Event logs are the technical foundation and starting point of PM (van der Aalst, 2016; Aalst, 2012; Reinkemeyer, 2020). Reinkemeyer (2020) defines an event log as:"a collection of events which have taken place in order to perform a business process. Each event refers to a specific activity that took place at a certain moment in time and can be assigned to a unique case". PM algorithms require an event log to have at least three attributes: case ID, activity name, and timestamp. Case ID is a numeric identifier of a single instance of the process, activity name is a specification of what happened, and timestamp indicates the precise time an activity happened on (dos Santos Garcia et al., 2019; Reinkemeyer, 2020). These event logs are generated by process-aware systems (PASs). A PAS system is a software system that produces correlated activities as a process instance representing a

*Corresponding author
feaalzhrani@kau.edu.sa (F. Alzhrani); ksaeedi@kau.edu.sa (K. Saeedi); liping.zhao@manchester.ac.uk (L. Zhao)
ORCID(s): 0000-0002-5780-0469 (F. Alzhrani)

complete execution of a business process (van der Aalst, 2016).

Mining business processes from blockchain applications is obstructed by several challenges related to the quality of event data produced by the applications and to the process awareness of those applications in the first place. The cryptographic format and heterogeneous structures of event data generated by different smart contracts require technical expertise to extract encoded transactions data from blockchain networks and transform the data into a format adequate for PM techniques (Wirawan et al., 2021; Mühlberger et al., 2019). Several approaches for event log generation from blockchain data have been proposed in the literature. They imply three steps: extracting, decoding, and formatting event data. However, the discussion focused on the formatting step and overlooked the preceding steps, which are of equal importance. Moreover, the availability of event data is constrained by the process awareness of a blockchain application, which is under-reported by the literature.

This study aims to address these challenges in the context of Ethereum Virtual Machine (EVM)-compatible platforms. In particular, it proposes a framework to assist in identifying suitable blockchain applications for PM and in generating event logs for those applications. The framework consists of two modules: Process Awareness Recognizer (PAR) and Event Log Generator (ELG). PAR is a binary rule-based classifier aims to assess the process awareness of a given blockchain application. ELG is a batch process model, consisting of three ordered methods: Extractor, Decoder, and Formatter, aims to automated event log generation from blockchain data. Moreover, this study provides a reusable implementation of the framework which can be used for EVM-compatible applications.

The remainder of this paper is organized as follows. Section 2 reviews related studies. Section 3 represents the motivation of this study. Section 4 presents our proposed framework. Section 5 presents a proof-of-concept application of the proposed framework and analyzes the results. Section 6 highlights findings and limitations of the study. Finally, Section 7 concludes the paper.

2. Related work

There are very few studies on mining blockchain processes. The scope of these studies varies from proposing event log generation techniques from blockchains process data to demonstrating the use of extracted process data for different purposes. For example, Corradini et al. (2019) and Ivković and Luković (2021) applied PM to validating and evaluating smart contracts. The study by Di Ciccio et al. (2020) applied PM techniques to enable monitoring of business processes executed on the blockchain. Müller and Ruppel (2019) demonstrated the use of PM on platform-level data. Hobeck et al. (2021) assessed the business value of mining blockchain processes through an in-depth case study analysis of a blockchain application.

One of the studies that focused on event log generation from blockchain data is presented by M'Baba et al. (2022). The study proposes a new format for event logs known as Artifact Centric Event Log (ACEL). In ACEL, blockchain applications are viewed as artifact-centric rather than activity-centric applications. However, the proposed format is not yet supported by PM tools. The extractor by Mühlberger et al. (2019) is a proof-of-concept application written in Python. It also focused on transforming the event data retrieved from the blockchain into eXtensible Event Stream (XES) format. The tool was implemented for the transactions of the incident management process stored on Ethereum Mainnet and produced by the execution engine described in the study (Weber et al., 2016).

The event log generation method proposed by Klinkmüller et al. (2019) focused on formatting blockchain event data into XES format. First, a manifest describing how to interpret logged data from a process perspective has to be prepared, ideally by the smart contract's developer. Then, the extractor algorithm takes two command-line inputs: a blockchain's remote node URL to get log data of the smart contract and the manifest specification. Afterward, the algorithm transforms the retrieved data into XES according to the specified rules in the manifest. The extractor was implemented in JavaScript and tested using a sample real-world application.

The reviewed methods for event log generations focus on the data transformation task. Also, they require retrieving event data from the blockchain one at a time for each smart contract. The mechanism of retrieving the data from a blockchain network and the mechanism of decoding the retrieved data are overlooked. Moreover, all of the reported literature assumed that blockchain applications are process aware, an assumption that needs to be validated.

3. Challenges of blockchain-oriented process mining

3.1. Process awareness

Smart contracts' capability to efficiently express business logic depends on their operating platform. There are different logging levels supported by blockchain platforms. Generally, EVM-compatible platforms support two types of logs: system events and smart contract events. The system events are low-level logs generated by a node to monitor the node's status, independent of a business domain (Hyperledger Besu community, 2022; The go-ethereum Authors, 2022). The smart contract events are high-level custom events which are platform-independent and for a specific business case (Wood, 2022). Events in a smart contract are defined by its programmer, produced by the application, and stored in the host blockchain network. Some of these events are business-relevant that capture the business processes and workflow of the application. Thus, a process-aware EVM-based application is a system that its constituent smart contracts perform operational processes on the basis of process models interpreted by business-relevant events

in the smart contract source codes. That is, not all EVM-based applications are process-aware. For example, there are applications where the blockchain is used as a registry, without the business logic that characterizes a business process, although its smart contracts have a set of events.

3.2. Event log quality

Generating event logs from blockchain application data is challenging (Klinkmüller et al., 2019). The challenges are summarized as follows:

- *Data extraction*: It requires a clear understanding of which and where a log is stored, and access to all necessary data (Reinkemeyer, 2020). Running a local node is not simple, it requires technical expertise and computing resources. It comes with the cost of maintaining storage and network bandwidth and having to divert engineering time and resources. Alternatively, event data can be retrieved relying on remote nodes via Remote Procedure Calls (RPCs) to the blockchain network, then receiving event data as a response. A single blockchain application may have more than one smart contract. For example, the Vdice application has 49 contract addresses on Ethereum Mainnet. Therefore, at least 49 RPCs should be sent to get the event data of the application.
- *Data format*: Further effort is needed to transform the extracted data into an appropriate format for PM by decrypting the encrypted data and formulating process traces. It would be a time and energy-consuming task, especially for applications with multiple smart contracts. For example, a log record includes a topic array to describe an event. The first topic, i.e., $topics[0]$ includes the event signature, which is the event name and its parameter types hashed using keccak256 hash function. To decode the event signatures into activities of process traces, the Application Binary Interface (ABI) of a smart contract should be retrieved from the Ethereum network to get event names, calculate their signatures, and then match these signatures with $topics[0]$ in the event log.
- *Data availability*: To be able to retrieve a contract ABI, the smart contract must be verified on the blockchain network. Not all smart contracts are verified. Thus, no ABI will be available for unverified contracts. Therefore, no event logs will be available for systems with unverified smart contracts.

4. The proposed framework

The proposed framework has two objectives: 1) to support identifying the right case for PM; and 2) to automate the workflow of generating event logs from blockchain data. It consists of two modules: Process Awareness Recognizer (PAR) and Event Log Generator (ELG). Figure 1 is a graphical representation of the framework. The input of PAR is a set of blockchain applications to be examined for their process awareness, and the output is two sets of applications classified as process-aware and process-unaware. The set, or a subset, of process-aware applications are then ingested into ELG as an input set of smart contracts and the output is a set of event log files readily available for PM tools. In what follows, we describe each module.

4.1. Process awareness recognizer

The PAR module is a human-in-the-Loop model where the user (PAR-H) and the machine (PAR-M) collaborate to decide if a given blockchain application is process-aware and suitable for PM or not. That is, there could be blockchain applications that are process-aware but not suitable for PM and vise versa, as described in the consequent subsections. PAR consists of a rule base and an inference engine. In what follows, we describe each component.

4.1.1. Rule base

The *Rule Base* is based on a decision tree consisting of an ordered set of rules that characterizes a process-aware blockchain application. The tree is shown in Figure 2 with five nodes to classify applications into two classes: Valid and Not Valid. *Valid* class represents applications that are process-aware and suitable for PM, whereas *Not Valid* class represents applications that are not. Given a set of applications:

1. The first node examines the capability of operating platforms to support defining business-relevant events within programmable smart contracts. If a non-supporting platform is found, then all applications belong to that platform are excluded from a further analysis.
2. The second node examines if a network is reachable and the user is authorized to access event data on the network or not. If an inaccessible network is found, then all applications deployed on that network are excluded from a further analysis.
3. If the network is accessible, then the verification status of the application's smart contracts should be examined, as in the third node. Unverified smart contracts are excluded from further analysis.
4. The fourth node scans the smart contract ABI for defined events. Contracts with no defined events are excluded.
5. The fifth node determines if the detected events are business-relevant or not.
6. At the end, only applications that pass all the conditions are valid. The inclusion rule can be expressed as:
   IF (platform supports smart contract events) $\wedge$ (network is accessible) $\wedge$ (smart contract is verified) $\wedge$ (smart contract has events) $\wedge$ (events are business-relevant)
   THEN the application is Valid
   ELSE the application is Not Valid

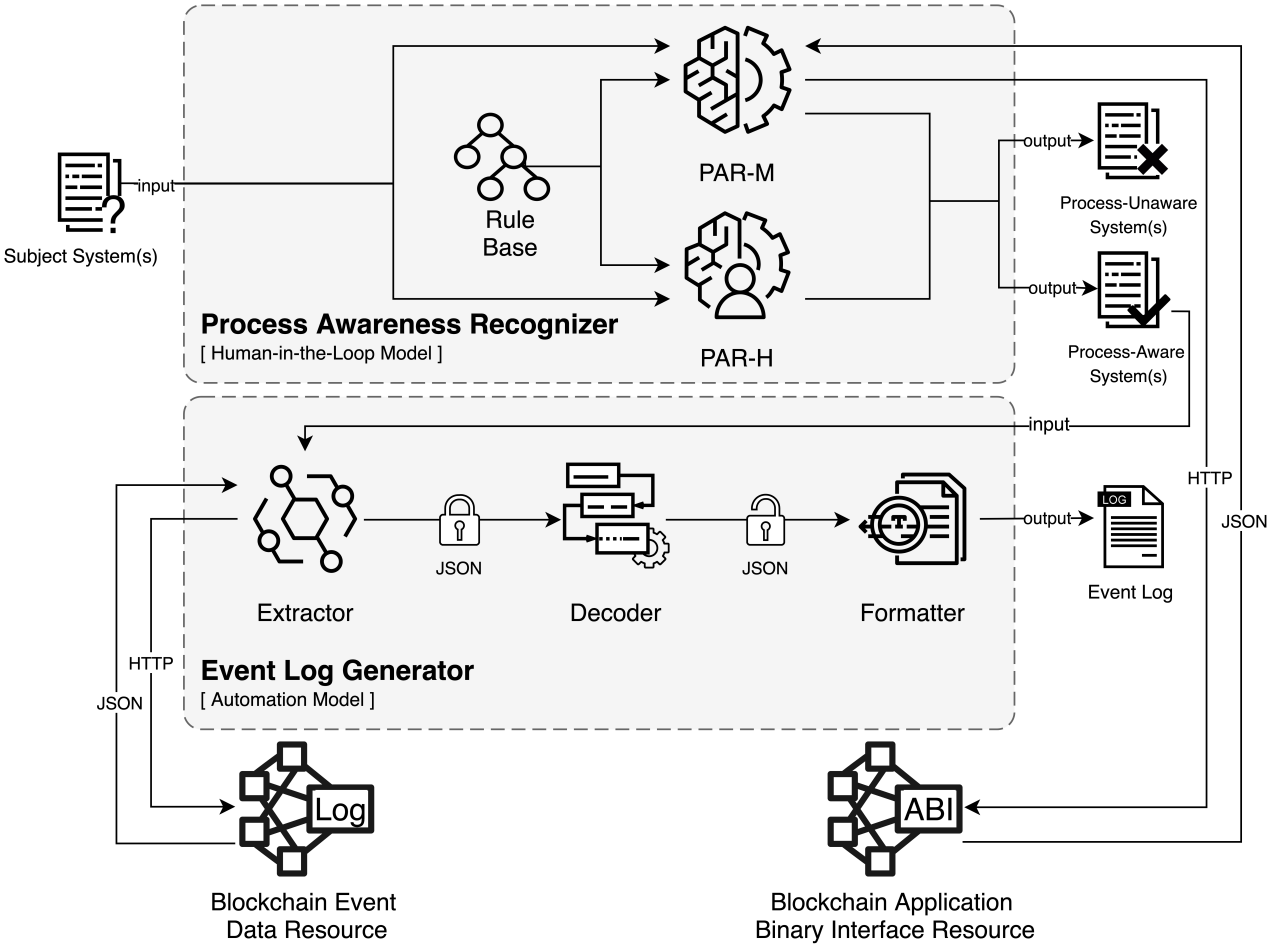

Figure 1: The proposed framework for supporting mining business processes from blockchain applications

4.1.2. Inference engine

This is a binary classification engine which consists of *PAR-H* representing the human expert and *PAR-M* representing the automation algorithm which supports decision-making by the *PAR-H*. The engine follows a forward chaining strategy starting with data provided by the user, selecting rules by pattern-matching and executing the corresponding actions. Due to the hyperdevelopment of blockchain technology, the first, second, and fifth decisions are assigned to the domain expert.

For a given set of subject applications, *PAR-H* starts the process by examining the platform's capability and network accessibility. Smart contracts of the applications that pass the two criteria are then ingested into *PAR-M* to examine the contract verification status and the existence of events. These two decisions require communication with dedicated blockchain networks to get the ABIs of the subject smart contracts. The pseudocode of *PAR-M* is presented in Algorithm 1. It takes as input a manifest contains three attributes for each smart contract: $scID$ represents the contract address, $appName$ represents the name of a blockchain application, and $network$ refers to the blockchain network where the smart contract is deployed. It classifies smart contracts into two classes: $Include$ and $Exclude$. The $Include$ class contains smart contracts that satisfy the inclusion rule. The $Exclude$ class contains smart contracts that violate the inclusion rule for one of the following reasons:

- The smart contract is not verified
- The smart contract does not have events

The output of this algorithm is a manifest that contains, in addition to the input data, classification result, reason for the classification, contract verification status and the event list where applicable. *PAR-H* then takes the role to decide valid applications for PM by examining the events within the included smart contracts. The smart contract addresses of the valid applications can be then ingested to the ELG module to generate their event logs.

4.2. Event log generator

This module is an automated method to generate log files from process-aware blockchain applications' event data. It consists of three algorithms: Extractor, Decoder, and

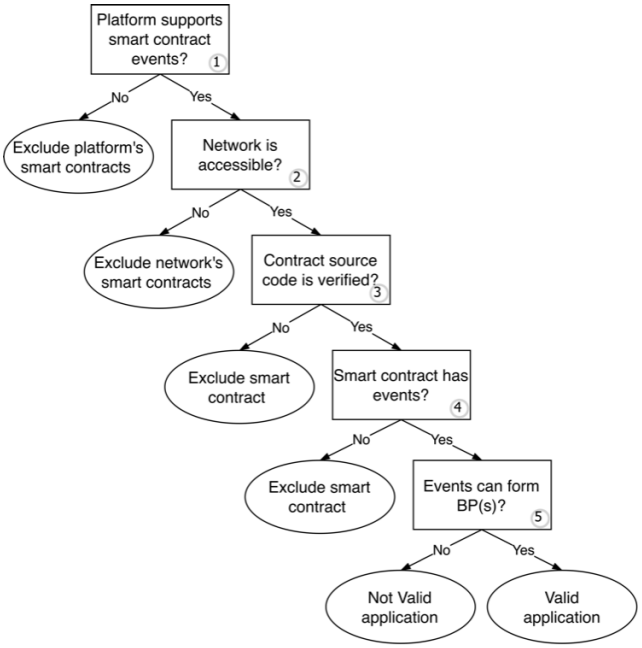

Figure 2: The decision tree to support identifying process-aware EVM-based blockchain applications

Formatter. These algorithms are described in the following subsections.

4.2.1. Extractor

This algorithm retrieves event data from blockchain networks, see Algorithm 2. It relies on end point nodes of the networks. It supports batch retrieval of event logs by dynamically generating RPCs to dedicated blockchain networks for a defined set of contract addresses. By default, it starts extracting data from the first block to the latest. Then, it adjusts these parameters depending on the received response. The maximum number of extracted activity records for a single application is determined by four factors: network's API limit ($Limit_{API}$), number of constituent smart contracts ($CCount_S$), number of events defined within each smart contract ($ECount_C$), and number of RPCs per event ($RPCCount_E$). First, the maximum number of records for a single event is calculated as in Equation 1. Second, the maximum number of records for a single smart contract is calculated as in Equation 2. Finally, the maximum number of records for a single blockchain application is calculated as in Equation 3. For example, assume we have an Ethereum-based application consists of five smart contracts, each contract has four events, we set $RPCCount_E$ to 15; and we use an API with a limit of 1000 records per RPC, then we can get an event log file with a maximum of 300,000 records. For each event, the algorithm executes RPCs as many as $RPCCount_E$. The output of this algorithm is an encrypted log object. It passes the log and the event list vector to Decoder (Algorithm 3).

$$Records_E \leq Limit_{API} \times RPCCount_E \tag{1}$$

Algorithm 1: PAR-M

Input : A finite set of smart contracts $A = \{a_1, \cdots, a_n\}$. $a \in A \mid a$={$appName, scID, network$}

Output: A finite set of smart contracts $B = \{b_1, \cdots, b_n\} \mid |B| = |A|$. $b \in B \mid b$={appName, scID, network, scStatus, events, class, reason } $v \in b.events \mid v.value$ is the event name and $v.index$ is the event signature

```
B ← {};
b ← {};
foreach a in A do
    abiURL ← RPC url for a;
    send RPC Request(abiURL);
    if isSuccessfule(RPC) then
        b ← a;
        abiJSON ← RPC result;
        flag ← false;
        eventList ← {};
        repeat
            if 'type' of current abiJSON item = "event" then
                flag=true
            end if
        until end of abiJSON OR flag=true;
        if flag=true then
            foreach i in abiJSON do
                if i.type="event" then
                    signtr ← encodeEventSignature(i.name);
                    eventList[signtr] ← i.name;
                end if
            end foreach
            b.events ← eventList;
            b.class ← "Include";
            b.reason ← "Contract is verified and has events";
        else
            b.class ← "Exclude";
            b.reason ← "Contract is verified but no defined
              events";
        end if
    else
        b.class ← "Exclude";
        b.reason ← "Contract is not verified";
    end if
    B ← B + b;
end foreach
```

$$Records_C \leq Records_E \times ECount_C \tag{2}$$

$$Records_S = \sum_{n=1}^{CCount_S} Records_{C_n} \tag{3}$$

4.2.2. Decoder

This algorithm intends to transform raw event data, received from Extractor, into a human-readable format. The Decoder is shown in Algorithm 3. It takes as input an encoded log object and a related event list. The log object has the following attributes: $address$, $topics$, $data$, $blockNumber$, $timeStamp$, $gasPrice$, $gasUsed$, $logIndex$, $transactionHash$, $transactionIndex$. Apart from $address$ and $transactionHash$, the data are decoded into appropriate formats. The Unix timestamp is converted to Universal Time Coordinated (UTC) format. The event names

Algorithm 2: Extractor

Input : A finite set $A = \{a_1, \cdots, a_n\}$ of blockchain applications. $a \in A \mid a$={$appName, scID, network, events$}
Output: A finite set $L = \{log_1, \cdots, log_n\}$ of event logs. $l \in L$ such that log={$address, topics, data, blockNumber, timeStamp, gasPrice, gasUsed, logIndex, transactionHash, transactionIndex$}.
Param : $RPCParams$ = {$api, module = logs, action =' getLogs', fromBlock, toBlock, address, topic0, apiKey$}

```
L ← {};
l ← {};
fromBlock ← 1;
toBlock ← latest;
rounds ← integer value;
foreach a in A do
    if |a.events| > 0 then
        l.log ← {};
        foreach event in a.events do
            repeat
                api ← api for a.network;
                apiKey ← api access key for a.network
                  topic0 ← index of event;
                logURL ← fill RPCParams with api +
                  fromBlock + toBlock + a.scID + topic0;
                send RPC Request(logURL);
                if isSuccessfule(RPC) then
                    logJSON ← RPC result;
                    l.log ← l.log + logJSON;
                end if
                fromBlock ← last block number in l.log+1;
                Decoder(l.log, a.events);
            until finish rounds;
        end foreach
        L ← L + l;
    end if
end foreach
```

Algorithm 3: Decoder

Input : A finite set of event logs $L = \{l_1, \cdots, l_n\}$. $l \in L \mid l$={Object log={$address, topics, data, blockNumber, timeStamp, gasPrice, gasUsed, logIndex, transactionHash, transactionIndex$}, Object $events$}
Output: A finite set of decoded event logs $D = \{log_1, \cdots, log_n\}$

```
foreach l in L do
    eventList ← l.events;
    encodedLog ← l.log;
    decodedLog ← [];
    foreach record in encodedLog do
        decodedRecord ← {};
        for i ← 0 to eventList.length-1 do
            if record.topics = eventList[i].key then
                decodedRecord.topics ← eventList[i];
                return
            end if
        end for
        decodedRecord.data ← decodeLogs(record.data);
        decodedRecord.blockNumber ←
          hexToNumberString(record.blockNumber);
        decodedRecord.timestamp ←
          unixToUtc(record.timestamp);
        decodedRecord.gasPrice ←
          hexToNumberString(record.gasPrice);
        decodedRecord.gasUsed ←
          hexToNumberString(record.gasUsed);
        decodedRecord.logIndex ←
          hexToNumberString(record.logIndex);
        decodedRecord.transactionIndex ←
          hexToNumberString(record.transactionIndex);
        decodedLog.push(decodedRecord);
    end foreach
    Formatter(decodedLog);
end foreach
```

are identified by matching $topics[0]$ with the calculated signatures in the event list. The other attributes are converted from Hexadecimal format to String format. The output of this algorithm is a decoded log object passed to Formatter (Algorithm 4).

4.2.3. Formatter

This algorithm generates a local file in a format readable by PM tools. The algorithm is depicted in Algorithm 4. It takes as input the encoded log object resulted from the Decoder. Then, it converts the object into a format that is acceptable by PM tools. The current industry-wide event data standard is eXtensible Event Stream (XES) (IEEE, 2016). However, this traditional format is to be evolved to Object-Centric Event Data (OCED) (van der Aalst and Berti, 2020; Berti and van der Aalst, 2022). According to IEEE Task Force on Process Mining (personal communication, 9 September 2022), the meta model of OCED is under development and its reference implementation is planned to be available in 2023. As many PM tools support Comma Separated Values (CSV) files as an intermediate format, we settle for this format in the meantime. The rows in a CSV file correspond to events and the columns correspond to attributes of the events. The columns should indicate the case identifier, the activity name, and the timestamp of an event, and any other attributes.

5. Proof of concept

To demonstrate the feasibility of the proposed framework, we developed a Blockchain Event Log App (BELA). BELA is a reusable Node.js implementation of the framework. The source code of BELA and the produced event log are publicly available (Alzhrani, 2023).

5.1. Experimental design

The experiment was performed through four steps as follows:

1. *Data collection and pre-processing*: We applied the proposed framework to Alzhrani's dataset (Alzhrani, 2020). The dataset includes applications from nine blockchain platforms: Corda, Hyperledger Fabric, Ethereum, EOS, Klaytn, Steem, Hive, Blockstack, and POA. For the purpose of this study, we only included EVM-compatible applications. The dataset has 282 EVM-compatible applications. Also, since the dataset does not include the smart contract addresses of the applications, we tried to collect them. The addresses for some applications were not publicly provided by their developers, so they were excluded from the

Algorithm 4: Formatter

Input : A finite set of decoded event logs $D = \{log_1, \cdots, log_n\}$ in JSON format
Output: A finite set of formatted event logs $F = \{log_1, \cdots, log_n\}$ in the required format
Param : string *outputFormat*='CSV'
foreach *log in D* **do**
  $formattedLog \leftarrow []$;
  foreach *record in log* **do**
    $formattedRecord \leftarrow \{\}$;
    $formattedRecord.caseID \leftarrow record.transactionHash$;
    $formattedRecord.activity \leftarrow record.topics$;
    $formattedRecord.resource \leftarrow record.address$;
    $formattedRecord.data \leftarrow record.data$;
    $formattedRecord.blockNumber \leftarrow record.blockNumber$;
    $formattedRecord.timestamp \leftarrow record.timestamp$;
    $formattedRecord.gasPrice \leftarrow record.gasPrice$;
    $formattedRecord.gasUsed \leftarrow record.gasUsed$;
    $formattedRecord.logIndex \leftarrow record.logIndex$;
    $formattedRecord.transactionIndex \leftarrow record.transactionIndex$;
    $formattedLog$.**push**($formattedRecord$);
  end foreach
  save($formattedLog$, $outputFormat$);
end foreach

analysis. As a result, 201 applications are used for this study.

2. *Applying PAR*: The input of this step was the dataset of 201 blockchain applications. Based on the predefined *Rule Base*, $PAR-H$ (the researchers) examined the platform and network rules. Then, $PAR-M$ examined the verification status and existence of business-relevant events. The output was a classified set of $Included$ and $Excluded$ smart contracts. Finally, $PAR-H$ examined the event registry of the $Included$ smart contract set.
3. *Applying ELG*: At this step, we selected a sample blockchain application, i.e., CryptoKitties, for investigation. The application was selected due to its popularity in both, industry and academia. The input was a set of four smart contract addresses that belongs to CryptoKitties. The output was an event log file in CSV format.
4. *Applying PM*: In this step, we used the generated event log of Cryptokitties to perform an Automated Business Process Discovery (ABPD) using Apromore[1] tool. The goal of this PM practice is not to give a business analysis for the application. Rather, the goal is to validate that the generated event log by ELG is readable by PM tools and feasible to discover business processes.

5.2. Experimental results

In this section, we first provide statistical analysis of the results. Then, we demonstrate the use of Cryptokitties's event log for ABPD.

[1] https://apromore.com [last accessed 1/5/2023]

5.2.1. Results analysis

The input dataset consists of 201 blockchain applications. They are on four different blockchain platforms: Ethereum, POA, EOS, and Klaytn. Figure 3 summarizes the results of applying PAR to the dataset. When the first criterion was applied, we found that EOS and Klaytn platforms were in older versions that do not support smart contract events. Therefore, 47 applications were excluded. The remaining 154 applications consist of 564 smart contracts. They were investigated for the network accessibility criterion. The applications exist on five networks: Ethereum mainnet, POA Core, Ropsten, Rinkeby, and Kovan. Since the Ropsten, Rinkeby, and Kovan networks are deprecated (Protocol Support Team, 2022), 58 smart contracts were excluded. The remaining 147 applications comprises a total of

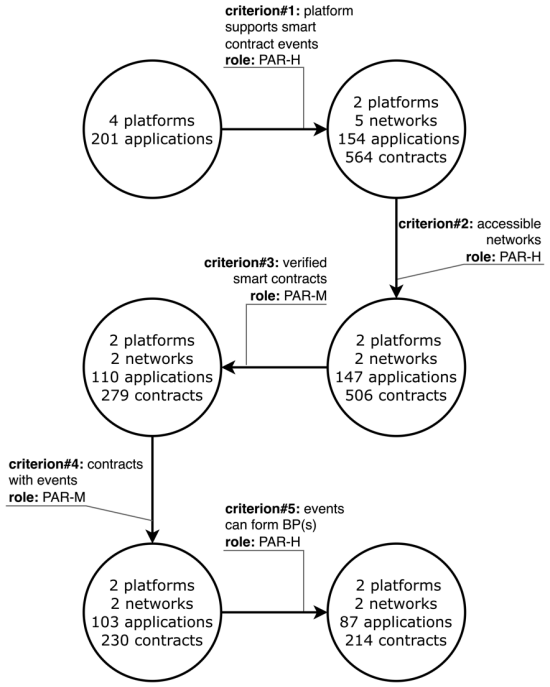

Figure 3: The results of applying Process Awareness Recognizer (PAR) on the dataset

506 smart contracts. At this step, a manifest for these smart contracts was prepared to be ingested into BELA. *PAR-M* executed 506 RPCs to the two networks, i.e., Ethereum Mainnet and POA Core. Of 506 contracts, *PAR-M* found 279 verified contracts of 110 applications. These contracts were further examined for defined events. As a result, 230 contracts of 103 applications were included. Figure 4 shows the classification results of the 506 contract addresses by PAR-M. Then, the included applications were examined by PAR-H for process awareness, which was resulted in 87 valid applications.

Afterward, we applied ELG on a sample application. CryptoKitties has four out of four verified smart contracts with events. Three of the four contracts have an equal number of 5 events. The fourth contract has 12 defined events. The Extractor (Algorithm 2) was configured with

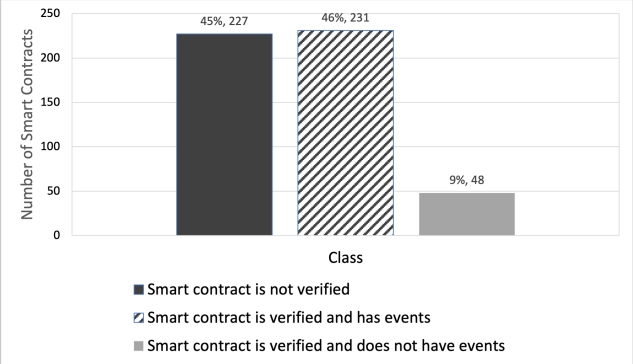

Figure 4: Classification results produced by PAR-M for 506 smart contract addresses

$RPCCount \leq 15$; for efficiency and performance. That is, if no more event data is found for an event in the current RPC, then stop the remaining RPCs for this event and proceed to the next event. We used Etherscan API[2] which has a limit of 1000 records/rpc and 5 rpcs/second. Therefore, the number of extracted log entries for CryptoKitties should be $Records_{CK} \leq 1000 \times 15 \times 27 = 405,000$.

We prepared a manifest containing the required attributes in JSON format for the four smart contracts. The manifest was ingested into BELA to be processed by the ELG module. The ELG successfully executed 202 RPCs to the Ethereum Mainnet, decoded and formatted 185 JSON responses of event data. That is, 17 RPCs did not find more event data. As a result, a CSV event log file was successfully generated for CryptoKitties, with an actual number of log entries of 177,201.

5.2.2. Business process discovery

To validate the outcome of ELG by BELA, we used the generated event log of CryptoKitties with the Apromore tool. We are interested to gain an overview of the lifecycle process of the kitties (i.e., non-fungible tokens), where the lifecycle of each kitty is viewed as an independent process instance. Therefore, the identifier of a kitty (i.e., tokenId) is used as the $caseID$. The log timeframe is from 2017-11-23 to 2022-11-06, which is from block number 4605167 to block number 15907923. Figure 5 shows the discovered business process model for CryptoKitties. It shows the dynamic breeding and trading of the kitties and gives insightful information for the game developers about the behavior of the game users. It can be obsevered that the discovered model in this study is consistent with, and more detailed than, the one that was discovered in Klinkmüller et al. (2019). This indicates the validity of our discovered model, and hence, the validity of our proposed framework.

6. Discussion

This study addresses the aforementioned challenges of applying PM in the context of EVM-compatible blockchain applications. Process awareness of a blockchain application is impacted by multiple components. Based on a taxonomy proposed by Alzhrani et al. (2022) to characterize blockchain applications, we can characterize a process-aware EVM-based blockchain application as: A system that its execution environment supports programmable smart contracts and executed them on a Turing complete machine, and the system's internal components include smart contracts that are programmed with business-relevant events. Figure 6 shows the relationship between different entities related to the process awareness of a blockchain application.

Our proposed ELG contrasts the literature in several points. First, our proposed method focuses on event data retrieval and decoding steps instead of the format of the event log. Second, once the manifest is ready, it is a fully automated method to generate event logs, with support for batch smart contract addresses. Third, our proposed method is independent of the developers of smart contracts and the manifest is prepared with minimal information: *(appName, scID, network)* for PAR and *(appName, scID, network, eventList)* for ELG, and no prior knowledge of a contract's source code or defined events is required to build the manifest. Last, ELG relies on remote nodes eliminating the complexity of deploying local nodes and thus, no blockchain-specific technical expertise is required.

Since the ELG relies on RPCs, the method is impacted by the limit of API providers. As more $RPCCount_E$ per event as more event data are retrieved and thus, more possible variants of a process can be discovered. However, there should be a trade-off between the $RPCCount_E$ and the performance for the scalability of BELA. Regarding the event log formatting in this study, the content of the $data$ field is determined by the programmer of the smart contract. Thus, different data are stored for different activities, which complicate defining a unified set of attributes in the CSV format.

While it is possible to provide a deep analysis of CryptoKitties, the purpose of ABPD in this study is to validate the applicability and feasibility of the proposed framework. We successfully identified valid blockchain applications for PM, extracted event data, decoded them into a human-readable format, stored them in the CSV format, and discovered the relevant process models from the generated event log. To generate different analysis, our framework allows business analysts to adapt a user-guided approach for PM, where the analyst can try different attribute configurations during the PM session to find the best model for their goals Burattin (2015).

7. Conclusion

This study introduced the problem of process awareness of blockchain applications. It outlined several challenges for mining business processes from these applications. Also, we proposed a framework to address these challenges in the context of EVM-based blockchain applications. The framework consists of two modules. The first is PAR, a

[2] https://etherscan.io/apis

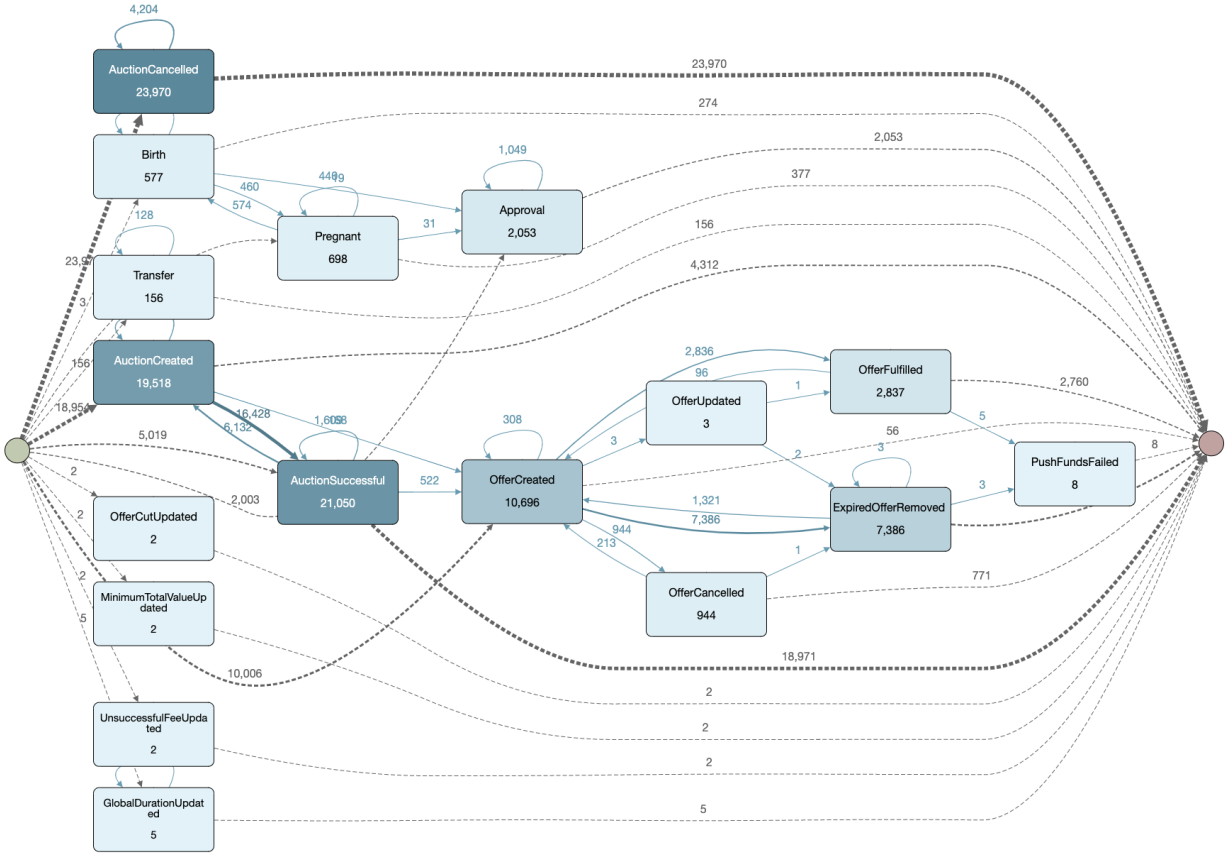

Figure 5: A discovered business process model for CryptoKitties using Apromore

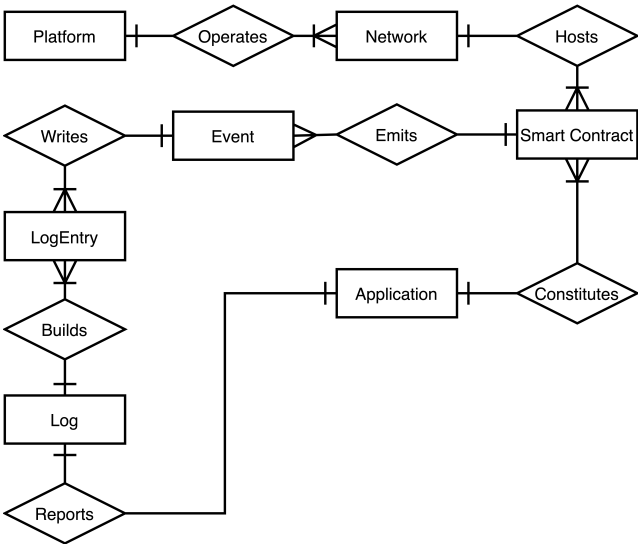

Figure 6: The relationship between entities related to process awareness of a blockchain application

semi-automated model to support the decision-making about the process awareness of a given blockchain application. The second is ELG, an automated model to support event log generation activities. Additionally, a Proof-of-Concept application (BELA) was implemented as a reusable Node.js app. It was used to validate the proposed framework on a set of real-world blockchain applications. The results approved the feasibility and applicability of the framework. This work is an initial proposal and involves human decisions, which may introduce subjectivity. In the future, we aim to improve PAR-M to detect business-relevant events from smart contract source codes. Additional future work is to evaluate the performance of the algorithms with different input sizes. Also, we work to generalize the framework for non-EVM-based blockchain applications.

Declaration of competing interest

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

Data availability

Data used for this research are available in Zenodo (Alzhrani, 2020).

Acknowledgements

Fouzia Alzhrani wishes to gratefully acknowledge a scholarship she received from King Abdulaziz University that

enables her to pursue a Ph.D. degree at the University of Manchester.

References

J. Mendling, I. Weber, W. V. D. Aalst, J. V. Brocke, C. Cabanillas, F. Daniel, S. Debois, C. D. Ciccio, M. Dumas, S. Dustdar, Blockchains for business process management-challenges and opportunities, ACM Transactions on Management Information Systems (TMIS) 9 (2018) 4. doi:10.1145/3183367.

J. S. Arun, J. Cuomo, N. Gaur, Blockchain for Business, Addison-Wesley Professional, 2019.

V. A. d. Sousa, B. Corentin, Towards an integrated methodology for the development of blockchain-based solutions supporting cross-organizational processes, in: 2019 13th International Conference on Research Challenges in Information Science (RCIS), 2019, pp. 1–6. doi:10.1109/RCIS.2019.8877045.

W. Viriyasitavat, D. Hoonsopon, Blockchain characteristics and consensus in modern business processes, Journal of Industrial Information Integration 13 (2019) 32–39. doi:10.1016/j.jii.2018.07.004.

A. Burattin, Process Mining, Springer International Publishing, Cham, 2015, pp. 33–47. doi:10.1007/978-3-319-17482-2_5.

W. v. d. Aalst, Process mining: Overview and opportunities, ACM Trans. Manage. Inf. Syst. 3 (2012) Article 7. doi:10.1145/2229156.2229157.

L. Reinkemeyer, Process Mining in a Nutshell, Springer International Publishing, Cham, 2020, pp. 3–10. doi:10.1007/978-3-030-40172-6_1.

W. van der Aalst, Data Science in Action, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 3–23. doi:10.1007/978-3-662-49851-4_1.

C. dos Santos Garcia, A. Meincheim, E. R. Faria Junior, M. R. Dallagassa, D. M. V. Sato, D. R. Carvalho, E. A. P. Santos, E. E. Scalabrin, Process mining techniques and applications – a systematic mapping study, Expert Systems with Applications 133 (2019) 260–295. doi:10.1016/j.eswa.2019.05.003.

W. van der Aalst, A. Adriansyah, A. K. A. de Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. van den Brand, R. Brandtjen, J. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. de Leoni, P. Delias, B. F. van Dongen, M. Dumas, S. Dustdar, D. Fahland, D. R. Ferreira, W. Gaaloul, F. van Geffen, S. Goel, C. Günther, A. Guzzo, P. Harmon, A. ter Hofstede, J. Hoogland, J. E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. La Rosa, F. Maggi, D. Malerba, R. S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H. R. Motahari-Nezhad, M. zur Muehlen, J. Munoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H. Seguel Pérez, R. Seguel Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard, M. Wynn, Process mining manifesto, in: F. Daniel, K. Barkaoui, S. Dustdar (Eds.), Business Process Management Workshops, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 169–194. doi:10.1007/978-3-642-28108-2_19.

W. van der Aalst, Getting the Data, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 125–162. doi:10.1007/978-3-662-49851-4_5.

L. Reinkemeyer, Processtraces: Technology, Springer International Publishing, Cham, 2020, pp. 31–35. doi:10.1007/978-3-030-40172-6_5.

W. van der Aalst, Process Mining: The Missing Link, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 25–52. doi:10.1007/978-3-662-49851-4_2.

N. Y. Wirawan, B. N. Yahya, H. Bae, Incorporating Transaction Lifecycle Information in Blockchain Process Discovery, Springer Singapore, Singapore, 2021, pp. 155–172. doi:10.1007/978-981-33-4122-7_8.

R. Mühlberger, S. Bachhofner, C. Di Ciccio, L. García-Bañuelos, O. López-Pintado, Extracting event logs for process mining from data stored on the blockchain, in: International Conference on Business Process Management, Business Process Management Workshops, Springer International Publishing, 2019, pp. 690–703. doi:10.1007/978-3-030-37453-2_55.

F. Corradini, F. Marcantoni, A. Morichetta, A. Polini, B. Re, M. Sampaolo, Enabling Auditing of Smart Contracts Through Process Mining, Springer International Publishing, Cham, 2019, pp. 467–480. doi:10.1007/978-3-030-30985-5_27.

V. Ivković, I. Luković, An approach to validation of business-oriented smart contracts based on process mining, in: European Conference on Advances in Databases and Information Systems, New Trends in Database and Information Systems, Springer International Publishing, 2021, pp. 303–309. doi:10.1007/978-3-030-85082-1_27.

C. Di Ciccio, G. Meroni, P. Plebani, Business process monitoring on blockchains: Potentials and challenges, in: International Conference on Business Process Modeling, Development and Support, International Conference on Evaluation and Modeling Methods for Systems Analysis and Development, Enterprise, Business-Process and Information Systems Modeling, Springer International Publishing, 2020, pp. 36–51. doi:10.1007/978-3-030-49418-6_3.

M. Müller, P. Ruppel, Process mining for decentralized applications, in: 2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON), 2019, pp. 164–169. doi:10.1109/DAPPCON.2019.00031.

R. Hobeck, C. Klinkmüller, H. M. N. D. Bandara, I. Weber, W. M. P. van der Aalst, Process mining on blockchain data: A case study of augur, in: International Conference on Business Process Management, Business Process Management, Springer International Publishing, 2021, pp. 306–323. doi:10.1007/978-3-030-85469-0_20.

L. M. M'Baba, N. Assy, M. Sellami, W. Gaaloul, M. F. Nanne, Extracting artifact-centric event logs from blockchain applications, in: 2022 IEEE International Conference on Services Computing (SCC), 2022, pp. 274–283. doi:10.1109/SCC55611.2022.00048.

I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, J. Mendling, Untrusted business process monitoring and execution using blockchain, in: M. La Rosa, P. Loos, O. Pastor (Eds.), Business Process Management, Springer International Publishing, Cham, 2016, pp. 329–347. doi:10.1007/978-3-319-45348-4_19.

C. Klinkmüller, A. Ponomarev, A. B. Tran, I. Weber, W. van der Aalst, Mining blockchain processes: Extracting process mining data from blockchain applications, in: C. Di Ciccio, R. Gabryelczyk, L. García-Bañuelos, T. Hernaus, R. Hull, M. Indihar Štemberger, A. Kő, M. Staples (Eds.), Business Process Management: Blockchain and Central and Eastern Europe Forum, Springer International Publishing, Cham, 2019, pp. 71–86. doi:10.1007/978-3-030-30429-4_6.

Hyperledger Besu community, Configure logging - hyperledger besu, https://besu.hyperledger.org/en/stable/public-networks/how-to/monitor/logging/, 2022.

The go-ethereum Authors, Geth logs, https://geth.ethereum.org/docs/interface/logs, 2022.

G. Wood, Ethereum: A secure decentralised generalised transaction ledger, https://ethereum.github.io/yellowpaper/paper.pdf, 2022.

IEEE, IEEE standard for extensible event stream (XES) for achieving interoperability in event logs and event streams, IEEE Std 1849-2016 (2016) 1–50. doi:10.1109/IEEESTD.2016.7740858.

W. M. van der Aalst, A. Berti, Discovering object-centric petri nets, Fundam. Informaticae 175 (2020) 1–40. doi:10.48550/arXiv.2010.02047.

A. Berti, W. M. P. van der Aalst, OC-PM: analyzing object-centric event logs and process models, International Journal on Software Tools for Technology Transfer (2022). doi:10.1007/s10009-022-00668-w.

F. Alzhrani, BELA: Blockchain Event Log App, https://github.com/alzhraniFouz/BELA, 2023. doi:10.5281/zenodo.7620035.

[Dataset] F. Alzhrani, 2020, A catalogue of blockchain-based applications, doi:10.5281/zenodo.4268953.

Protocol Support Team, Ropsten, Rinkeby & Kiln deprecation announcement, https://geth.ethereum.org/docs/interface/logs, 2022.

F. E. Alzhrani, K. A. Saeedi, L. Zhao, A taxonomy for characterizing blockchain systems, IEEE Access 10 (2022) 110568–110589. doi:10.1109/ACCESS.2022.3214837.

A. Burattin, User-Guided Discovery of Process Models, Springer International Publishing, Cham, 2015, pp. 113–118. doi:10.1007/978-3-319-17482-2_13.

Chapter 6

A Business Process Modeling Pattern Language for Blockchain Application Requirement Analysis

Research Objective

Objective 5: To identify reusable business process patterns for blockchain application development

Thesis Context

This chapter uses the event logs dataset from Chapter 2. This chapter is based on a paper that has been submitted to Requirements Engineering. The full reference of this paper is as follows:

F. Alzhrani, K. Saeedi, and L. Zhao, "A Business Process Modeling Pattern Language for Blockchain Application Requirement Analysis," *Requirements Engineering*, Under Review.

CRediT authorship contribution statement

Fouzia Alzhrani: Conceptualization, Data Curation, Methodology, Writing - Original Draft, Writing - Review & Editing. **Kawther Saeedi**: Conceptualization, Supervision, Writing - Review & Editing. **Liping Zhao**: Conceptualization, Supervision, Writing - Review & Editing.

A Business Process Modeling Pattern Language for Blockchain Application Requirement Analysis

Abstract

Blockchain technology has the potential to revolutionize e-business through smart contracts. However, developing blockchain applications is obstructed by several usability challenges from a software engineering perspective. Defining proper smart contracts and solution architecture requires a tighter connection between the analysis and design phases. Current research on software development for blockchain applications focuses mainly on software design, rarely on requirement analysis or advance to implementation overlooking the requirements. Literature and industry practices on blockchain development justified the significant lack of a common language to communicate requirements to the software design. This study aims to address this research gap. It proposes a data-driven business process modeling pattern language for blockchain application requirements analysis. The pattern language consists of nine business process patterns. This study applies Process Mining (PM) techniques in the context of software engineering. The patterns are identified using Automated Business Process Discovery (ABPD) and validated using Conformance Checking (CC) techniques. The contribution of this study provides a common language that connects requirements to software design to support the development of blockchain applications.

Keywords: Blockchain, smart contract, design pattern, software design, requirements analysis, business process modelling

1 Introduction

Blockchain technology has the potential to revolutionize e-business by offering the ability of a peer-to-peer business management system which eliminates the third party and ensures the integrity of the processes and data [1]. Business processes can be encoded within smart contracts (SCs), allowing for confidential execution and monitoring of inter- and intra- organizational business processes [2, 3]. However, developing blockchain applications (BCApps) is obstructed by several usability challenges from a software engineering perspective [4].

Industry practices in blockchain development evident the difficulty of communicating stakeholders' requirements in the design [5], leaving requirement formulation and specification a technical task [6]. These difficulties are further exacerbated by the significant lack of requirement analysis from a business process perspective, making it challenging to identify which requirements, if they are addressed by SCs, constitute value to stakeholders. Current research on software development for BCApps focuses mainly on software design, rarely on requirement analysis, or advances to implementation overlooking the requirements [7–9]. Defining proper SCs and solution architecture requires a tighter connection between analysis and design phases [8]. Business Process Modeling (BPM) is profound in improving requirement analysis [10].

This study continues the effort to address usability issues that affect BCApp development. The difference is that it targets the connection between analysis and design phases not addressed in the literature. This study aims to systematically identify business processes embedded within SCs and document them in terms of reusable BPM Patterns (BPMPs). Towards this, it is essential to discover business processes from Industry-developed BCApps. Manual process discovery from a large number of applications and systems is subjective and requires a significant amount of

time and effort [11]. Figuring out possible methods for a single application requires multiple sessions to use the application, read its documentation, interview its developers, and analyze user reviews if available.

Away from the perception-based discussion, Process Mining (PM) [12] allows for a fact-based analysis through the processing of the system's event logs by state-of-the-art algorithms, techniques, and tools [13–15]. PM is a unique approach that applies data mining to automatically extract knowledge from captured process traces in the system's log files [13–17]. Experience shows that less time is spent analyzing business processes using the results of PM [15]. Therefore, PM arguably allows for evidence-based pattern identification. Instead of intuitively proposing patterns based on experience and observations, then validating their existence through searching available use cases, PM allows for a data-driven pattern identification that ensures the validity of the identified patterns beforehand.

As a result, the main contribution of this study is a data-driven BPMPs to assist requirements analysis of BCApps. This study contributes to software engineering literature by providing a common language that connects analysis and design phases to support the development of BCApps. The remainder of this paper is organized as follows. Section 2 presents a background of BCApps, BPMPs, and PM. Section 3 reviews related work to software patterns for BCApps. Section 4 elucidates the research methodology. Section 5 gives an overview of the proposed BPMPL. Sections 6 and 7 describe the BPMPs. Section 8 represents the validation results of the BPMPL. Section 9 discusses the implications and limitations of the BPMPL. Finally, Section 10 summarizes the paper.

2 Background

2.1 Blockchain applications

A BCApp is a software application that utilizes blockchain technology. The application uses web services to communicate the off-chain tiers with the blockchain [18]. A BCApp is intended to provide functionality according to use-case needs and requirements. The business logic of a BCApp can be implemented in three general ways: 1) it can be implemented solely off-chain and uses the blockchain for data storage, 2) it can be implemented solely on-chain through SCs, or 3) it can be distributed between the off-chain app and SCs. Generally, a BCApp has the following characteristics [19]:

- It operates on one or more blockchain networks. For example, IBM Food Trust is built on Hyperledger Fabric and IBM Blockchain.
- It uses one or more distributed ledgers. This is especially true for applications built on multi-ledger architecture platforms, such as Hyperledger Fabric, through the concept of channels.
- It has one or more SCs. For example, the CryptoKitties Ethereum application has five SCs.
- It consists of off-chain and on-chain components.
- Its data and computation can be on a single chain, multiple chains (sidechain), or off-chain. This characteristic has a significant impact on the scalability and performance of the application.
- Its data are immutable, in contrast to conventional applications. This feature is inherited from the immutability of the blockchain.

2.2 Business process modeling patterns

A business process (BP) is a goal-centered arrangement of value-added activities and resources. It utilizes certain inputs to create desired outputs when the activities are executed by their relevant originators [20, 21]. A BP pattern is a reusable business process model that represents the relationship between users, the application, and the data [11, 22, 23]. BPMPs are generalized process models of how a business should be run in a given context [24]. According to Sommerville [25], patterns and pattern languages capture experience by describing best practices and good designs in a reusable way. The pattern describes the recurring problem, identifies its solution, and explains the importance of the solution. A pattern could be built as a combination of smaller patterns and their relationships [26].

The goal of a pattern is to assist developers in resolving recurring problems throughout the development lifecycle by creating a body of literature for recurring problems [27]. Among the

different types of reusable assets in software engineering, the most used one is software patterns. They are software abstractions that, if properly documented and well applied, can aid in building high-quality software while reducing the development time and cost [28, 29]. As new technologies emerge, patterns should be identified, adapted, and used to build real applications [30].

2.3 Process mining

PM is a growing research field, and it has been proven through several use cases [14]. PM arose as a result of merging techniques from Data Mining and BPM to enable evidence-based BPM [13, 14, 16]. PM is defined as: "techniques, tools, and methods to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs commonly available in today's (information) systems" [17]. There are three types of PM: process discovery, conformance checking, and process enhancement. Process discovery aims to construct a process model based on an event log. Thus, it takes an event log as an input, learns a process model as throughput, and produces an as-is business process model as an output [17].

Automated Business Process Discovery (ABPD) is facilitated using process miners [31]. A process miner is an algorithm responsible for producing a comprehensible process model based on process traces in logs. Conformance Checking (CC) analyzes whether a log conforms to a model and vice versa. Thus, it takes an event log and a reference BP model as input, compares the log and model to each other as throughput, and produces diagnostics as an output. Process Enhancement (PE) aims at extending or improving a BP model based on extracted information from a log. Thus, it takes an event log and a BP model as input, analyzes the log to improve the model as throughput, and produces a new BP model as an output.

3 Related work

Numerous studies have been conducted on software patterns for BCApps. However, most effort has been devoted to design patterns. Table 1 summarizes the list of patterns proposed in the reviewed literature [32–40]. In the context of blockchain-based payments, a set of 12 patterns is presented in [32]. In the context of blockchain-based supply chain finance, [33] proposed five design patterns to solve financing difficulties and costs of enterprises through blockchain. The study in [34] identified foundational aspects for characterizing oracle patterns introduced in the literature based on data flow direction and initiator. Subsequently, four patterns were derived through a literature review. The study in [35] proposed a pattern for executing legal contract clauses by binding an off-chain legal contract, an on-chain SC, and a business process engine to ensure companies' pseudonymity and business data confidentiality.

A list of 12 design patterns for Ethereum SCs was proposed in [36]. The pattern collection in [37] focuses on the security aspects of SCs, which consists of six security patterns to assure the reliability of SC execution. Another work is a collection of 15 design patterns [38], which viewed the blockchain as a fundamental element of large-scale decentralized systems. Solidity-based SC patterns are also discussed in [39] and nine patterns were identified. The study in [40] proposed six trust patterns for collaborative business processes derived from literature.

This study contrasts the literature in several aspects. First, the literature focuses on software design, while this study focuses on requirement analysis. Second, the literature identified patterns intuitively by observation and experience, while this study follows a systematic data-driven approach through the application of ABPD and CC for pattern identification and validation. Third, the literature embodies a technology-centered viewpoint [32–39], whereas this study represents a business process-centered viewpoint. Fourth, the literature is either aimed at process engineers [40] or software engineers [32, 37–39], whereas this study intends to support both business and software engineers.

4 Methodology

This study analyzed event logs retrieved from Ethereum and POA blockchains. These event logs dataset is publicly available [41]. The dataset comprises 116 event log files produced by 101 BCApps from five networks. An overview of the dataset is presented in Table 2. This study used the IBM

Table 1 A list of proposed design patterns in the literature.

Reference No.	Application Context	List of Patterns
[32]	Payments	Token template, Token registry, Policy contract, Burned token, Escrow, Payment channel, Seller credential, Stealth address, Oracle, Multisignature, Token swap, Authorised spender
[33]	Supply chain finance	Pledge template, Pledge register, Ownership challenge, Two-way payment channel, Transfer
[34]	Data exchange	Pull-based inbound oracle, Push-based inbound oracle, Pull-based outbound oracle, Push-based outbound oracle
[35]	Legal contracts	On/off-chain smart-contract binding for confidential contract enforcement
[36]	General	Pull payment, State machine, Commit and reveal, Oracle, Ownership, Access restriction, Mortal, Automatic deprecation, Data segregation, Satellite, Contract register, Contract relay
[37]	SC security	Checks-effects-interaction, Emergency stop, Speed bump, Rate limit, Mutex, Balance limit
[38]	General	Oracle, Reverse oracle, Legal and smart contract pair, Encrypting on-chain data, Tokenisation, Off-chain data storage, State channel, Multiple authorization, Off-chain secret, Xconfirmation, Contract registry, Data contract, Embedded permission, Factory contract, Incentive execution
[39]	General	Token, Authorization, Oracle, Randomness, Poll, Time constraint, Termination, Math, Fork check
[40]	Collaborative BPs	Hash storage, Transparent event log, Blockchain BP engine, Smart contract activities, Blockchain-based reputation system, Decentralization

Process Mining tool to apply two of the PM techniques: ABPD and CC. Fig. 1 illustrates the pattern mining process in IDEF0 [42]. The steps are as follows:

1. *Preprocess dataset:* The dataset is randomly split into analysis and validation datasets. The analysis dataset consists of 58 event log files produced by 51 BCApps. The validation dataset consists of 58 event log files produced by 50 BCApps. Fig. 2 shows the distribution of the applications per domain within each dataset. The SCs are developed in compliance with token standards of Ethereum, mostly ERC20 and ERC271. However, as different programmers developed those SCs, events of similar activities are named differently. For example, a token burn event is named Burn, Burned, Burnt, Tokens Burned, Crystal Burned, and Kitty Burned, while they are all triggered when a token is transferred from the token's owner balance to a zero address. Thus, unifying event names of the same activity is fundamental to ensure a valid conformance analysis. First, all event names have been extracted from the dataset to create a pool of events. Then, a set of potential aliases are identified from the pool. Finally, each alias is validated by analyzing its role in the relevant SC source code to ensure that the alias is triggered by a similar business activity of the aliased event. The event aliases are listed in Appendix A.

Table 2 The distribution of event logs per blockchain network.

Blockchain Network	Number of Event Log Files
Ethereum Mainnet	101
Kovan	4
POA Core	1
Rinkeby	2
Ropsten	8

2. *Discover business processes:* The ABPD was applied to the analysis dataset to identify recurrent business processes. Process discovery algorithms require an event log to have at least three properties: case ID, activity, and timestamp. The mapping from the event logs to process trace data is as follows: transaction hash is used as the case ID, log timestamp is used as the process timestamp, and topics is used as the activity. Examples of such topics include Transfer, Approval, Deposit, Withdraw.

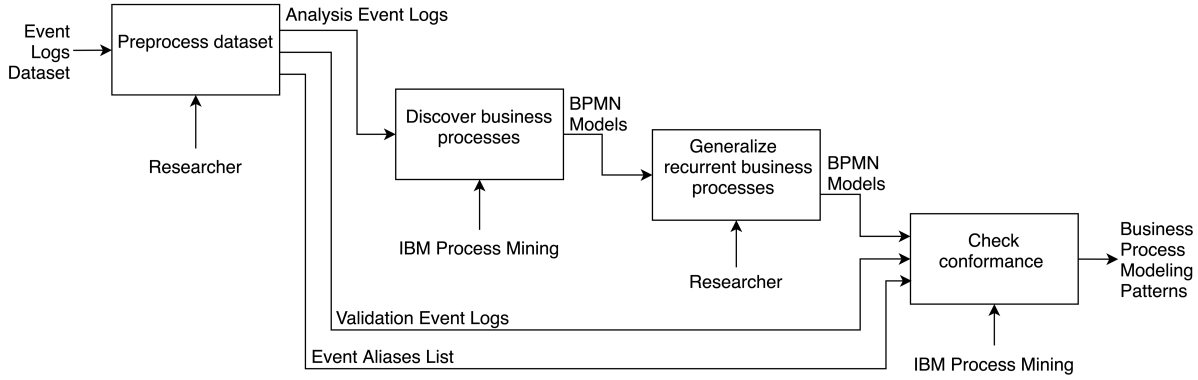

Fig. 1 The pattern mining process

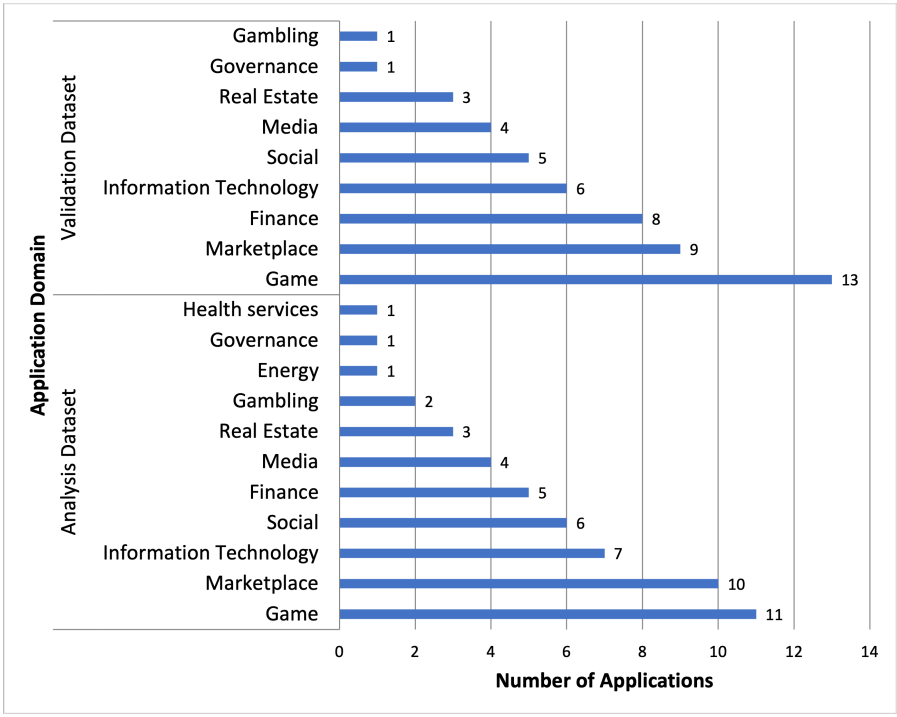

Fig. 2 The distribution of blockchain applications per application domain

3. *Generalize recurrent business processes:* After discovering BPs in the preceding step, recurrent business processes were generalized in Business Process Management Notation (BPMN) models. These models became the reference models for CC.
4. *Check conformance:* Later, the reference models, event aliases list, and the validation event log files were ingested into the PM tool for CC. The IBM Process Mining tool visualizes the conformance between two business models using similarity and fitness indicators. The similarity indicates how the data-derived model compares with the reference model. The fitness indicates how the data-derived model is representative of the cases, depending on the model details. The values are normalized from 0 (bad) to 1 (good), where values from 0.5 to 1 are satisfactory. For this study, the similarity indicator plays an essential role. It indicates to what extent the data-derived model from a BCApp conforms to the generalized BPMN model of the pattern under investigation. Thus, any value from 0.5 to 1 is accepted as an indicator of the pattern's existence.

5 The proposed business process modeling pattern language

The proposed BPM pattern language (BPMPL) in this study consists of nine patterns classified into two broad categories from a business process view (BPV) perspective. The first category is *Token-Oriented Patterns (TOPs)*. This category represents on-chain transactions that can be performed on a token. It consists of three subcategories: *Token Circulation*, *Token Supply*, and *Token Authorization*. The Token Circulation consists of three patterns describing the flow of digital assets from account to account. The Token Supply consists of two patterns describing the production and destruction of digital assets on the blockchain. The Token Authorization consists of a single pattern describing authority on a token. The second category is *Smart Contract-Oriented Patterns (SCOPs)*. This category represents on-chain transactions that can be performed on an SC. It consists of two subcategories: *Smart Contract Security* and *Smart Contract Authorization*. The Smart Contract Security consists of a single pattern describing administrative tasks on an SC. The Smart Contract Authorization consists of two patterns describing access controls to govern access to SC business logic. Table 3 summarizes the BPMPL and interrelationship between the patterns.

5.1 Pattern description

The collective description of the individual patterns constitutes a description of the BPMPL. The pattern descriptions follow the guidelines of requirement and design pattern languages [43–45]. Each BPMP description encapsulates a BP model, requirement, and design pattern. Each BPMP is described using a standard format as follows:

- The pattern's *Name* reflects the principle behind the pattern.
- *Summary* gives an overview of the pattern's usage.
- *Classification* is a BPV-agnostic categorization that indicates the type of the pattern as Transfer, Lifecycle, or Accessibility.
- *Context* describes the situation where the pattern can be applied from a design perspective.
- *Applicability* describes the situation(s) where the pattern can or cannot be applied from a requirement perspective.
- *Problem* expresses what the pattern intends to solve.
- *Solution* describes how the problem is resolved.
- *Business Process Meta-Model* describes the flow of activities and events of the solution in a logical business sequence using an Event-driven Process Chain (EPC) [46]. A reference model of EPC notation is presented in Appendix A.
- *Requirement Template* represents the most abstract expression of an FR that reflect the pattern.
- *Extended Requirement Template* represents other ways to express the requirement template, with possible alternatives or derivative expressions. Each extended requirement template is a typical fill-in-the-blanks definition for an FR of the business process, where <and> encapsulate a mandatory parameter, e.g., <mandatory parameter>, and [and] encapsulates an optional parameter, e.g., [optional parameter]. The FR list is sequentially numbered as FR1, FR2, ... etc.
- *Implementation Considerations* point out fundamental guidelines on how to implement the pattern.
- *Testing Considerations* provide overall guidelines for testing the pattern's business process.
- A running *Example* is used throughout the BPMPL to showcase the application of each pattern on a single business use case, see Section 5.2.
- *Related Patterns* are complementary patterns from the same language. If none, this item is omitted.
- *Known Uses* are BCApps from the validation dataset, introduced in Appendix A, that conform to the pattern's reference model.

5.2 Running example

The running example is a simple use case with basic activities intended to demonstrate the use of the proposed BPMPL to design a new BCApp. This section introduces the application example and identifies the main components and actors. The utilization of each pattern is explained within the relevant description in Sections 6 and 7. The

Table 3 The business process modeling patterns and their interrelationships.

No.	Pattern Type	Category	Main Pattern	Related Pattern	Relationship
P1	Token-Oriented	Token Circulation	Atomic Tx	P2, P3	Alternative
P2			Micro Tx	P6	Requisite
P3			Macro Tx	P1, P2	Alternative
P4		Token Supply	Bloom	P2	Variant
P5			Digester	P1, P2	Variant
P6		Token Authorization	Approval	P2	Prerequisite
P7	Smart Contract-Oriented	Smart Contract Security	Switch	P8, P9	Requisite
P8		Smart Contract Authorization	Exclusive Authorization	P9	Alternative
P9			Shared Authorization	P8	Alternative

sample use case concerns developing an Ethereum-based application to manage entrance passes to a paid event. The application is a reusable solution for different events. Each event has a different manager.

The application allows for controlling the pass issuance process as required, such as starting the process when the registration is open and stopping pass issuance when the registration period ends or the event is canceled. An authorized registrar only should be able to issue passes during the registration period. The application handles the payment of registration fees through a legitimate third-party account. The fee should be paid only when a pass is successfully issued and sent to the requesting invitee. Once a pass is issued for an invitee, the invitee can authorize a representative to attend the event on behalf of the invitee.

A pass is a one-time ticket that must be ineffective after checking in, whether by the invitee or their representative. The application uses two types of tokens: Ether and PASS. Ether is a fungible token and the native cryptocurrency in Ethereum that is used to pay registration fees by event invitees. PASS is a non-fungible token managed by its corresponding token SC, i.e., Pass SC (PassSC), that complies with the ERC270 standard. PASS token is used to allow invitees to check in to the event.

The app has the following users: Manager represents the event manager who is responsible for administering the PassSC and authorizing other users as required, Registrar represents an SC which acts as the event registration officer, Accountant represents an SC wallet which acts as the company's financial officer, Invitee represents an end user who can register to attend the event, Representative represents end user who is authorized to attend on behalf of Invitee, Receptionist represents front desk staff during event check-in, and Escrow represents an SC who handle the payment process and manage funds between Invitee and Accountant.

6 Token-oriented patterns

TOPs comprise a collection of patterns from the perspective of a token. The process of moving a token between accounts on the blockchain can be designed using the Atomic Tx, Micro Tx, and Macro Tx patterns. These Token Transfer patterns are depicted in Section 6.1. The process of creating or destroying a token can be designed using the Bloom and Digester patterns, respectively. These Token Supply patterns are depicted in Section 6.2. The process of authorizing non-owner accounts to manage a token can be designed using the Approval pattern. This Token Authorization pattern is depicted in Section 6.3.

6.1 Token circulation

A transaction (Tx) is a cryptographically-signed command created by an external actor to the blockchain and validated by a designated actor on the blockchain [47]. The three patterns for token

transfer allow for the granular design of token-centered business processes. They describe how digital assets – fungible or non-fungible tokens - can be transferred between accounts on the blockchain.

6.1.1 Atomic Tx Pattern

Summary: A single-step transfer where an account directly moves its owned assets to another account.
Classification: Transfer
Context: A blockchain application supports transferable assets.
Applicability: - Use the Atomic Tx pattern to move tokens between accounts by tokens' owner.
- Do not use the Atomic Tx pattern to burn tokens by their owner; use the Digester pattern instead.
- Do not use the Atomic Tx pattern to move tokens on behalf of their owner; use the Micro Tx pattern instead.
Problem: How a digital asset can be moved between two participants on the blockchain?
Solution: Blockchain technology allows business participants to transact in a peer-to-peer manner, eliminating the need for a central authority. Thus, digital assets can be transferred directly by their owners to other accounts as an atomic transaction. The blockchain records the existence of digital assets via tracking their ownership in an immutable distributed ledger. The digital assets are transferred through transferring their ownership, reflected by changing balances of the token receiver and sender. In the case of transferring fungible tokens, this pattern results in decreasing the sender balance and increasing the receiver balance based on the number of tokens transferred. In the case of transferring non-fungible tokens, the ownership of the transferred token is renounced from the sender and granted to the receiver. This pattern is the bottom line of other transfer transactions on the blockchain. For example, this pattern is an enabler for atomic swaps between two types of tokens. A token swap is a bidirectional Atomic Tx where a seller sends tokens to a buyer as an atomic transaction, and the buyer sends equivalent tokens to the seller as another atomic transaction. In an atomic transaction, the transaction initiator is the token's owner, and the token transfer flows directly from the owner's balance to the receiver's balance.
Business Process Meta-Model: Fig. 3 shows the meta-model of the Atomic Tx pattern business process.

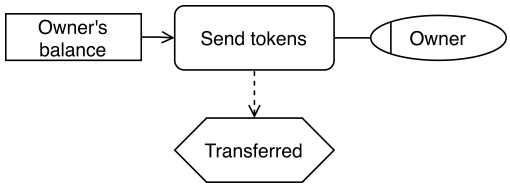

Fig. 3 Atomic Tx pattern EPC

Requirement Template: The system shall allow user to send and receive tokens on the blockchain.
Extended Requirement Template:
FR1:The system shall allow <user> [on <blockchain network>] to send <token> from <user's> balance to <receiver> [on <blockchain network>] [when <conditions met>]
FR2:The system shall emit <Transferred> event when <token> is successfully transferred [from <owner> to <receiver>]
Implementation Considerations: Define a method in the smart contract of a token to handle token transfers by owners, e.g., `transfer()`. The method should get the receiver's address and amount or ID of tokens to be transferred as minimum arguments. Check that invoking address owns the token if non-fungible and that amount of token is within the balance of the sender if fungible. Check that both sender and receiver are not null addresses. After each transfer, you should update the balances of both sender and receiver based on the transferred tokens. You may need to check custom transfer conditions if required. If the blockchain platform allows for user-defined events, consider defining completion events, such as `Transferred`, with relevant arguments. Refer to Exclusive Authorization and Shared Authorization patterns to specify authorization for accessing the transfer method. In the context of multi-token applications, consider implementing fungibility-agnostic token smart contracts. That is, a single token contract represents multiple types of tokens instead of having a separate contract for each token type. This approach leads to less gas consumption, reducing deployment cost and complexity. You may also consider batch transfers to allow transferring multiple tokens

in a single transaction. In the context of multichain applications, consider specialized tools, such as Layer 2 solutions, to process cross-chain operations. Check your implementation against platform-specific standards for compatibility.
Testing Considerations: Ensure only the token's owner can transfer the tokens. Ensure to-be-transferred tokens are within the balance of the sender. Ensure the token owner's balance is decreased after each transfer. Ensure the token owner's balance is decreased with the transferred amount. Ensure the receiver's balance is increased after each transfer. Ensure the receiver's balance is increased with the transferred amount. Ensure only existing tokens can be transferred. If defined, ensure transfer conditions are not bypassed. If relevant, make extra tests for cross-chain operations. If relevant, refer to Testing Considerations of Exclusive Authorization and Shared Authorization patterns for relevant hints on testing. If defined, ensure the smart contract emits desired events upon completion of ownership transfer.
Example: In the example application, the Invitee transfers their PASS to the Receptionist at the time of event check-in.
Related Patterns: Micro Tx and Macro Tx patterns.
Known Uses: CryptoCrystal, Landemic, and FOAM allow their users to transfer CC, LAND, and FOAM tokens, respectively, as atomic transactions.

6.1.2 Micro Tx Pattern

Summary: A single-step transfer where an account moves assets owned by another account directly.
Classification: Transfer
Context: A blockchain application supports transferable assets.
Applicability: - Use the Micro Tx pattern to move tokens on behalf of their owner. - Do not use the Micro Tx pattern to burn tokens on behalf of their owner; use the Digester pattern instead. - Do not use the Micro Tx pattern to specify that an account is authorized to transact on behalf of another account; use the Approval pattern instead. - Do not use the Micro Tx pattern to move tokens by their owner; use the Atomic Tx pattern instead.
Problem: Transfer transactions of digital assets on the blockchain must be approved by the assets' owner. Sometimes, transactions need to be completed when the owner is unavailable. How can digital assets be transferred on the blockchain when their owner is unavailable?
Solution: This pattern allows tokens to be transferred between accounts by approved delegates on behalf of the token's owners. This pattern allows for automation. For example, a user may approve an exchange smart contract to automatically trade a certain amount of the user's cryptocurrency. There is no need to wait for the user to approve each trade transaction. This pattern can also be seen in token sales, where a project owner can mint an initial token supply and approve a crowd sale smart contract to release tokens to customers on behalf of the owner. Applying this pattern results in decreasing both sender's balance and delegate's allowance and increasing the receiver's balance. The transaction initiator is the delegate, and the token transfer flows directly from the owner's balance to the receiver's balance.
Business Process Meta-Model: Fig. 4 shows the meta-model of the Micro Tx business process.

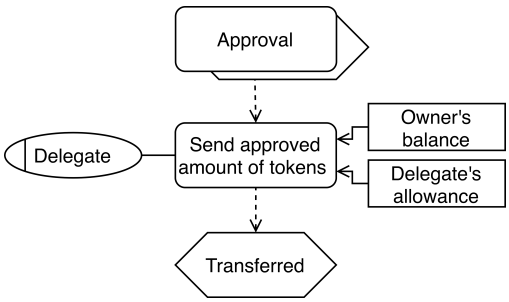

Fig. 4 Micro Tx pattern EPC

Requirement Template: The system shall allow user to transact on behalf of another user on the blockchain.
Extended Requirement Template:
FR1:The system shall allow <delegate> [on <blockchain network>] to send <token> from <delegator's> balance [on <blockchain network>] to <receiver> [on <blockchain network>] [when <conditions met>]
FR2:The system shall emit <Transferred> event when <token> is successfully transferred [from <delegator> to <receiver>]
Implementation Considerations: Define a method in the smart contract of a token

to handle token transfers by delegates, e.g., `transferFrom()`. The method should get the token's owner address as sender, receiver address, and amount or ID of tokens to be transferred as minimum arguments. Check that the invoking address is an authorized delegate by the sender. Check that the transferred tokens are within the delegate allowance. Check that the transferred tokens are within the sender balance. Check that both sender and receiver are not null addresses. After each transfer, you should update the balance of the sender, the balance of the receiver, and the allowance for the delegate based on the spent tokens. You may need to check custom transfer conditions if required. If the blockchain platform allows for user-defined events, consider defining completion events, such as `TransferredBy`, with relevant arguments. Refer to Implementation Considerations of Atomic Tx pattern for hints for multi-token and multichain implementation contexts. Check your implementation against platform-specific standards for compatibility.
Testing Considerations: Ensure only the authorized delegate can spend tokens from the token owner's balance. Ensure only the approved allowance can be spent by the delegate from the sender's balance. Ensure only existing tokens can be transferred. Ensure the delegate's allowance is decreased after each transfer. Ensure the delegate's allowance is decreased based on the spent tokens. Ensure the token owner's balance is decreased after each transfer. Ensure the token owner's balance is decreased with the spent amount. Ensure the receiver's balance is increased after each transfer. Ensure the receiver's balance is increased with the spent amount. If relevant, make specific tests for cross-chain operations. If defined, ensure the smart contract emits desired events upon completion of a token transfer.
Example: In the example application, PASS is owned by the Invitee. When the Representative check-in at the event, PASS is transferred from the Invitee's wallet by the Representative on behalf of the Invitee to the Receptionist. The application shall allow the Representative to send PASS from the Invitee's balance to the Receptionist on Ethereum when event check-in.
Related Patterns: This pattern requires the Approval pattern. Atomic Tx and Macro Tx are alternatives to this pattern.
Known Uses: - FOAM, Upfiring, and TimeX allow users to transfer tokens on behalf of the tokens' owners.

6.1.3 Macro Tx Pattern

Summary: A multi-step transfer where tokens are indirectly moved from owner account to receiver account.
Classification: Transfer
Context: A blockchain application requires a guarantee for token transfers.
Applicability: - Use the Macro Tx pattern to move tokens in multiple steps from the token's owner balance to the receiver's balance. - Do not use the Macro Tx for simple frequent token transfers; use Atomic Tx or Micro Tx instead.
Problem: In some cases, especially commercial ones with high-value assets, there is a potential risk of fraud. Central payment systems usually mitigate this risk. The blockchain tends to eliminate the centralization of transactions. How to guarantee the sending and receiving of digital assets?
Solution: Blockchain eliminates the need for central systems between sender and receiver, using intermediary smart contracts. A token in this pattern can be generally transferred in two approaches: deposit-withdraw or approve-deposit-withdraw. This pattern is a composition of other BPMPs. The deposit-withdraw approach is a composition of multiple sequential applications of the Atomic Tx pattern. The approve-deposit-withdraw approach is a composition of the Approval, Micro Tx, and Atomic Tx patterns. The general mechanism can be described as follows. Firstly, the sender transfer required tokens from its balance to an intermediary smart contract as a deposit. Alternatively, the sender can approve the intermediary smart contract to transfer the deposit on behalf of the sender. The intermediary smart contract holds the deposit until release conditions are met. Then, the intermediary smart contract transfers the deposit to the receiver as a withdrawal. The deposited tokens are transferred back to the owner if the conditions are not met. The token transfer flows indirectly from the owner's balance to the receiver's balance, where the initiator of the first transfer is either the tokens' owner or an authorized delegate, and the second transfer is the intermediary smart contract. A simple example of the deposit-withdraw

approach can be illustrated through its application to reduce the risk of fraud in a social media prize draw, where a winner is chosen from several entrants. First, prize tokens are transferred by an influencer to an intermediary smart contract via an atomic transaction. The participants can see the deposited tokens and verify encoded release conditions due to the transparency of the blockchain. Next, winners can claim their prizes within a defined period. Once verified as an eligible winner, the intermediary smart contract transfers the tokens to the winner via an atomic transaction. By the end of the period, any unclaimed tokens are sent back to the influencer via an atomic transaction. A simple example of the approve-deposit-withdraw approach can be illustrated through its application in a token exchange market context. First, the token's owner acts as an approver and authorizes the Exchange smart contract to transfer a certain number of tokens on its behalf. Then, Exchange transfers the tokens from the depositor balance to its account and holds it for the beneficiary as a Micro Tx. Later, the Exchange performs an atomic transaction to transfer the deposited tokens to the beneficiary.

Requirement Template: The system shall allow user to send tokens to another user through intermediary account.

Extended Requirement Template:

FR1:The system shall allow <sender> [on <blockchain network>] to transfer <token> to <receiver> [on <blockchain network>] through <intermediary> [on <blockchain network>] as follows:

FR1.1:The system shall allow <sender> [on <blockchain network>] to transfer <token> to <intermediary> [on <blockchain network>] [when <condition met>].

FR1.2:The system shall allow <intermediary> [on <blockchain network>] to send <token> from <sender's> balance [on <blockchain network>] to <receiver> [on <blockchain network>] [when <conditions met>]

FR1.3:The system shall allow <intermediary> [on <blockchain network>] to transfer <token> to <receiver> [on <blockchain network>] [when <condition met>].

FR2:The system shall emit <Deposited> event when <token> is successfully transferred by <sender> to <intermediary>

FR3:The system shall emit <Withdrawn> event when <token> is successfully transferred by <intermediary> to <receiver>

Business Process Meta-Model: Fig. 5 shows the meta-model of the Macro Tx business process.

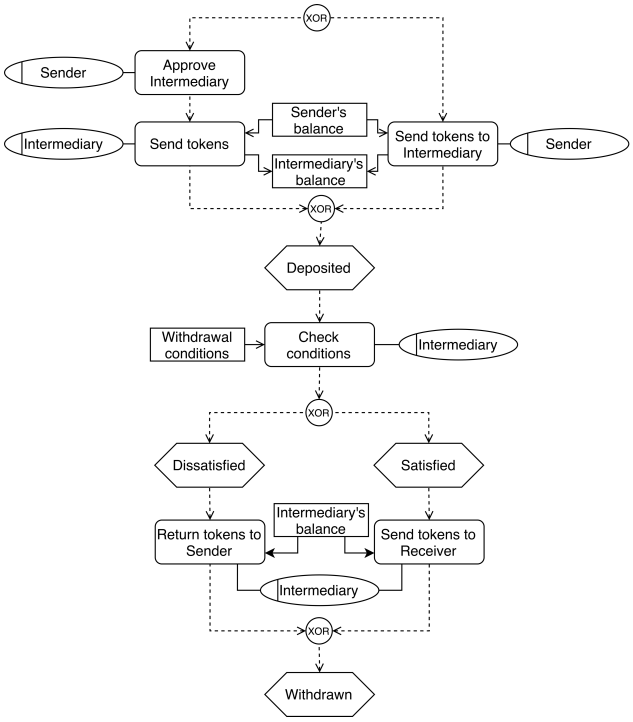

Fig. 5 Macro Tx pattern EPC

Implementation Considerations: Refer to the Implementation Considerations of the Atomic Tx, Micro Tx, and Approval patterns.

Testing Considerations: Refer to the Testing Considerations of the Atomic Tx, Micro Tx, and Approval patterns.

Example: In the example application, this pattern is applied for registration fee payments. The settlement procedure and conditions are specified in the Escrow SC. Upon completion of the registration process, the Invitee must deposit the fee in Ether to the Escrow within a specified time. When the Escrow confirms that the fee is paid in full and in Ether, a confirmation event is informed to the Registrar to issue a PASS. Once the Invitee receives the PASS, a confirmation event is informed to the Escrow. The Escrow releases the deposited fee to the Accountant when the predefined conditions are met. In case the Invitee fails to fulfill the payment conditions or withdrawal conditions are not met, such as PASS is

not received by the Invitee, any deposited tokens by the Invitee are sent back.
Related Patterns: Atomic Tx, Micro Tx, and Approval patterns.
Known Uses: FOAM, Upfiring, and TimeX.

6.2 Token supply

These patterns describe possible solutions on how the total amount of a token on a blockchain network can be increased or decreased.

6.2.1 Bloom Pattern

Summary: This pattern enables creating new tokens as required.
Classification: Lifecycle
Context: A blockchain application requires a dynamic supply of digital assets.
Applicability: Use the Bloom pattern to create a token generation mechanism.
Problem: Token standards describe how tokens can be used, but how can tokens be created?
Solution: Tokens can be generated in different ways. First, fixed supply is where the total amount of tokens is minted at the initial phase of a project and then made available to users via a token sale. Usually, the total supply amount is hard coded in the token smart contract source code. Second, flexible limited supply is another way where tokens are generated when a certain condition is met, such as mining rewards. Still, the total amount of tokens that can be ever generated will not exceed the maximum supply limit. Finally, flexible unlimited supply is where tokens are generated conditionally with no upper limit for the total amount of tokens that can be ever generated. In the flexible supply, the token generation mechanism is implemented in the smart contract source code, declaring the maximum supply if applicable. This approach affects the total supply of the destroyed token increasingly.
Business Process Meta-Model: Fig. 6 shows the meta-model of the Bloom business process.
Requirement Template: The system shall provide a mechanism to generate new tokens.
Extended Requirement Template:
FR1:The system shall allow <minter> [on <blockchain network>] to send <token> from <mint address> [on <blockchain network>] to <receiver> [on <blockchain network>] [when <conditions met>]

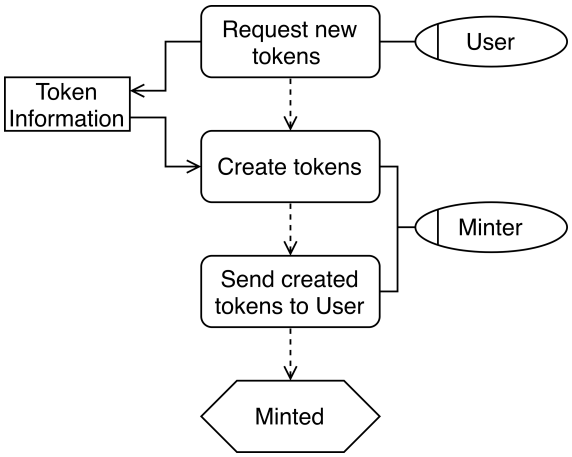

Fig. 6 Bloom pattern EPC

FR2:The system shall emit <Minted> event when new <token> is successfully generated.
Implementation Considerations: Define a method in the smart contract of a token to handle token generation, e.g., `mint()`. The method should get the receiver's address and amount or ID of tokens to be generated as minimum arguments. Enforce transfer from the mint address, e.g., zero address. Check that receiver is not a null address. If implementing a limited supply token, check that the total amount of existing tokens does not exceed the token total supply limit. In the case of non-fungible tokens, check that the to-be-minted token ID must not exist. After each transfer, you should update the receiver's balance based on the transferred tokens. You may need to check custom transfer conditions if required. If the blockchain platform allows for user-defined events, consider defining completion events, such as `Minted`, with relevant arguments. Refer to Exclusive Authorization and Shared Authorization patterns to specify authorization for accessing the mint method. In the context of multi-token and multichain applications, refer to the Implementation Considerations of the Atomic Tx pattern for relevant guidelines. Check your implementation against platform-specific standards for compatibility.
Testing Considerations: Ensure transfer is always from the mint address. Ensure only non-existing tokens can be minted. Ensure every generated non-fungible token is assigned a unique ID. Ensure the receiver's balance is increased after each transfer. Ensure the receiver's balance is increased based on the generated tokens. If defined, ensure transfer conditions are not

bypassed. If relevant, make extra tests for cross-chain operations. If relevant, refer to Testing Considerations of Exclusive Authorization, Shared Authorization, and Atomic Tx patterns for relevant hints. If defined, ensure the smart contract emits desirable events upon completion of relevant functions.
Example: In the example application, PASS should be issued for the Invitee when the registration fee is paid. Since a limited number of people will be invited, the maximum number of PASS tokens is limited. Thus, PASS token supply should be implemented as a flexible limited supply. The maximum supply can be defined as a formula to suit different capacity events. Tokens on Ethereum are generated by a null address to a specified account. The null address (i.e., 0x0...0) is not owned by any user and is often associated with token mint events on Ethereum.
Related Patterns: Exclusive Authorization and Shared Authorization patterns.
Known Uses: - CryptoCrystal implemented an initial supply of 20,000,000 for the PKX token. CryptoStrikers implemented a mint limit for a given checklist item, based on its tier. In POA Bridge, transferring from POA network to Ethereum result in minting a new POA20 token on Ethereum.

6.2.2 Digester Pattern

Summary: This pattern enables destroying existing tokens as required.
Classification: Lifecycle
Context: A blockchain application requires a dynamic token supply.
Applicability: Use the Digester pattern to destroy existing tokens, either by the tokens' owner or an authorized delegate.
Problem: Tokens may become undesirable for different reasons, such as expiry of the underlying asset, time-constrained availability, tokens redemption, or low market value. Thus, they must be removed. However, the immutability of the blockchain ensures that existing tokens are irremovable. Hence, how to prevent the use of undesirable tokens?
Solution: Since undesirable tokens are irremovable, they should be made inaccessible. This can be achieved by sending the undesirable tokens to a receive-only address, i.e., a burn address. Such address holds received tokens persistently and cannot initiate transfers to other accounts. Therefore, this process is irreversible, making tokens inaccessible. This pattern affects the total supply of the destroyed token decreasingly, which increases the token market value. Also, this pattern prevents unauthorized use of spent tokens, such as when they are redeemed or expired.
Business Process Meta-Model: Fig. 7 shows the meta-model of the Digester business process.

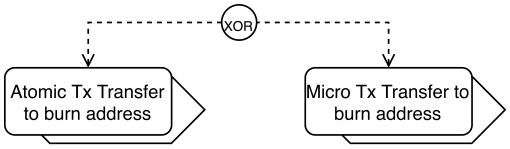

Fig. 7 Digester pattern EPC

Requirement Template: The system shall provide a mechanism to destroy existing tokens.
Extended Requirement Template:
FR1:The system shall allow <user> on <blockchain network> to send <token> from <user's> balance to <burn address> on <blockchain network> [when <conditions met>]
FR2:The system shall allow <user> on <blockchain network> to send <token> from <another user's> balance on <blockchain network> to <burn address> on <blockchain network> [when <conditions met>]
Implementation Considerations: Define a method in the smart contract of a token to handle token generation, e.g., `burn()`. The method should get the token owner's address and amount or ID of tokens to be destroyed as minimum arguments. Enforce transfer to the burn address, e.g., zero address. Check that sender is not a null address. Check that the invoking address is the owner of the to-be-burn token or an approved delegate. Check that invoking address is an authorized account to burn tokens. In the case of non-fungible tokens, check that the to-be-burned token ID must exist. After each transfer, you should update the token owner's balance based on the burned tokens. After each transfer, you may need to update the allowance of the delegate, if applicable, based on the burned tokens. You may need to check custom burn conditions if required. If the blockchain platform allows for user-defined events, consider defining completion events, such as `Burned`, with relevant arguments.

Refer to the Exclusive Authorization and Shared Authorization patterns to specify authorization for accessing the burn method. In the context of multi-token and multichain applications, refer to Implementation Considerations of the Atomic Tx pattern for relevant guidelines. Check your implementation against platform-specific standards for compatibility.
Testing Considerations: Ensure transfer is always to the burn address. Ensure only the token's owner or approved delegate can burn the tokens. Ensure only existing tokens can be burned. Ensure to-be-burned tokens are within the balance of the token owner. Ensure to-be-burned tokens are within the allowance of the delegate. Ensure the token owner's balance is decreased after each transfer. Ensure the token owner's balance is decreased with the burned tokens. Ensure the delegate's allowance is decreased after each transfer. Ensure the delegate's allowance is decreased with the burned tokens. If defined, ensure burn conditions are not bypassed. If relevant, make extra tests for cross-chain operations. If relevant, refer to Testing Considerations of Exclusive Authorization, Shared Authorization, and Atomic Tx patterns for relevant hints. If defined, ensure the smart contract emits desirable events upon completion of relevant functions.
Example: In the example application, PASS becomes dispensable once used at the event check-in. Thus, it should be destroyed to prevent double-spending. The Invitee provides his PASS to the Receptionist, who is responsible for burning PASS tokens by sending them to the Ethereum's null address. The null address (i.e., 0x0...0) is not owned by any user and is often associated with token burn events on Ethereum.
Related Patterns: Approval pattern.
Known Uses: CryptoCrystal, CryptoStrikers allows for burning their tokens as part of the game logic. In POA Bridge, transferring POA20 from Ethereum back to POA Network result in the burning of the POA20 token on Ethereum.

6.3 Token authorization

Patterns in this section describe possible solutions on how accounts can be authorized to perform token-centered business processes.

6.3.1 Approval Pattern

Summary: This pattern allows an account to approve other accounts to transact on its behalf.
Classification: Accessibility
Context: A blockchain application requires delegated token transfer transactions.
Applicability: - Use the Approval pattern to specify that an account is authorized to spend allowance from the balance of another account. - Use the Approval pattern to change the allowance of an approved delegate. - Do not use the Approval pattern to move tokens on behalf of their owner; use the Micro Tx pattern instead.
Problem: Micro transfer transactions are performed by a third-party account (delegate) on the balance of another account (approver). How can blockchain verify that it is a legitimate delegate performing a valid transaction?
Solution: Before a delegate can perform any transaction on the approver's balance, the token owner must approve the delegate account through an approval transaction, determining the number of tokens the delegate can spend on behalf of the approver. This way, the blockchain can verify that the token owner authorizes the delegate and that the number of tokens spent by the delegate is within the allowed amount by the approver.
Business Process Meta-Model: Fig. 8 shows the meta-model of the Approval business process.

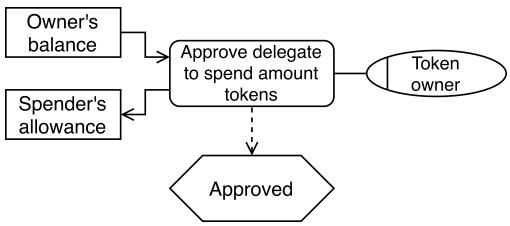

Fig. 8 Approval pattern EPC

Requirement Template: The system shall allow user to approve another user to transact on their behalf on the blockchain.
Extended Requirement Template:
FR1:The system shall allow <token owner> on <blockchain network> to approve <delegate> on <blockchain network> to spend <allowance> from <token owner's> balance on <blockchain network>
Implementation Considerations: Define a method in the token smart contract that the token

owner can call to approve another account, e.g., `approve()`. The method should get the delegate account address and the allowed amount as minimum parameters. Check that the allowance is less than or equal to the approver balance for fungible tokens. For non-fungible tokens, check that the token exists and that the approver owns the token or is an approved delegate to manage the token. Check that both approver and delegate are not null addresses. When changing an allowance for an approved delegate, you should create a mechanism to prevent the delegate from using the new allowance in addition to the old one; a case may occur by unfortunate transaction ordering described in [48]. Refer to Micro Tx pattern for more on managing delegate's allowance. If the blockchain platform allows for user-defined events, consider defining completion events, such as `Approval` with relevant arguments. In the context of multichain operations, consider the Shared Authorization pattern.
Testing Considerations: Ensure a user can approve a delegate. Ensure a user can specify allowance for a delegate. Ensure the application does not bypass any defined condition. Ensure changing an allowance for a delegate replaces the previous allowance. If relevant, make extra tests for cross-chain operations. If defined, ensure the smart contract emits desired events upon completion of ownership transfer.
Example: In the example application, PASS is issued for Invitee. The representative is approved to use PASS on behalf of the Invitee to attend the event. The application shall allow the Invitee to approve the Representative to spend one PASS from the Invitee's balance on Ethereum.
Related Patterns: Micro Tx pattern.
Known Uses: CryptoCrystal, FOAM, and Upfiring allow users to approve delegates to spend a specified amount of tokens on their behalf.

7 Smart contract-oriented patterns

SCOPs comprise a collection of patterns from the perspective of an SC. The process of terminating and resuming an SC can be designed using the Switch pattern, which is depicted in Section 7.1. The process of authorizing accounts on an SC can be designed using the Exclusive Authorization and Shared Authorization pattern. These SC authorization patterns are depicted in Section 7.2.

7.1 Smart contract security

The pattern in this category describes how to control the execution of SCs.

7.1.1 Switch Pattern

Summary: This pattern enables terminating and resuming smart contracts as required.
Classification: Lifecycle
Context: A blockchain application requires a controlled execution of smart contracts.
Applicability: - Use the Switch pattern to suspend smart contract operations temporarily or permanently. - Do not use the Switch pattern to specify that an account is authorized to suspend the smart contract; use the Exclusive Authorization or Shared Authorization patterns instead.
Problem: Once smart contracts are deployed on the blockchain, they become immutable and cannot be altered. In some situations, such as software upgrades, vulnerable contracts, or temporary suspension of a token trade, there is a need to suspend smart contract operations totally or partially, temporarily or permanently. How to control the execution of a smart contract running on a blockchain network?
Solution: A flexible suspension mechanism can be designed to terminate the execution of a smart contract. This pattern allows for pausing a contract or part of its methods when certain conditions are met. When a smart contract is paused, all its methods become inaccessible and, thus, inexecutable. This prevents other addresses from using business logic defined in that smart contract. Unpausing the smart contract contrasts the situation. This pattern can also be applied on a method-level in which certain methods within a smart contract are suspended. An authorization mechanism should be applied with this pattern.
Business Process Meta-Model: Fig. 9 shows the meta-model of the Switch business process.
Requirement Template: The system shall provide the ability to terminate and resume smart contract operations.
Extended Requirement Template:
FR1:The system shall allow <user> to <pause/unpause> <[method in] smart contract

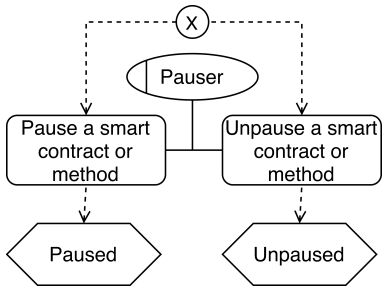

Fig. 9 Switch pattern EPC

instance> on <blockchain network>
FR2:The system shall emit <Paused> event when <smart contract instance> is successfully <paused>
FR3:The system shall emit <Unpaused> event when <smart contract instance> is successfully <unpaused>
Implementation Considerations: Define a global variable that stores the current state of the smart contract. If this pattern is to be applied on a method-level, use separate variables to distinguish between terminating a smart contract and terminating specific methods within the contract. Use modifiers, e.g., `whenPaused` and `whenNotPaused`, to constrain the execution of restricted methods based on the current state of the smart contract. Define specific restricted methods to handle state changes of the smart contract, e.g., `pause()` and `unpause()` methods. Refer to Exclusive Authorization and Shared Authorization patterns to specify authorization for accessing these methods. If the blockchain platform allows for user-defined events, consider defining completion events, such as `Paused` and `Unpaused`, with relevant parameters.
Testing Considerations: Ensure required operations are stopped when they are paused. Ensure required operations are functional when resumed. Ensure constrained methods are accessible only by authorized addresses. Refer to Testing Considerations of Exclusive Authorization and Shared Authorization patterns for relevant hints on testing. If defined, ensure the smart contract emits desirable events upon completion of relevant functions.
Example: The example application is based on Ethereum, a platform where smart contracts cannot be paused on a protocol-level. Thus, we need modifiers in the source code to enable and disable smart contract operations as required. During the registration, the PASS mint method should be enabled for Registrar. When registration is closed, the mint method should be disabled. When checking in to the event, the burn method should be accessible by an authorized attendee, be it registered Invitees or their Representatives. When the event ends, the PassSC should be terminated. Modifiers for each stage should be defined in the PassSC source code, and their values should only be altered by the event's Manager. By default, PassSC is paused. For each event, PassSC methods are enabled and disabled as required.
Related Patterns: Exclusive Authorization and Shared Authorization patterns.
Known Uses: CryptoStrikers, Nestree, and MyCryptons.

7.2 Smart contract authorization

Patterns in this section describe possible solutions on how accounts can be authorized to perform SCC business processes.

7.2.1 Exclusive Authorization Pattern

Summary: An authorization mechanism with a single role for all privileged actions. The role is transferable between accounts.
Classification: Accessibility.
Context: A blockchain application requires a single administrative account of smart contracts.
Applicability: - Use the Exclusive Authorization pattern to specify that an account on the same chain is exclusively authorized to access certain smart contract functions. - Do not use the Exclusive Authorization pattern when granular permissions are required; use the Shared Authorization pattern instead. - Do not use the Exclusive Authorization pattern when the owner address is on another chain; use the Shared Authorization pattern instead.
Problem: By default, A smart contract has no owner. No account has special privilege on a smart contract once deployed on the blockchain, including the smart contract's deployer. By default, all deployed smart contracts are accessible and callable by all accounts on the blockchain, especially in permissionless settings. Thus, critical methods may be used maliciously. To prevent unauthorized access to smart contracts, access control can be embedded in smart contracts to authenticate accounts trying to access restricted

methods. How to authorize an account to access restricted methods in a smart contract?
Solution: The exclusive authorization is a simple mechanism of access control with a single account, i.e., the contract owner is authorized for all privileged actions. It is a one-to-many relationship where a contract owner is authorized for all privileged actions. The current owner can transfer the ownership of the smart contract to another account. The current owner initially nominates an account as a new contract owner. Once the nominated account accepts the ownership, all privileges are transferred to the new owner and revoked from the previous owner. By default, the owner of an ownable smart contract is its deployer, who can transfer the contract's ownership to another account, e.g., another organizational unit, as an atomic transaction.
Business Process Meta-Model: Fig. 10 shows the meta-model of the Exclusive Authorization business process.

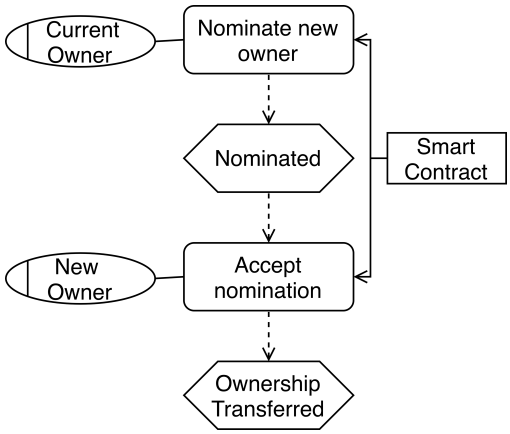

Fig. 10 Exclusive Authorization pattern EPC

Requirement Template: The system shall provide exclusive access control for certain smart contracts
Extended Requirement Template:
FR1:The system shall provide exclusive access control for <smart contract instance> on <blockchain network>
FR2:The system shall allow <user> to own <smart contract instance> on <blockchain network>
FR3:The system shall allow the owner of <smart contract instance> to transfer ownership of the smart contract to another user on <blockchain network>
FR4:The system shall emit <Ownership Transferred> event when smart contract ownership is successfully transferred.
Implementation Considerations: The contract owner can be defined as a variable within the smart contract source code. The variable stores the address of the current contract owner. Use a single modifier, e.g., `onlyOwner`, to restrict method execution to the contract owner address. Define a specific restricted function, e.g., `transferOwnership()` that can modify the owner address; to only allow the current owner to transfer the contract's ownership to another account. The method should get the new owner address as minimum parameters. You may need to check if the nominated address is a null address to prevent or enforce the transferability of the contract's ownership. If the blockchain platform allows for user-defined events, consider defining completion events, e.g., `OwnershipTransferred`, with relevant parameters.
Testing Considerations: Ensure restricted methods are accessible only by the owner's address. When ownership is transferred, ensure the previous owner can no longer access restricted methods, and the new owner's address can access the restricted methods. If defined, ensure the smart contract emits desired events upon completion of ownership transfer.
Example: Since the example application intends to be a reusable solution for differently managed events, Registrar SC can implement the Exclusive Authorization mechanism in which the contract's ownership can be transferred between event managers as required.
Related Patterns: Shared Authorization
Known Uses: FOAM, Nestree, and Upfiring.

7.2.2 Shared Authorization Pattern

Summary: An authorization mechanism with a separate role for each privileged action. A role can have many authorized accounts, and an authorized account can have many roles.
Classification: Accessibility
Context: A blockchain application requires configurable authorization.
Applicability: - Use the Shared Authorization pattern to specify that a set of accounts on the same chain is authorized or is not authorized to

access certain smart contract functions. - Use the Shared Authorization pattern to specify that an account or a set of accounts on another chain is authorized or is not authorized to do certain smart contract operations.

Problem: The simplicity of the exclusive authorization can be useful for use cases where granular permissions are not required. How can business logic in a smart contract reflect complex use cases where different authorization levels are needed?

Solution: Instead of a simple ownership approach, each privileged action is assigned a separate role in this pattern. This pattern establishes a many-to-many relationship between accounts and roles. For example, an account may have a minter and pauser role, whereas the minter role may be assigned multiple accounts. More granular permissions can be implemented by splitting concerns, creating complex permission structures. For example, this pattern can be realized in situations resembling organizational charts, when authorization is given to an account for a limited time, or when the list of authorized addresses is unknown beforehand. In this scheme, the contract owner has a contract admin role who can grant and revoke other roles. The dynamicity of role allocation makes it a solution to manage simple applications by assigning all roles to the contract admin. In this pattern, the contract's administration can be transferred by assigning the contract's admin role to another account. In contrast to the Exclusive Authorization Pattern, changing the admin of the smart contract in this pattern does not tacitly grant all the other roles to the new admin; instead, it transfers the control over other roles to the new admin.

Business Process Meta-Model: Fig. 11 shows the meta-model of the Shared Authorization pattern business process.

Requirement Template: The system shall provide role-based access control for certain smart contracts.

Extended Requirement Template:

FR1:The system shall allow <default admin> on <blockchain network> to administer <smart contract instance> on <blockchain network>

FR2:The system shall allow <role admin> on <blockchain network> to assign <role> to other users for <smart contract instance> on <blockchain network>

FR3:The system shall allow <role admin> on <blockchain network> to revoke <role> from other users for <smart contract instance> on <blockchain network>

FR4:The system shall allow <user> on <blockchain network> to act as <assigned role> for <smart contract instance> on <blockchain network>

FR5:The system shall prevent <user> on <blockchain network> to act as <unassigned role> for <smart contract instance> on <blockchain network>

FR6:The system shall allow <default admin> of <smart contract instance> on <blockchain network> to transfer administration of the smart contract to another user on <blockchain network>

FR7:The system shall emit <Role Granted> event when a role is successfully granted to an account.

FR8:The system shall emit <Role Revoked> event when a role is successfully revoked from an account.

FR9:The system shall emit <Role Admin Changed> event when <role admin> is successfully changed.

FR10:The system shall emit <Role Admin Changed> event when <default admin> of <smart contract instance> is successfully changed.

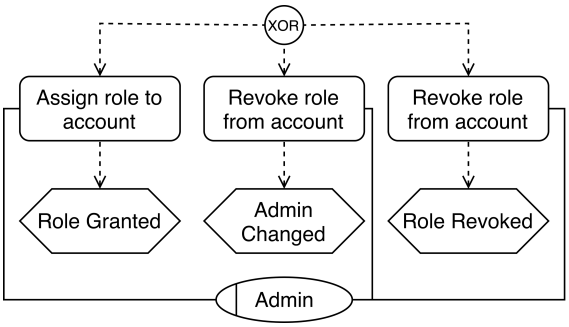

Fig. 11 Shared Authorization pattern EPC

Implementation Considerations: Use a separate constant for each role, e.g., `MINTER_ROLE` and `BURNER_ROLE` represent two different roles that can be assigned to one account. Use separate modifiers for each role to authenticate access to restricted methods. e.g., `onlyMinter` and `onlyBurner` can be defined to restrict access to mint and burn functions. Define role granting and revoking methods for each role. There should

be a default admin role that can be authenticated with an `onlyAdmin` modifier to restrict the execution of administrative methods to the contract's admin. Be aware that bearing a role does not imply that the role-bearer can assign or revoke the role from other accounts. Consider defining a role admin for each role if desired, e.g., to reflect an organizational chart. Define a specific restricted function, e.g., `changeAdmin()`, that can modify the contract admin address; to only allow the current admin to transfer the contract's administration to another account. The method should get the new admin address as minimum parameters. You may need to check if a nominated admin address is a null address; to prevent or enforce the transferability of the contract's administration. If the blockchain platform allows for user-defined events, consider defining completion events, such as `Role Granted`, `Role Revoked`, and `Admin Changed` with relevant parameters. In the context of multichain applications, refer to the Atomic Tx pattern for relevant guidelines. You may need to check your implementation against platform-specific standards for compatibility.

Testing Considerations: Ensure a role can access methods it has permission to. Ensure a role cannot access methods the role is not permitted to access. Ensure an authorized account can perform the assigned role. Ensure an authorized account is not able to perform unassigned roles. Ensure a role admin can assign the role to other accounts. Ensure a role admin can revoke the role from an authorized account. Ensure the contract admin can transfer contract administration to another account. When a contract admin is changed, ensure the new admin can do relevant operations, and the previous admin can no longer do so. If relevant, make extra tests for cross-chain operations. Ensure permissions are controlled concerning chains where authorized addresses belong. If defined, ensure the smart contract emits desirable events upon completion of relevant functions.

Example: The example application requires different roles for different accounts. Primarily, the Manager is the contract owner who can administer the SC by default. The Manager can assign roles to other accounts. A pauser role is required to terminate and resume SC operations. This role is kept for the Manager. A minter role should be assigned to the Registrar to be able to issue new PASS tokens as required. A burner role should be assigned to the Receptionist to be able to destroy PASS upon check-in.

Related Pattern: Exclusive Authorization pattern

Known Uses: Upfiring, CryptoStrikers, and Zinc.

8 Validation of the proposed pattern language

The validation is performed through CC against existing BCApps. The analysis is twofold. The first aimed at validating the BPMPs individually in different applications. The second aimed at validating the BPMPs collectively within a single application. The validity of the individual patterns is implied by the pattern's identification process. That is, deriving the patterns factually from existing applications assures the existence of the pattern beforehand. The goal of applying CC is to quantitatively validate the existence of the patterns per BCApp.

Table 4 shows the number of BCApps that conforms to each identified pattern, grouped by application domains. The Atomic Tx pattern is realized in 37 out of 50 applications. The Micro Tx and Macro Tx patterns are realized in 7 out of 50 each. That is, the token transfer patterns are realized in 40 out of 50 (80%) BCApps in the validation dataset, where seven imply more than one alternative.

The Bloom pattern is realized in 11 out of 50 applications, whereas the Digester pattern is realized only in five out of 50 applications. That is, the token supply patterns are realized in 11 out of 50 (22%) BCApps in the validation dataset, where five imply both patterns. The Approval pattern is realized in 26 out of 50 applications. That is, the token authorization pattern is realized in 52% of BCApps in the validation dataset.

The Switch pattern is realized in 8 out of 50 applications. That is, the smart contract security pattern is realized in 16% of BCApps in the validation dataset. The Exclusive Authorization pattern is realized in 13 out of 50 applications, whereas the Shared Authorization pattern is realized only in six out of 50 applications. That is, the smart contract authorization patterns are realized in 16

Table 4 The Distribution of Validation Applications per BPMP grouped by Application Domain

		Transfer Patterns			Lifecycle Patterns			Accessibility Patterns		
		Token Circulation			Token Supply		Smart Contract Security	Token Authz.	Smart Contract Authorization	
		Atomic Tx	Micro Tx	Macro Tx	Bloom	Digester	Switch	Approval	Exclusive Authz.	Shared Authz.
Application Domain	Finance	6	1	0	2	1	0	5	4	0
	Gambling	1	0	0	0	0	1	0	0	0
	Game	10	0	0	4	3	4	5	1	1
	Governance	0	0	0	0	0	0	0	0	0
	IT	6	0	0	2	1	0	5	1	2
	Marketplace	5	3	3	1	0	2	4	4	1
	Media	3	2	2	1	0	0	3	1	1
	Real Estate	2	1	1	0	0	0	1	1	0
	Social	4	0	1	1	0	1	3	1	1
	TOTAL	37	7	7	11	5	8	26	13	6

Authz. = Authorization

out of 50 (32%) BCApps in the validation dataset, where three of them imply both patterns.

Most applications that imply Transfer patterns are Marketplace and Game applications, followed by Finance and Media applications. Lifecycle patterns have a massive application in the Game domain. Marketplace and Finance applications show a major application of Accessibility patterns, followed by Information Technology (IT) and Game applications.

From a different point of view, the applications were grouped based on the count of the implied BPMPs. The highest value of BPMPs collectively found in a single application is six. That is, two out of 50 BCApps realize six of the identified BPMPs. The first is the CryptoStrikers application that implies the Atomic Tx, Bloom, Digester, Approval, Switch, and Shared Authorization patterns. The second is the Upfiring application that implies the Atomic Tx, Micro Tx, Macro Tx, Approval, Exclusive Authorization, and Shared Authorization patterns.

Contrarily, 13 out of 50 applications imply only a single pattern of the BPMPs. DopeRaider, Landemic, Mars, MoonCatRescue, PopulStay, Status, and ZED applications realize the Atomic Tx pattern. EtherHabits application realizes the Macro Tx pattern. Ether Stake, Set, and Wibson applications realize the Exclusive Authorization pattern. GeoEth and World of Ether applications realize the Switch pattern. Five out of 50 BCApps imply none of the identified BPMPs.

The other values of BPMPs collectively found in a single application out of 50 are as follows: five patterns are found in four applications, four patterns are found in eight applications, three patterns are found in seven applications, and two patterns are found in 11 applications. To this end, we can conclude that 90% of the validation applications imply 11%-67% of the identified BPMPs.

9 Discussion

9.1 Implications

Implications of our proposed BPMPL can be summarized into three facets: software engineering support, technology support, and literature support. In what follows, we discuss these facets

9.1.1 Software Engineering Support

Arguably, our proposed BPMPL can support BCApp development methodologies elaborated by practitioners and researchers. According to [5], most BCApp developers follow fully Agile Development (AD) due to its flexibility. They do not follow any modeling patterns, methods, or frameworks for deriving their developed applications, which would be a reason for the reported difficulty in communicating the requirements of stakeholders who lack technical blockchain background in the design of the BCApps [5]. Besides, Model-Driven Development (MDD) approaches for the development of BCApps were proposed by researchers [49]. MDD allows building software through sequential model transformations, starting with a business model [11]. In MDD, the prospective software is abstracted into models that are used to communicate the intended solution to stakeholders.

Both AD and MDD emphasize the involvement of stakeholders in requirement elicitation. However, MDD lacks the flexibility provided by AD, unless MDD is used in an agile-based approach, which has been proven as a good practice in software development [50]. Our proposed BPMPL can support the Agile Model-Driven Development of BCApps by providing a set of pluggable and platform-agnostic requirement templates and BPMs for recurrent business processes embedded within SCs. Besides, the proposed BPMPs can be relied on as a starting point to process reduction techniques for BCApp development [5].

Moreover, the unmethodical development of SCs hinders the comprehensive testing of BCApps [51]. The need for blockchain-specific testing methods is justified, and the lack of such methods is significant [49]. SC testing is a sophisticated assignment. It requires expert knowledge of business, scenarios, and blockchain-specific transaction variables. Test engineers of SCs need specialized technical and business validation skills, including software quality assurance, security, regulatory, compliance, and business process management skills [49, 51]. While not a testing method, our proposed BPMPL opens doors for novel SC testing methods. For example, the proposed BPMPL can be considered as an enabler for SC testing based on the requirements. Requirement-Based Testing (RBT) is a testing method where

test cases and conditions are derived from requirements [52], which has been proven in the industry [53].

Although the description of each BPMP provides a common language between different roles involved in developing a BCApp, certain sections target individual roles. For the requirement engineers, our proposed BPMPL impacts requirement elicitation, modeling, and documentation. The *Business Process Meta-Model* and *Requirement Template* allow requirement analysts to reuse them into requirement descriptions as a starting point for writing functional requirements of blockchain solutions.

Moreover, our proposed BPMPL supports blockchain solution designers and developers with *Implementation Considerations*. It provides implementation guidelines to satisfy the requirements of a business process, which are derived from industry-leading standards and generalized to be platform-agnostic guidelines. Likewise, our proposed BPMPL supports test engineers of BCApps with *Testing Considerations*. They are written primarily considering unit and user acceptance testing, but can also be manipulated for other software testing types.

Appendix A shows the modeling of the example use case, the Pass Management Application, as a result of applying the proposed BPMPL. Although it is not a significant application, the successful modeling of the example use case implies the soundness of the proposed BPMPL in analyzing and modeling the functional requirements of BCApps.

9.1.2 Technology Support

Our proposed BPMPL is mainly derived from SCs executed by Ethereum Virtual Machine (EVM). An EVM-compatible blockchain is any blockchain platform that uses EVM, or a compatible implementation of EVM, for transactions, data, and SC execution. According to Chain List [54], there are approximately 285 EVM-powered blockchain networks. This implies the applicability of our proposed BPMPL to design almost any application for EVM-compatible blockchains beyond Ethereum.

Additionally, the proposed BPMPs were written while taking interoperability into account. The *Requirement Template* has a `<on blockchain network>` parameter for multichain requirements. The *Implementation Considerations* and *Testing Considerations* passages provide guidelines for designing, implementing, and testing multichain applications.

9.1.3 Literature Support

The proposed BPMPs complement the proposed design patterns in the literature. Access control is widely used to control which methods of an SC an account can use. Authentication and authorization provide a means of access control for a system. Authentication in BCApps can be designed using patterns proposed in the literature, such as Access Restriction [36] and Embedded permission [38] patterns. In contrast, authorization can be designed using the *Exclusive Authorization* and *Shared Authorization* patterns proposed in this study.

Some design patterns in the literature are considered instantiations of some BPMPs proposed in this study, as the former provides a technology view of the latter, possibly in a specific context. Two-way Payment Channel [33] and Escrow patterns [32] are instantiations of the *Macro Tx* pattern in the context of payment applications. The Two-way Payment Channel pattern is designed for frequent transactions between two supply chain parties. The Escrow pattern is designed for infrequent high-value payments between buyer and seller. Token Swap pattern [32] instantiates the *Atomic Tx* pattern. The Token Swap pattern allows for direct trading between two types of tokens.

The Authorised Delegate pattern [32] combines the *Approval* and *Micro Tx* patterns in the context of payment applications. It is a design of how a delegate account can pay another account on behalf of the delegator. Ownership [36] and Authorization [39] patterns are examples of the *Exclusive Authorization* pattern. Termination [39] and Emergency Stop [37] patterns are instantiations of the *Switch* pattern. The Emergency Stop pattern to disable sensitive methods upon attacks on SCs. The Termination pattern is used to make an SC unresponsive. Burned Token [32] is an instantiation of the *Digester* pattern. The Burned Token pattern allows remarking redeemed tokens as unusable in payment systems.

9.2 Limitations

There are some limitations of our proposed BPMPL:

- The BPMPL considered only SC (on-chain) functionality. The frontend (off-chain) functionality is out of the scope of this study.
- Although the proposed BPMPs have been validated using existing applications, blockchain solution developers would input valuable to validate the proposed BPMPL in developing new applications.
- Despite its proven support for developing applications based on EVM-compatible platforms, the BPMPL was not tested for developing non-EVM-compatible BCApps.
- A different sample of event logs could result in different CC results. For example, the five applications that do not conform to any identified BPMPs may become conformant using different event log samples, since their contract Application Binary Interfaces (ABIs) define similar events to those in the event aliases list. Also, the ERC20 and ERC271 token contracts in the dataset that are implemented through inheritance of OpenZeppelin contracts emit Transfer events for mint and burn methods, making it not so obvious to distinguish them from other token transfer methods. However, improving the event logs to include additional decoded transaction data such as addresses of senders and receivers would lead to more accurate validation results for the Bloom and Digester patterns.

10 Conclusion

The blockchain is a promising cross-industry technology that is obstructed by several usability challenges from a software engineering perspective. In contrast to the studies conducted to address these challenges, this study targets the connection between analysis and design phases not addressed in the literature. It proposed a BPMPL derived from real-world BCApps through PM to support the analysis of functional requirements for BCApps.

The BPMPL consists of nine patterns organized according to two-dimensional classification. The first dimension is BPV-oriented: TOPs and SCOPs. TOPs consist of Token Circulation, Token Supply, and Token Authorization patterns. SCOPs consist of Smart Contract Security and Smart Contract Authorization patterns. The second dimension is BPV-agnostic. It consists of Transfer, Lifecycle, and Accessibility patterns. The description of each BPMP encapsulates a requirement and design pattern accompanied by a BP meta-model. The results proved the validity of the proposed BPMPL.

The implications of the proposed BPMPL were discussed in three facets: software engineering support, technology support, and literature support. Future work aims at addressing the discussed limitations of this study, with a focus on validating the BPMPL through developing new EVM- and non-EVM-compatible BCApps.

Acknowledgments. Fouzia Alzhrani wishes to gratefully acknowledge a scholarship she received from King Abdulaziz University that enabled her to pursue a Ph.D. degree at the University of Manchester.

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

Data availability statement. The dataset analyzed during the current study are available in the Zenodo repository [41].

References

[1] Ramias, A.J., Rummler, R.: The evolution of the effective process framework: A model for redesigning business processes. Performance Improvement **48**(10), 25–32 (2009). https://doi.org/10.1002/pfi.20112

[2] Morales-Sandoval, M., Molina, J.A., Marin-Castro, H.M., Gonzalez-Compean, J.L.: Blockchain support for execution, monitoring and discovery of inter-organizational business processes. PeerJ. Computer science **7**, 731–731 (2021). https://doi.org/10.7717/peerj-cs.731

[3] Di Ciccio, C., Meroni, G., Plebani, P.: On the adoption of blockchain for business process monitoring. Software and Systems Modeling **21**(3), 915–937 (2022). https://doi.org/10.1007/s10270-021-00959-x

[4] Vacca, A., Di Sorbo, A., Visaggio, C.A., Canfora, G.: A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges. Journal of Systems and Software **174**, 110891 (2021). https://doi.org/10.1016/j.jss.2020.110891

[5] Udokwu, C., Anyanka, H., Norta, A.: Evaluation of approaches for designing and developing decentralized applications on blockchain. In: Proceedings of the 2020 4th International Conference on Algorithms, Computing and Systems, pp. 55–62. Association for Computing Machinery. https://doi.org/10.1145/3423390.3426724

[6] Chakraborty, P., Shahriyar, R., Iqbal, A., Bosu, A.: Understanding the software development practices of blockchain projects: A survey. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM '18. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3239235.3240298

[7] Six, N., Herbaut, N., Salinesi, C.: Blockchain software patterns for the design of decentralized applications: A systematic literature review. Blockchain: Research and Applications **3**(2), 100061 (2022). https://doi.org/10.1016/j.bcra.2022.100061

[8] Farooq, M.S., Ahmed, M., Emran, M.: A survey on blockchain acquainted software requirements engineering: Model, opportunities, challenges, and future directions. IEEE Access **10**, 48193–48228 (2022). https://doi.org/10.1109/ACCESS.2022.3171408

[9] Bouraga, S., Burnay, C., Jureta, I., Faulkner, S.: Requirements elicitation for applications running on a blockchain: Preliminary results. In: Nurcan, S., Korthaus, A. (eds.) Intelligent Information Systems, pp. 38–46. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79108-7_5

[10] de la Vara, J.L., Sánchez, J.: Improving requirements analysis through business process modelling: A participative approach. In: Abramowicz, W., Fensel, D. (eds.) Business Information Systems, pp. 165–176. Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-79396-0_15

[11] Barros, O.: Business process patterns and frameworks. Business Process Management Journal **13**(1), 47–69 (2007). https://doi.org/10.1108/14637150710721122

[12] van der Aalst, W.M.P.: In: Jensen, K., van der Aalst, W.M.P. (eds.) Process-Aware Information Systems: Lessons to Be Learned from Process Mining, pp. 1–26. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00899-3_1

[13] Burattin, A.: Process Mining, pp. 33–47. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17482-2_5

[14] Aalst, W.v.d.: Process mining: Overview and opportunities. ACM Trans. Manage. Inf. Syst. **3**(2), 7 (2012). https://doi.org/10.1145/2229156.2229157

[15] Reinkemeyer, L.: In: Reinkemeyer, L. (ed.) Process Mining in a Nutshell, pp. 3–10. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40172-6_1

[16] Garcia, C.d.S., Meincheim, A., Faria Junior, E.R., Dallagassa, M.R., Sato, D.M.V., Carvalho, D.R., Santos, E.A.P., Scalabrin, E.E.: Process mining techniques and applications – a systematic mapping study. Expert Systems with Applications **133**, 260–295 (2019). https://doi.org/10.1016/j.eswa.2019.05.003

[17] van der Aalst, W., Adriansyah, A., de Medeiros, A.K.A., Arcieri, F., Baier, T., Blickle, T., Bose, J.C., van den Brand, P., Brandtjen, R., Buijs, J., Burattin, A., Carmona, J., Castellanos, M., Claes, J., Cook, J., Costantini, N., Curbera, F., Damiani, E., de Leoni, M., Delias, P., van Dongen, B.F., Dumas, M., Dustdar, S., Fahland, D., Ferreira, D.R., Gaaloul, W., van Geffen, F., Goel, S., Günther, C., Guzzo, A., Harmon,

P., ter Hofstede, A., Hoogland, J., Ingvaldsen, J.E., Kato, K., Kuhn, R., Kumar, A., La Rosa, M., Maggi, F., Malerba, D., Mans, R.S., Manuel, A., McCreesh, M., Mello, P., Mendling, J., Montali, M., Motahari-Nezhad, H.R., zur Muehlen, M., Munoz-Gama, J., Pontieri, L., Ribeiro, J., Rozinat, A., Seguel Pérez, H., Seguel Pérez, R., Sepúlveda, M., Sinur, J., Soffer, P., Song, M., Sperduti, A., Stilo, G., Stoel, C., Swenson, K., Talamo, M., Tan, W., Turner, C., Vanthienen, J., Varvaressos, G., Verbeek, E., Verdonk, M., Vigo, R., Wang, J., Weber, B., Weidlich, M., Weijters, T., Wen, L., Westergaard, M., Wynn, M.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) Business Process Management Workshops, pp. 169–194. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28108-2_19

[18] Xu, X., Weber, I., Staples, M.: Architecture for Blockchain Applications, 1st edn., p. 307. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-03035-3

[19] Alzhrani, F.E., Saeedi, K.A., Zhao, L.: A taxonomy for characterizing blockchain systems. IEEE Access **10**, 110568–110589 (2022). https://doi.org/10.1109/ACCESS.2022.3214837

[20] Ko, R.K.L.: A computer scientist's introductory guide to business process management (bpm). XRDS **15**(4), 4 (2009). https://doi.org/10.1145/1558897.1558901

[21] Burattin, A.: Introduction to Business Processes, BPM, and BPM Systems, pp. 11–21. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17482-2_2

[22] Adams, J., Koushik, S., Galambos, G., Vasudeva, G.: Patterns for E-business: A Strategy for Reuse, p. 334. Mc Press, Boise (2001)

[23] Zhao, L., Macaulay, L., Adams, J., Verschueren, P.: A pattern language for designing e-business architecture. Journal of Systems and Software **81**(8), 1272–1287 (2008). https://doi.org/10.1016/j.jss.2007.11.717

[24] Fellmann, M., Koschmider, A., Laue, R., Schoknecht, A., Vetter, A.: Business process model patterns: state-of-the-art, research classification and taxonomy. Business Process Management Journal **25**(5), 972–994 (2019). https://doi.org/10.1108/BPMJ-01-2018-0021

[25] Sommerville, I.: Software Engineering, 9th edn., p. 792. Pearson Education, USA (2011)

[26] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture A System of Patterns. Wiley Software Patterns Series, vol. 1, p. 476. Wiley, USA (1996)

[27] Appleton, B.: Patterns and software: Essential concepts and terminology. Object Magazine Online **3**(5), 20–25 (1997)

[28] Ampatzoglou, A., Kritikos, A., Kakarontzas, G., Stamelos, I.: An empirical investigation on the reusability of design patterns and software packages. Journal of Systems and Software **84**(12), 2265–2283 (2011). https://doi.org/10.1016/j.jss.2011.06.047

[29] Coplien, J.O.: Software design patterns: common questions and answers. The patterns handbook: Techniques, strategies, and applications **13**, 311 (1998)

[30] Lazarica, M.: The patterns development process for e-business applications. In: The Annals of University of Oradea - Economic Science, pp. 1402–1404. https://mpra.ub.uni-muenchen.de/15556/

[31] Schimm, G.: Process miner — a tool for mining process schemes from event-based data. In: Flesca, S., Greco, S., Ianni, G., Leone, N. (eds.) Logics in Artificial Intelligence, pp. 525–528. Springer, Berlin, Heidelberg (2002). https://doi.org/0.1007/3-540-45757-7_47

[32] Lu, Q., Xu, X., Bandara, H.M.N.D., Chen, S., Zhu, L.: Patterns for blockchain-based payment applications. In: 26th European Conference on Pattern Languages of Programs, p. 28. Association for Computing Machinery. https://doi.org/10.1145/3489449.3490006

[33] Yang, X.: Blockchain-based supply chain finance design pattern. In: 2021 13th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), pp. 200–203 (2021). https://doi.org/10.1109/IHMSC52134.2021.00053

[34] Mühlberger, R., Bachhofner, S., Castelló Ferrer, E., Di Ciccio, C., Weber, I., Wöhrer, M., Zdun, U.: Foundational oracle patterns: Connecting blockchain to the off-chain world. In: Asatiani, A., García, J.M., Helander, N., Jiménez-Ramírez, A., Koschmider, A., Mendling, J., Meroni, G., Reijers, H.A. (eds.) Business Process Management: Blockchain and Robotic Process Automation Forum, pp. 35–51. Springer, Cham. https://doi.org/10.1007/978-3-030-58779-6_3

[35] Six, N., Ribalta, C.N., Herbaut, N., Salinesi, C.: A blockchain-based pattern for confidential and pseudo-anonymous contract enforcement. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1965–1970. https://doi.org/10.1109/TrustCom50675.2020.00268

[36] Wöhrer, M., Zdun, U.: Design patterns for smart contracts in the ethereum ecosystem. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 1513–1520. https://doi.org/10.1109/Cybermatics_2018.2018.00255

[37] Wohrer, M., Zdun, U.: Smart contracts: security patterns in the ethereum ecosystem and solidity. In: 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE), pp. 2–8. https://doi.org/10.1109/IWBOSE.2018.8327565

[38] Xu, X., Pautasso, C., Zhu, L., Lu, Q., Weber, I.: A pattern collection for blockchain-based applications. In: Proceedings of the 23rd European Conference on Pattern Languages of Programs, p. 3. Association for Computing Machinery. https://doi.org/10.1145/3282308.3282312

[39] Bartoletti, M., Pompianu, L.: An empirical analysis of smart contracts: Platforms, applications, and design patterns. In: Brenner, M., Rohloff, K., Bonneau, J., Miller, A., Ryan, P.Y.A., Teague, V., Bracciali, A., Sala, M., Pintore, F., Jakobsson, M. (eds.) Financial Cryptography and Data Security, pp. 494–509. Springer. https://doi.org/10.1007/978-3-319-70278-0_31

[40] Müller, M., Ostern, N., Rosemann, M.: Silver bullet for all trust issues? blockchain-based trust patterns for collaborative business processes. Business Process Management: Blockchain and Robotic Process Automation Forum, pp. 3–18. Springer. https://doi.org/10.1007/978-3-030-58779-6_1

[41] Alzhrani, F.: A Collection of Event Logs of Blockchain-based Applications. Zenodo (2022). https://doi.org/10.5281/zenodo.6637059

[42] Presley, A., Liles, D.H.: The use of idef0 for the design and specification of methodologies. In: Proceedings of the 4th Industrial Engineering Research Conference, pp. 442–448 (1998)

[43] Meszaros, G., Doble, J.: Metapatterns: A pattern language for pattern writing. In: The 3rd Pattern Languages of Programming Conference, Monticello, Illinois (1996)

[44] Riehle, D., Züllighoven, H.: Understanding and using patterns in software development. Theory and Practice of Object Systems **2**(1), 3–13 (1996). https://doi.org/10.1002/(SICI)1096-9942(1996)2:1⟨3::AID-TAPO1⟩3.0.CO;2-%23

[45] Withall, S.: Software Requirement Patterns. Microsoft Press, USA (2007)

[46] Davis, R., Brabänder, E. (eds.): The Event-driven Process Chain, pp. 105–125. Springer, London (2007). https://doi.org/10.1007/978-1-84628-613-1_7

[47] Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Report, Ethereum (2020). https://ethereum.github.io/yellowpaper/paper.pdf?

[48] Rahimian, R., Eskandari, S., Clark, J.: Resolving the multiple withdrawal attack on erc20 tokens. In: 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pp. 320–329 (2019). https://doi.org/10.1109/EuroSPW.2019.00042

[49] Sánchez-Gómez, N., Torres-Valderrama, J., García-García, J.A., Gutiérrez, J.J., Escalona, M.J.: Model-based software design and testing in blockchain smart contracts: A systematic literature review. IEEE Access **8**, 164556–164569 (2020). https://doi.org/10.1109/ACCESS.2020.3021502

[50] Zhang, Y., Patel, S.: Agile model-driven development in practice. IEEE Software **28**(2), 84–91 (2011). https://doi.org/10.1109/MS.2010.85

[51] Sujeetha, R., Preetha, C.A.S.D.: A literature survey on smart contract testing and analysis for smart contract based blockchain application development. In: 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC), pp. 378–385. https://doi.org/10.1109/ICOSEC51865.2021.9591750

[52] Sarwar, T., Habib, W., Arif, F.: Requirements based testing of software. In: 2013 Second International Conference on Informatics & Applications (ICIA), pp. 347–352. https://doi.org/10.1109/ICoIA.2013.6650281

[53] Mirarab, S., Ganjali, A., Tahvildari, L., Li, S., Liu, W., Morrissey, M.: A requirement-based software testing framework: An industrial practice. In: 2008 IEEE International Conference on Software Maintenance, pp. 452–455. https://doi.org/10.1109/ICSM.2008.4658102

[54] Chain List: Helping users connect to EVM powered networks (2022). https://chainlist.org/

Appendix A

A.1 Event Aliases List

The list of event aliases is provided in Table A1.

A.2 Event-driven Process Chain Notation

EPC is a business process modeling method that illustrates the relationship between events, activities, organization roles, and information resources. A more complex business workflow can be described using logical operators or rules. The elements of an EPC diagram are summarized in Fig. A1.

A.3 Known Uses in Blockchain Applications

The blockchain applications from the validation dataset that are used in the descriptions of the BPMPs are introduced in Table A2.

A.4 Modeling of Pass Management Application

Figures A2 to A4 show the modeling of the example use case, the Pass Management Application, as a result of applying the proposed BPMPL.

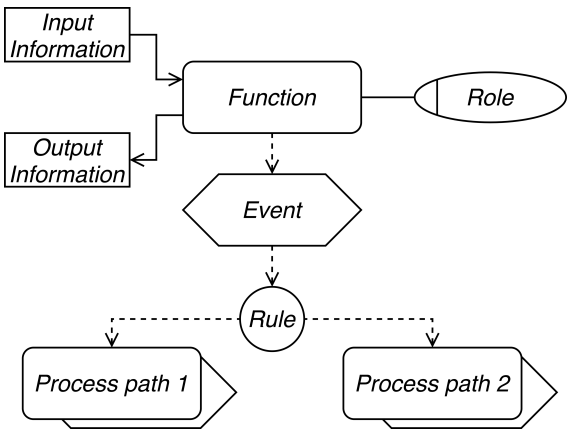

Fig. A1 Elements of Event-driven Process Chain. The text inside an element identifies the element. The connectors between the elements indicate a control flow (↓), information flow (←→), or role allocation (—). A process path is a connection to another process.

Table A1 The list of event aliases used in conformance checking analysis

Alias	Events
Transferred	TokensTransferred, TransferToParent, TransferFromParent, LogOwnerTransfer, NewTokenGrant, PaymentTransferredToPreviousOwner, PO8Bought, TokenPurchase, LandPurchased, PlotSectionSold, PlotPurchased, TokenSold, TokensUndelegated, TokensDelegated, ClaimedTokens, HorseTransferredIn, HorseTransferredOut, Transfer, LOG_SuccessfulSend
OwnershipTransferred	ProxyOwnershipTransferred, LogOwnerShipTransferred, LogOwnerShipTransferInitiated, UpdaterTransferred, GDPOracleTransferred, FeeWalletTransferred, OwnershipChanged, OwnershipTransferred
Approved	ApprovalForAll, Approval, Approve
Paused	Paused, LoggPaused, ContractIsPaused, Pause, Stopped, LOG_ContractStopped, LOG_EmergencyAutoStop
RoleAllocated	RoleRemoved, MinterRemoved, AuthorizedAddressRemoved, RemoveAllowed, WhitelistAdminRemoved, PauserRemoved, AdminRemoved, RoleAdded, PlayerAdded, MinterAdded, AuthorizedAddressAdded, PauserAdded, AdminAdded, WhitelistAdminAdded, AdminUpdated, AdminChanged, ProxyAdminChanged, ManagementChanged, Allowed, RemoveAllowed, MintingAgentChanged
Unpaused	UnPaused, LogUnpaused, Unpause, LOG_ContractResumed, Started
Upgraded	UpgradeAgentSet, Upgraded, ContractUpgrade, , M5TokenUpgrade, M5LogicUpgrade, FinishUpgrade, UpgradeProposal
Minted	Minted, TokensMinted, Minted, CrystalMinted, CardMinted, TokenCreated, Birth, Mint
Withdrawn	_Withdrawal, Withdrawal, LogWithdrawal, playerWithdrawal, LogEtherWithdrawn, LogWithdraw, _ListingWithdrawn, _VotingRightsWithdrawn, Withdrawn, PaymentWithdrawn, PaymentWithdrawnByDispute, LogOperationFeeWithdraw, WithdrawM5, Withdraw
OwnershipRenounced	OwnershipRenounced
Deposited	LogDeposit, DividendsDeposited, _Deposit, LogEtherDeposited, playerDeposit, ReferralDeposit, PaymentDeposited, Deposit
Burned	TokensBurned, CrystalBurned, KittyBurned, Burnt, Burned, Burn

Table A2 An overview of known uses in blockchain applications cited in patterns' descriptions

Application	Description	Domain	Application Token Symbol	Token Type
CryptoCrystal https://cryptocrystal.io	A crystal collectible game that allows users to mine, collect, and trade virtual crystals	Game	CC, PKX	Non-fungible, Fungible
Landemic https://landemic.io/	A property collectibles game that allows users to buy, sell and claim ownership interest in virtual lands	Game	LAND	Non-fungible
FOAM https://map.foam.space/	A consensus-driven map of the world that enables crowdsourced verification of points of interest and location claims.	Real Estate	FOAM	Fungible
Upfiring https://www.upfiring.com	A P2P file-sharing desktop application incentivizes users for seeding files	Media	UFR	Fungible
TimeX https://timex.io/	A decentralized cryptocurrency exchange	Marketplace	TIME	Fungible
CryptoStrikers http://www.cryptostrikers.com	A card collectible game that allows users to buy, sell, and trade rare sports cards	Game	STRK	Non-fungible
POA Bridge https://bridge.poa.net	A bridge application that enables transferring native tokens from the POA Network to the Ethereum network	Finance	POA, POA20	Fungible
Nestree https://www.nestree.io	An Android and iOS reward-based blockchain integrated messenger. Users earn tokens for activities such as daily app check-ins, playing games, and answering surveys	Social	EGG	Fungible
MyCryptons https://mycryptons.com	A digital collectibles game that combines political expression and social media interaction with games	Game	CRYPTON	Non-fungible
Zinc https://zinc.work/	A reputation system that provides an authenticated proof of career experience and qualifications	Information Technology	ZINC	Fungible

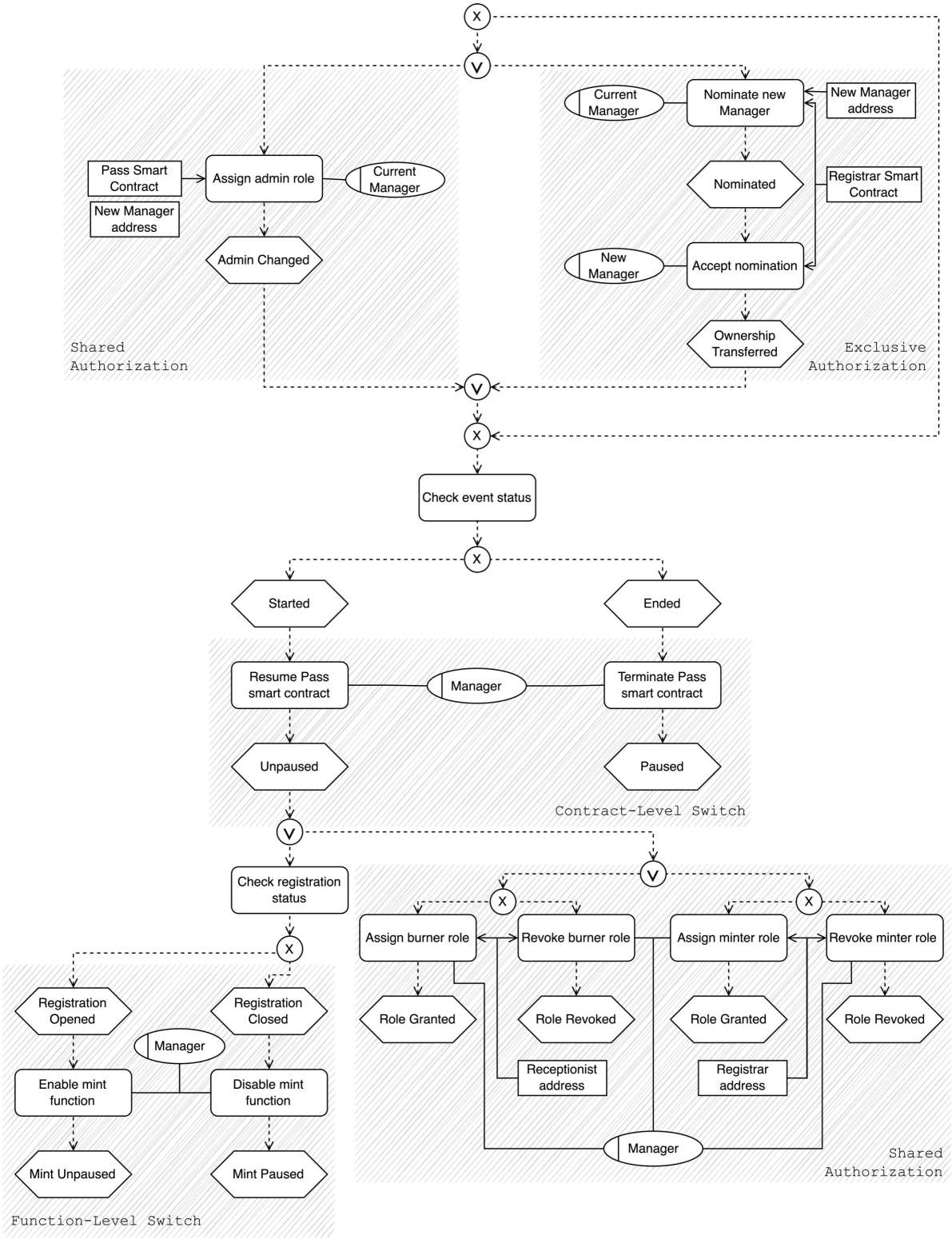

Fig. A2 Smart Contract Administration process of Pass Management Application.

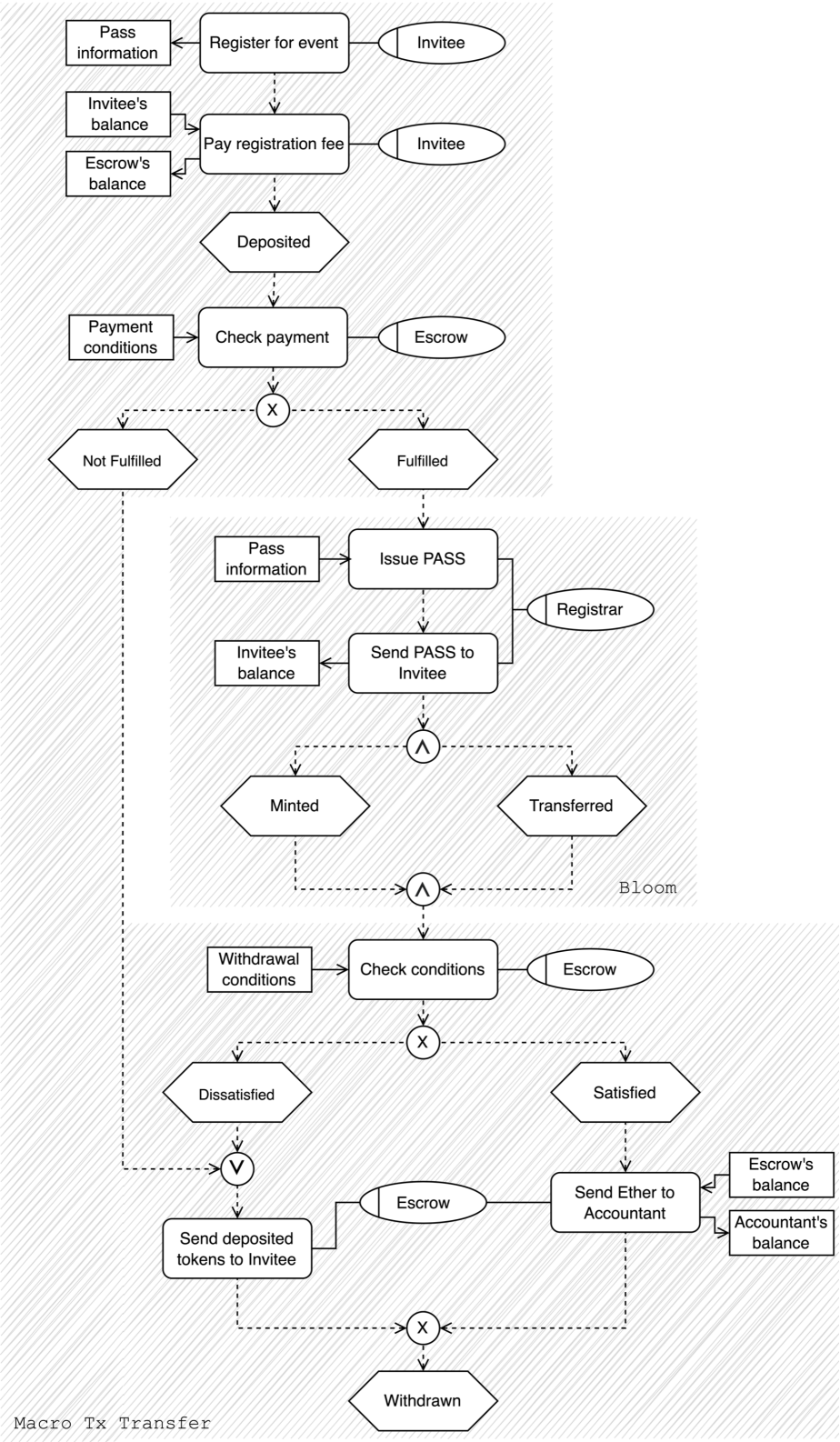

Fig. A3 PASS Token Issuance process of Pass Management Application.

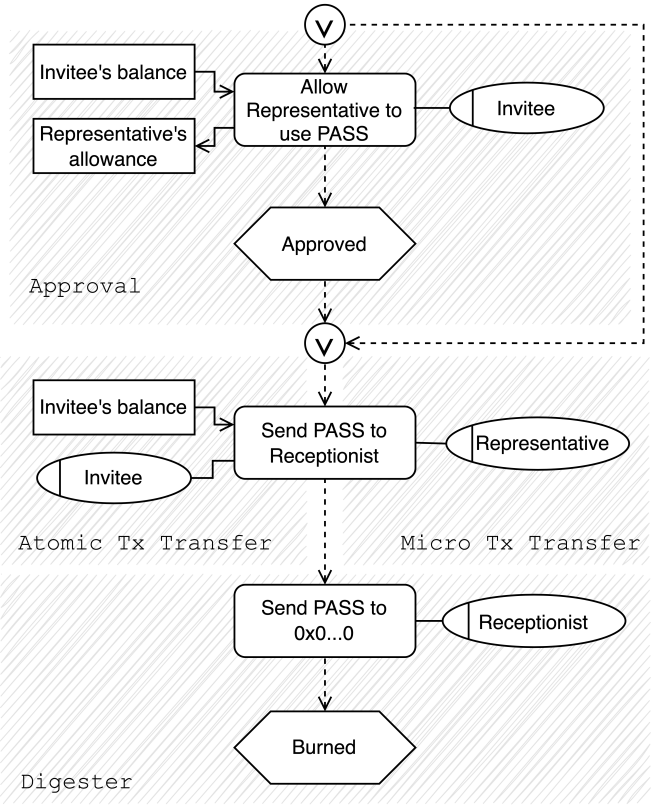

Fig. A4 Event Check-In process of Pass Management Application.

Chapter 7

Conclusion

This thesis set out to gain an understanding of blockchain applications from architectural and business process perspectives. It defined five research objectives as goals for this understanding. The thesis described how these objectives have been achieved in Chapters 2 to 6, and what research contributions have been obtained by achieving these objectives. This chapter concludes the thesis by summarizing the main research contributions, discussing the limitations, and highlighting future work.

7.1 Contributions and Key Findings

In what follows, I summarize the contributions during my research, map them onto the thesis objectives that are listed in Chapter 1, and highlight key findings from each contribution.

7.1.1 Development of a Blockchain Applications Catalog

This thesis adopted a multi-view concept for data catalogs through building one that joins up different, yet relevant, software-, process-, and application-centric data about blockchain applications. There are three relevant datasets produced in this thesis: Industry-developed applications, event logs for these applications, and Academia-researched applications. The analysis of these datasets helped in producing the other contributions in this thesis. This catalog fulfills the first research objective and was described in Chapter 2.

One of the main findings from this contribution is that blockchain technology and its applications are gaining significant interest in the community, however; there is a gap in application trends between the literature and industry. The most discussed use cases in the literature are related to healthcare, finance, energy, supply chain, and Internet of Things applications. On the other hand, the most Industry-developed applications are related to P2P exchange, game, social, and media applications. Based on the three application categories, identified in Chapter 4, we can conclude that the literature focuses on Enterprise-Centric and IoT-Based applications whereas the industry focuses on Consumer-Centric applications.

Besides, although blockchain systems can operate stand-alone, in real business, there is a need to communicate with other non-blockchain systems. This need imposes two general

uses of blockchains. First, blockchains can be used as a data store in the form of an immutable, append-only, consensus-based distributed ledger. For example, MediaChain[1] is an open universal blockchain-based media library that can be used as a data storage for relevant media systems. Second, blockchains can be used as a transaction platform to automate business processes and transactions in the form of decentralized applications. Actifit[2] is an example of a decentralized application which aims at incentivizing healthy lifestyle to provide an incentive for people to get active, healthy and fit.

7.1.2 Proposal of a Taxonomy for Characterizing Blockchain Systems

This thesis classified blockchain system characteristics into a holistic component-based taxonomy. The taxonomy synthesizes the features and aspects of eight fundamental components of blockchain systems into three main dimensions. The first dimension describes the components of the Execution Environment Subsystem: *network*, *distributed ledger*, and *platform*. The second dimension describes the components of the Internal Subsystem: *consensus protocol*, *smart contract*, and *token*. The last dimension describes the components of the External Subsystem: *nodes* and *digital wallets*. This contribution fulfills the second research objective and was described in Chapter 3.

Chapter 3 pointed out several observations related to the development of blockchain applications, including the need for improved incentive mechanisms. It highlighted three expertise areas needed for developing blockchain applications: infrastructure settings, smart contract development, and front-end development. It, also, pointed out a set of design decisions related to the licensing, deployment, access control, and membership of blockchain applications.

In addition to the aforementioned findings, we find that the proposed taxonomy would increase the adoption of blockchain systems. It provides a shared language for describing, analyzing, classifying, designing, and documenting blockchain systems. The taxonomy provides a better understanding of the interrelationships between different blockchain components, and thus; it assists in articulating a variety of design choices of blockchain systems and their components. Additionally, it would assist towards a homogeneous development of blockchains and blockchain-based systems. It can be used to create robust frameworks to enable precise comparisons of different blockchains.

Another notable finding is related to the method of developing the taxonomy. Merging software decomposition and Nickerson et al.'s approaches in this thesis produced a novel method for taxonomy development in the context of software engineering research. It is a reuse-based taxonomy development approach that inherits the concept of decomposition by breaking down a complex knowledge domain under classification (i.e., blockchain system) into smaller, manageable, and well-defined subject units (i.e., software and hardware components) based on a meta-characteristic (i.e., execution point). This developed method allows

[1] http://www.mediachain.io
[2] https://actifit.io

to systematically and incrementally build a modular taxonomy that consists of a set of relevant, yet independent, sub-taxonomies.

7.1.3 Formulation of an Architectural Pattern Language for Blockchain Application Development

This thesis proposed an APL to support the development of blockchain applications. The APL contains 12 architectural patterns to support fulfilling NFRs and quality requirements during the architecture design of blockchain applications. The patterns are: Distributed Client-Server, On-Chain/Off-Chain, Distributed Backend, P2P On-Chain Backend, Centralized Off-Chain Backend, Distributed Off-Chain Backend, Blockchain Broker, Layered Client Application, On-Chain Pipe-Filter, Replicated Repository, Chain-of-Blocks, and Chain Fork. This pattern language can assist detecting quality issues in architecture reviews of blockchain applications. It fulfills the third research objective and was described in Chapter 4.

One of the main findings from this contribution is that blockchain platforms are evolving mostly to overcome protocol-related issues such as scalability and performance. For example, POA is a public side chain that enables cross-chain-bridging architecture within the Ethereum ecosystem. Moreover, Hive had been forked from the Steem blockchain to offer more decentralization and uncontrolled token ownership.

In another respect, considering reusable architectural patterns can reduce the number of concerns that blockchain application architects and developers need to think of. The APL provides blueprints to help the architects and developers in design decision-making and fulfilling application-specific NFRs and quality requirements. For example, the *Interoperability* of blockchain applications can be fulfilled with the Blockchain Broker pattern and the *Reliability* can be accomplished with the *Distributed Off-Chain Backend* pattern, coupled with the *P2P On-Chain Backend*. Certainly, there should be a trade-off between the different quality requirements when designing an application according to the use case needs.

7.1.4 Development of a Process-Aware Framework to Support Process Mining from Blockchain Applications

This thesis proposed a framework to support both, identifying process-aware blockchain applications and generating event logs for these applications. It addressed several challenges for mining business processes from blockchains in the context of EVM-compatible applications. The framework has two modules: *PAR* and *ELG*. PAR is a human-in-the-loop model that allows cooperation between users (*PAR-H*) and the automation algorithm (*PAR-M*) to recognize process-aware blockchain applications. ELG is an automation model to generate event logs from blockchain event data via three interdependent algorithms: *Extractor*, *Decoder*, and *Formatter*.

The design of this framework was instantiated by implementing *BELA*, a reusable software prototype. This software application is a stand-alone contribution that implements the PARM, Extractor, Decoder, and Formatter algorithms. These two contributions fulfill the fourth objective, and were described in Chapter 5.

One of the main observations from the development of this framework is that there are two approaches to extract event data from blockchain networks: directly and indirectly. The direct approach involves extracting event data via sending RPCs to the network and receiving the event data as a response. This can be done either through block explorers or through programmed RPCs by independent developers. For example, Etherscan[3] is the official block explorer and analytics platform for Ethereum. The indirect approach involves retrieving event data through querying state databases such as BigQuery[4], an enterprise data warehouse that enables SQL queries to retrieve log data from its log dataset.

The proposed ELG module is an example of the direct approach through programmed RPCs. It is worth mentioning that this thesis is not the first to support the extraction of event data with a tool and provide a set of formatted event logs. In contrast to existing literature, the user of our suggested algorithm has not to write a file where they describe the extraction requirements. Thus, the user bears the risk that events are not captured in the right way or wrong events.

Another observation is that blockchain-based business processes are more asset-centric than activity-centric. From one standpoint, tokenization and smart contracts allow for tracking the existence and ownership of digital and physical assets. These assets can go through several transformations in their lifecycles and changes in their ownership, such as in supply chains. On the other hand, event logs are centered around the activities happened as part of such processes, which is inadequate to track assets on blockchains that do not support programmable smart contracts, and thus, these blockchains are not activity-centric. Accordingly, it is notable that there is a need for blockchain-oriented business process management (BPM) tools, modeling notations, and event log standards. PM techniques and tools have to be adapted to allow for a more holistic analysis of blockchain applications.

7.1.5 Proposal of a Business Process Modeling Pattern Language for Blockchain Application Requirement Analysis

This thesis proposed a BPMPL to support the development of blockchain applications. The BPMPL consists of nine business process modeling patterns to support fulfilling FRs during the requirement analysis of blockchain applications. The patterns are: Atomic Tx, Micro Tx, Macro Tx, Bloom, Digester, Approval, Switch, Exclusive Authorization, and Shared Authorization. This contribution fulfills the fifth objective and was described in Chapter 6.

Several findings from this contribution were discussed in Chapter 6, including the BPMPL

[3] https://etherscan.io
[4] https://cloud.google.com/bigquery

support for different software engineering practices, such as agile model-driven development, requirement modeling and documentation, and blockchain application testing. Also, it pointed out how the proposed BPMPL supports the design of interoperable blockchain applications.

An additional finding from this contribution is that adapting PM to identify recurrent across-domain business processes is a novel approach that offers a systematic way for identifying software patterns in the context of software engineering. Therefore, PM is effective in reverse engineering FRs from existing blockchain applications, and arguably, from non-blockchain software applications.

Another remarkable finding is that PM significantly reduces the effort required to recover business processes embedded within smart contracts compared to manual software reverse engineering. Generating an event log using BELA, with $RPCCount = 5$, took an average time of 28 seconds per contract address. When ingesting the event log into the PM tool, the business model is immediately recovered.

7.2 Research Limitations and Future Work

The contributions of this thesis need to be considered in light of its limitations. In what follows, I discuss these limitations and future work.

7.2.1 Dataset Validity

Although we tried our best to *systematically* select a sufficiently comprehensive and relevant set of blockchain application software and publications, the manual screening process to filter thousands of them could have some valuable items been excluded. Moreover, blockchain technology is rapidly evolving, making some of the software applications inaccessible and outdated from a technological viewpoint, which may hinder research reproducibility. However, data about these applications are enduring. Data produced by these applications are stored in blockchains, and thus, they are permanent and can be retrieved from corresponding networks.

Furthermore, this work provides more extracted event logs than in the literature, but these logs only include a maximum of the last 5000 activities. Thus, some of them do not include all events that are defined in the subject smart contract. For example, in Augur event log, only the event ”Universe created” and ”Token minted” are included. Events about disputes and market creation and closing are missing. This limitation can be overcome by running BELA with custom configurations to extract more events. The *fromBlock* can be set to the last block number in the current event log and the *RPCCount* can be assigned a desired integer value, as described in Chapter 5.

7.2.2 Contribution Boundary

Some of the contributions have a narrow application scope. For example, the PAF and BELA target applications that are compatible with EVM. There is a need to generalize these solutions in future work to solve similar problems of mining business processes from other non-EVM-compatible applications. Moreover, the BPMPL considered only smart contract functionality, which have been derived from and tested on EVM-compatible applications. As there are non-EVM-compatible blockchains that use smart contracts, e.g., Hyperledger Fabric, the BPMPL needs to be validated on these blockchains in our future work. On the other hand, the taxonomy and APL are blockchain-agnostic; however, they focus on blockchain-distinct components and overlook other components that may influence architectural choices.

7.2.3 Contribution Viewpoints

Although the demonstrated results throughout this thesis showed the soundness of the contributions, this thesis does not claim that the contributions are complete. For example, we found that the applications we studied *broadly* fit into one of the three identified application categories in Chapter 4: *Consumer-Centric*, *Enterprise-Centric*, and *IoT-Based*. We have not identified other applications, as we think that the coarseness of our three categories has allowed us to align the applications to one of these categories. We believe that had our categories been more fine-grained, there would be outliers that cannot fit into exactly a category. This will be considered in our future work.

Moreover, distinguishing components and aspects when building the taxonomy is arguable. For example, while it is possible to consider ”incentive mechanism” and ”permission management” as *components*, the adapted viewpoint in this thesis to identify the blockchain system components considered them *Aspects* of the *Platform* component within the *Execution Environment Subsystem*. Adapting a different viewpoint for the system decomposition would lead to a different hierarchy of components and their interrelationships, which could be examined in future work.

In addition, the APL currently does not cover some important concerns of blockchain applications. The security concern is currently addressed in this pattern language as a fundamental layer in the “Layered client application” pattern, but has not been explicitly considered at the pattern level. Our future work will focus on the security patterns. Also, some patterns are general but fundamental to structure the blockchain applications, and they are described by other patterns.

7.2.4 Technology Constraints

Current BPM and PM tools do not support the analysis of blockchain applications from an asset-centric viewpoint. The process-awareness of blockchain applications was discussed in

this thesis from an activity-centric viewpoint, which maybe insufficient. Consequently, the modeling of the proposed BPMPL is influenced by the activity-centric viewpoint to model the business processes of the patterns. In the future, if blockchain-oriented BPM tools are developed, we will have the chance to do a more appropriate analysis.

Additionally, the evaluation sample used to demonstrate the proposed taxonomy does not cover some possible variations of the characteristics. For example, the token collection does not have non-transferable tokens, although this type has been identified in the Simple Asset Standard [5]. Paper wallets are handmade by their owner and secured offline, making them unreachable for evaluation.

7.2.5 Evaluation Context

Although this thesis has demonstrated the applicability and usability of the proposed contributions through diverse real-world applications, the industry was not involved. Blockchain solution architects and developers are invited to evaluate the proposed contributions in designing new applications. The benefits of such practice are two folds. First, the contributions will be validated by independent developers that are not involved in the development of the contributions. Second, the contributions will be evaluated in the context of forward software engineering, instead of the reverse engineering context that is adopted in this thesis.

In the future, we intend to provide the contributions of this thesis as a service, making it accessible by blockchain researchers and practitioners. Thus, the artifacts can be reused, enhanced, and standardized. Moreover, the performance of PAF was not evaluated on algorithm-level; as it is out of the scope of this thesis. In our future work, we will conduct a comparative assessment of the existing platforms for event-log extraction from blockchains, highlighting the capabilities of BELA that other tools are not equipped with, with a demonstration based on experimental data.

[5] https://github.com/CryptoLions/SimpleAssets

References

[1] X. Xu, I. Weber, and M. Staples, *Architecture for blockchain applications*. Springer, 2019, pp. XXII, 307. DOI: 10.1007/978-3-030-03035-3.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21 260, 2008.

[3] F. Casino, T. K. Dasaklis, and C. Patsakis, "A systematic literature review of blockchain-based applications: Current status, classification and open issues," *Telematics and Informatics*, vol. 36, pp. 55–81, 2019, ISSN: 0736-5853. DOI: https://doi.org/10.1016/j.tele.2018.11.006.

[4] U. Bodkhe, S. Tanwar, K. Parekh, *et al.*, "Blockchain for industry 4.0: A comprehensive review," *IEEE Access*, vol. 8, pp. 79 764–79 800, 2020. DOI: 10.1109/ACCESS.2020.2988579.

[5] P. K. Sharma and J. H. Park, "Blockchain based hybrid network architecture for the smart city," *Future Generation Computer Systems*, vol. 86, pp. 650–655, 2018, ISSN: 0167-739X. DOI: 10.1016/j.future.2018.04.060.

[6] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng, and V. C. M. Leung, "Decentralized applications: The blockchain-empowered software system," *IEEE Access*, vol. 6, pp. 53 019–53 033, 2018. DOI: 10.1109/ACCESS.2018.2870644.

[7] A. Alammary, S. Alhazmi, M. Almasri, and S. Gillani, "Blockchain-based applications in education: A systematic review," *Applied Sciences*, vol. 9, no. 12, 2019, ISSN: 2076-3417. DOI: 10.3390/app9122400.

[8] A. Tandon, A. Dhir, A. N. Islam, and M. Mäntymäki, "Blockchain in healthcare: A systematic literature review, synthesizing framework and future research agenda," *Computers in Industry*, vol. 122, p. 103 290, 2020, ISSN: 0166-3615. DOI: 10.1016/j.compind.2020.103290.

[9] A. Khatoon, P. Verma, J. Southernwood, B. Massey, and P. Corcoran, "Blockchain in energy efficiency: Potential applications and benefits," *Energies*, vol. 12, no. 17, 2019, ISSN: 1996-1073. DOI: 10.3390/en12173317.

[10] M. Pournader, Y. Shi, S. Seuring, and S. L. Koh, "Blockchain applications in supply chains, transport and logistics: A systematic review of the literature," *International Journal of Production Research*, vol. 58, no. 7, pp. 2063–2081, 2020. DOI: 10.1080/00207543.2019.1650976.

[11] J. Mendling, I. Weber, W. V. D. Aalst, *et al.*, "Blockchains for business process management-challenges and opportunities," *ACM Transactions on Management Information Systems (TMIS)*, vol. 9, no. 1, p. 4, 2018, ISSN: 2158-656X. DOI: 10.1145/3183367.

[12] J. S. Arun, J. Cuomo, and N. Gaur, *Blockchain for Business*. Addison-Wesley Professional, 2019, ISBN: 0135581400.

[13] W. Viriyasitavat and D. Hoonsopon, "Blockchain characteristics and consensus in modern business processes," *Journal of Industrial Information Integration*, vol. 13, pp. 32–39, 2019, ISSN: 2452-414X. DOI: 10.1016/j.jii.2018.07.004.

[14] M. Morales-Sandoval, J. A. Molina, H. M. Marin-Castro, and J. L. Gonzalez-Compean, "Blockchain support for execution, monitoring and discovery of inter-organizational business processes," *PeerJ Computer Science*, vol. 7, e731, 2021.

[15] C. Di Ciccio, G. Meroni, and P. Plebani, "On the adoption of blockchain for business process monitoring," *Softw. Syst. Model.*, vol. 21, no. 3, pp. 915–937, Jun. 2022, ISSN: 1619-1366. DOI: 10.1007/s10270-021-00959-x.

[16] V. A. d. Sousa and B. Corentin, "Towards an integrated methodology for the development of blockchain-based solutions supporting cross-organizational processes," in *2019 13th International Conference on Research Challenges in Information Science (RCIS)*, 2019, pp. 1–6, ISBN: 978-1-7281-4844-1. DOI: 10.1109/RCIS.2019.8877045.

[17] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International journal of web and grid services*, vol. 14, no. 4, pp. 352–375, 2018.

[18] A. Vacca, A. Di Sorbo, C. A. Visaggio, and G. Canfora, "A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges," *Journal of Systems and Software*, vol. 174, p. 110 891, 2021, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2020.110891.

[19] P. Tasca and C. J. Tessone, "A taxonomy of blockchain technologies: Principles of identification and classification," *Ledger*, vol. 4, Feb. 2019. DOI: 10.5195/ledger.2019.140.

[20] L. Oliveira, L. Zavolokina, I. Bauer, and G. Schwabe, "To token or not to token: Tools for understanding blockchain tokens," in *International Conference of Information Systems (ICIS 2018)*, 2018. DOI: 10.5167/uzh-157908.

[21] S. Wieninger, G. Schuh, and V. Fischer, "Development of a blockchain taxonomy," in *2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, 2019, pp. 1–9. DOI: 10.1109/ICE.2019.8792659.

[22] X. Xu, I. Weber, M. Staples, *et al.*, "A taxonomy of blockchain-based systems for architecture design," in *2017 IEEE International Conference on Software Architecture (ICSA)*, 2017, pp. 243–252. DOI: 10.1109/ICSA.2017.33.

[23] T. Schulze, S. Seebacher, and F. Hunke, "Conceptualizing the role of blockchain technology in digital platform business," in *Exploring Service Science*, H. Nóvoa, M. Drăgoicea, and N. Kühl, Eds., Cham: Springer International Publishing, 2020, pp. 150–163, ISBN: 978-3-030-38724-2.

[24] E. Bellini, Y. Iraqi, and E. Damiani, "Blockchain-based distributed trust and reputation management systems: A survey," *IEEE Access*, vol. 8, pp. 21 127–21 151, 2020. DOI: 10.1109/ACCESS.2020.2969820.

[25] S. M. Sarkintudu, H. H. Ibrahim, and A. B. Abdwahab, "Taxonomy development of blockchain platforms: Information systems perspectives," *AIP Conference Proceedings*, vol. 2016, no. 1, p. 020 130, 2018. DOI: 10.1063/1.5055532.

[26] S. Tönnissen and F. Teuteberg, "Towards a taxonomy for smart contracts," in *26th European Conference on Information Systems (ECIS 2018)*, vol. 12, 2018. [Online]. Available: https://aisel.aisnet.org/ecis2018_rp/12.

[27] T. Ankenbrand, D. Bieri, R. Cortivo, J. Hoehener, and T. Hardjono, "Proposal for a comprehensive (crypto) asset taxonomy," in *2020 Crypto Valley Conference on Blockchain Technology (CVCBT)*, 2020, pp. 16–26. DOI: 10.1109/CVCBT50464.2020.00006.

[28] H. Arslanian and F. Fischer, "A high-level taxonomy of crypto-assets," in *The Future of Finance: The Impact of FinTech, AI, and Crypto on Financial Services*. Cham: Springer International Publishing, 2019, pp. 139–156, ISBN: 978-3-030-14533-0. DOI: 10.1007/978-3-030-14533-0_12.

[29] A. A. Monrat, O. Schelén, and K. Andersson, "A survey of blockchain from the perspectives of applications, challenges, and opportunities," *IEEE Access*, vol. 7, pp. 117 134–117 151, 2019. DOI: 10.1109/ACCESS.2019.2936094.

[30] D. Garlan, "Software architecture: A roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 91–101.

[31] T. A. Syed, A. Alzahrani, S. Jan, M. S. Siddiqui, A. Nadeem, and T. Alghamdi, "A comparative analysis of blockchain architecture and its applications: Problems and recommendations," *IEEE access*, vol. 7, pp. 176 838–176 869, 2019.

[32] M. S. Farooq, M. Ahmed, and M. Emran, "A survey on blockchain acquainted software requirements engineering: Model, opportunities, challenges, and future directions," *IEEE Access*, vol. 10, pp. 48 193–48 228, 2022. DOI: 10.1109/ACCESS.2022.3171408.

[33] C. Udokwu, H. Anyanka, and A. Norta, "Evaluation of approaches for designing and developing decentralized applications on blockchain," in *Proceedings of the 4th International Conference on Algorithms, Computing and Systems*, ser. ICACS '20, Rabat, Morocco: Association for Computing Machinery, 2020, pp. 55–62, ISBN: 9781450377324. DOI: 10.1145/3423390.3426724.

[34] P. Chakraborty, R. Shahriyar, A. Iqbal, and A. Bosu, "Understanding the software development practices of blockchain projects: A survey," in *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '18, Oulu, Finland: Association for Computing Machinery, 2018, ISBN: 9781450358231. DOI: 10.1145/3239235.3240298.

[35] N. Six, N. Herbaut, and C. Salinesi, "Blockchain software patterns for the design of decentralized applications: A systematic literature review," *Blockchain: Research and Applications*, vol. 3, no. 2, p. 100 061, 2022, ISSN: 2096-7209. DOI: 10.1016/j.bcra.2022.100061.

[36] S. Bouraga, C. Burnay, I. Jureta, and S. Faulkner, "Requirements elicitation for applications running on a blockchain: Preliminary results," in *Intelligent Information Systems*, S. Nurcan and A. Korthaus, Eds., Cham: Springer International Publishing, 2021, pp. 38–46, ISBN: 978-3-030-79108-7.

[37] A. Burattin, "User-guided discovery of process models," in *Process Mining Techniques in Business Environments: Theoretical Aspects, Algorithms, Techniques and Open Challenges in Process Mining*. Cham: Springer International Publishing, 2015, pp. 113–118, ISBN: 978-3-319-17482-2. DOI: 10.1007/978-3-319-17482-2_13.

[38] N. Y. Wirawan, B. N. Yahya, and H. Bae, "Incorporating transaction lifecycle information in blockchain process discovery," in *Blockchain Technology for IoT Applications*, S.-W. Lee, I. Singh, and M. Mohammadian, Eds. Singapore: Springer Singapore, 2021, pp. 155–172, ISBN: 978-981-33-4122-7. DOI: 10.1007/978-981-33-4122-7_8.

[39] R. Mühlberger, S. Bachhofner, C. Di Ciccio, L. García-Bañuelos, and O. López-Pintado, "Extracting event logs for process mining from data stored on the blockchain," in *International Conference on Business Process Management*, ser. Business Process Management Workshops, Springer International Publishing, 2019, pp. 690–703, ISBN: 978-3-030-37453-2. DOI: 10.1007/978-3-030-37453-2_55.

[40] F. Alzhrani, *A collection of industry-developed blockchain-based applications*, version 1.0, Zenodo, Nov. 2020. DOI: 10.5281/zenodo.4268953. [Online]. Available: https://doi.org/10.5281/zenodo.4268953.

[41] F. Alzhrani, *A Collection of Event Logs of Blockchain-based Applications*, version 1.0, Zenodo, Jun. 2022. DOI: 10.5281/zenodo.6637059. [Online]. Available: https://doi.org/10.5281/zenodo.6637059.

[42] F. Alzhrani, *A Collection of Papers on Blockchain Business Applications*, Zenodo, Jan. 2023. DOI: 10.5281/zenodo.7564584. [Online]. Available: https://doi.org/10.5281/zenodo.7564584.

[43] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '14, London, England, United Kingdom: Association for Computing Machinery, 2014, ISBN: 9781450324762. DOI: 10.1145/2601248.2601268.

[44] F. E. Alzhrani, K. A. Saeedi, and L. Zhao, "A taxonomy for characterizing blockchain systems," *IEEE Access*, vol. 10, pp. 110 568–110 589, 2022, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3214837.

[45] R. C. Nickerson, U. Varshney, and J. Muntermann, "A method for taxonomy development and its application in information systems," *European Journal of Information Systems*, vol. 22, no. 3, pp. 336–359, 2013, ISSN: 1476-9344. DOI: 10.1057/ejis.2012.26.

[46] P. Avgeriou and U. Zdun, "Architectural patterns revisited - A pattern language," in *EuroPLoP' 2005, Tenth European Conference on Pattern Languages of Programs, Irsee, Germany, July 6-10, 2005*, A. Longshaw and U. Zdun, Eds., UVK - Universitaetsverlag Konstanz, 2005, pp. 431–470.

[47] M. Jaiswal, "Software architecture and software design," *International Research Journal of Engineering and Technology (IRJET) e-ISSN*, pp. 2395–0056, 2019.

[48] R. Prieto-Díaz, "Domain analysis: An introduction," *SIGSOFT Softw. Eng. Notes*, vol. 15, no. 2, pp. 47–54, Apr. 1990, ISSN: 0163-5948. DOI: 10.1145/382296.382703.

[49] V. Stanojević, S. Vlajić, M. Milić, and M. Ognjanović, "Guidelines for framework development process," in *2011 7th Central and Eastern European Software Engineering Conference (CEE-SECR)*, 2011, pp. 1–9. DOI: 10.1109/CEE-SECR.2011.6188465.

[50] F. Alzhrani, *BELA: Blockchain Event Log App*, https://github.com/alzhraniFouz/BELA, version 1.3.0, 2023. DOI: 10.5281/zenodo.7620035.