IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 27 September 2023, accepted 10 October 2023, date of publication 16 October 2023, date of current version 23 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3325060

RESEARCH ARTICLE

BlockASP: A Framework for AOP-Based Model Checking Blockchain System

ANAS M. R. ALSOBEH[1,2] **AND AWS A. MAGABLEH**[1,3]

[1]Department of Information Systems, Faculty of Information Technology and Computer Science, Yarmouk University, Irbid 21163, Jordan
[2]School of Computing, Information Technology (ITEC) Program, Southern Illinois University Carbondale, Carbondale, IL 62901, USA
[3]Department of Software Engineering, Prince Sultan University, Riyadh 11586, Saudi Arabia

Corresponding author: Aws A. Magableh (aws.magableh@yu.edu.jo)

ABSTRACT Blockchain systems are lauded for their security, and reliability. Security is a cornerstone, as they employ cryptographic techniques to ensure the immutability of data, making it extremely resistant to tampering. With decentralized networks, they also reduce the risk of a single point of failure, enhancing reliability. Model checking plays a vital role in ensuring the security, and reliability of blockchain systems. However, traditional model-checking approaches face challenges in handling the inherent dynamism exhibited in blockchain systems. To overcome this challenge, Aspect-Oriented programming (AOP) offers capabilities to enhance blockchain model checking through the modularization of cross-cutting concerns, enabling traceability and monitoring, facilitating dynamic instrumentation, and supporting fine-grained property specifications. The aim of this research is to enable more effective and efficient verification of dynamic behaviors in blockchain systems compared to conventional model-checking techniques using AOP. As a result, this research introduces *BlockASP*, a novel blockchain model verification method that leverages AOP to analyze and monitor dynamic behavior of the blockchain system. *BlockASP* integrates the benefits of aspect-orientation and model checking into the blockchain architecture to strengthen security, and reliability. This research has examined prior art that are related to blockchain modeling using Object-oriented (OO) and those are using AOP. Our research has proposed and discussed the *BlockASP* technique, the research provided a case study to demonstrate the validity and superiority in facilitating the monitoring of dynamic blockchain behavior using AOP compared to traditional approaches such as Model-Driven Architecture (MDA).

INDEX TERMS Aspect-oriented programming (AOP), BlockASP, blockchain, model checking, dynamic behaviors, real-time security verification.

I. INTRODUCTION

Blockchain technology is widely recognized for its transformative impact on various sectors, including banking, supply chain management, and healthcare [1]. Its decentralized nature, immutability, and transparency make it an attractive solution for numerous challenges [2]. As blockchain systems grow in complexity, guaranteeing their accuracy and dependability becomes increasingly important, and we believe that model checking can play a significant role in achieving this objective [3].

Model checking is a well-established technique in the field of formal verification that is used to evaluate whether a given system meets a predefined set of requirements or desired attributes. The process involves the creation of a model that accurately depicts the behavior of the system, followed by a comprehensive analysis of all possible states and transitions in order to ascertain the validity of the stated features [4]. Hence, the use of model checking offers a viable approach to address the inherent difficulty of guaranteeing the accuracy and dependability of blockchain systems by a comprehensive examination of all potential states inside the system's architecture. In addition, the use of model checking enables the verification of design choices in blockchain systems. This allows for the evaluation and comparison of different design options, parameter settings, or changes in protocols [13]., or protocol variations [13].

Through the analysis of the formal model under different scenarios and property specifications, model checking aids in identifying trade-offs, potential bottlenecks, or performance

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

concerns, enabling the optimization of the system's design to enhance correctness and reliability [5]. However, conventional model-checking approaches face challenges in handling the inherent dynamism of blockchain systems that can arise from runtime changes or developing requirements [54]. To overcome these limitations, innovative model-checking methods are needed that can capture and analyze the dynamic behaviors exhibited by blockchain systems [6].

Many facets of blockchain systems, including consensus methods, smart contracts, and transaction validation, have benefited from model checking [2], [38]. Yet, conventional model-checking approaches struggle to accommodate the dynamism inherent in blockchain systems, which may emerge from runtime adjustments or changing system needs. Consequently, there is a growing demand for innovative methods capable of analyzing dynamic behavior in blockchain networks. Aspect-oriented programming (AOP) offers potential in this area by modularizing crosscutting concerns and enabling flexible instrumentation to monitor runtime events [7], [15].

AOP is a programming paradigm that separates out and abstracts away problems that affect many parts of a system [18]. AOP has been successfully employed in numerous domains [8], [9] to improve the modularity, maintainability, and evolvability of software systems. This research presents a novel method that harnesses AOP to analyze and perform model checking on the dynamic behaviors exhibited by blockchain systems. This paper argues that integrating AOP into blockchain model checking can significantly enhance the verification of dynamic behaviors. AOP provides separation of concerns, traceability, dynamic weaving, and fine-grained specifications to facilitate analysis of blockchain systems' runtime adaptations [14], [45], [52]. This research also examines the efficiency and performance implications of integrating AOP with model checking.

Building upon the aforementioned potential advantages of AOP for model checking in blockchain systems, this study introduces a systematic method to integrate AOP and model checking, aiming to address the challenges associated with the dynamic and complex nature of blockchain systems. The innovative framework presented herein not only supports the modularization of cross-cutting concerns into separate, manageable aspects, but also utilizes the dynamic capabilities of AOP for runtime modification and analysis of system behavior. By dynamically weaving and unweaving aspects, it is anticipated that this approach will yield more precise and flexible verification processes [10]. The paper reviews prior art that presents the proposed integration and validates the effectiveness of AOP-based blockchain model checking through a case study. The goal is to advance research at the intersection of formal methods, distributed systems, and AO software engineering [47], [52], [53], [54], [56], [57], [58]. As blockchain technology continues to permeate various sectors, this research hopes to shed light on this area.

This paper is organized as follows. The introduction provides background on blockchain technology and model checking, and motivates the need for innovative model checking approaches. Section II gives an overview of key concepts like AOP, blockchain architecture, and model checking. Section III reviews prior work on blockchain model checking using object-oriented (OO) and aspect-oriented (AO) approaches. Section IV introduces our proposed *BlockASP* framework and describes its architecture and components in detail. Section V outlines the workflow for applying *BlockASP* and gives an example. Section VI compares *BlockASP* to traditional OO methods. Finally, the conclusion summarizes the contributions and mentions future work. The overall flow moves from introducing the concepts and motivation, to reviewing previous work, proposing our novel approach, describing its application, and concluding with a comparison and summary.

II. RESEARCH BACKGROUND AND LITERATURE REVIEW

This section provides a concise overview of the key ideas and terminology relevant to our research. To effectively apply and assess the proposed method for blockchain model checking with dynamic behavior analysis using an aspect-oriented approach, it is crucial to have a firm grasp of some key concepts. Blockchain technology has gained significant attention due to its potential in providing decentralized, transparent, and secure systems. However, ensuring the correctness and reliability of blockchain applications is a critical challenge. Model checking is a formal verification technique used to analyze the behavior and properties of systems [54]. This section explores the use of aspect-oriented (AO) and object-oriented (OO) approaches in blockchain model checking, highlighting their importance and advancements [15], [16].

AOP is a paradigm that improves modularity by allowing the separation of cross-cutting concerns into standalone modules called aspects. It addresses the limitations of traditional object-oriented programming in encapsulating behaviors like logging, security, and monitoring that span multiple components. In AOP, aspects modularize these cross-cutting concerns separately from the core system logic. Pointcuts specify join points where aspects can be applied, like method calls or exception handling. By weaving aspects at join points, AOP enables modularization and reuse of these concerns [10], [18].

A blockchain is a digital ledger that maintains a sequential record of transactions. It is designed to prevent unauthorized modifications due to its distributed and decentralized nature. The blockchain operates as a linked list of blocks, with each block containing a timestamp, a reference to the preceding block, and a collection of transactions. In simple terms, a blockchain system consists of three main components: nodes (participants), a consensus mechanism, and a cryptographic algorithm [2]. Blockchain architectures can vary depending on the specific implementation and use case. Here, we present a general overview, in a simplified form.

A blockchain architecture typically consists of blocks, each containing a list of transactions or data. These blocks are given a unique identifier called a hash, along with a timestamp and a reference to the preceding block [48]. Model checking is a formal verification technique used to analyze and verify the correctness of system models. It involves systematically exploring all possible states and transitions of a model to check if certain properties or specifications hold true. Model-checking techniques typically involve the following steps [12]:

- Model representation: The system to be verified is represented as a formal model, which captures its behavior, states, and transitions. The model can be expressed using various formalisms such as finite state machines, temporal logic, or Petri nets.
- Property specification: Properties or specifications that the system should satisfy are specified using formal language. These properties define desirable behaviors, safety conditions, or invariants that the system must adhere to. Examples of properties include the absence of deadlocks, preservation of data integrity, or adherence to a given protocol.
- State space exploration: The model checker systematically explores the state space of the model by examining all possible states and transitions. It generates a state graph or a state space graph that represents all reachable states and their connections.
- Property verification: The model checker then checks the specified properties against the state space. It verifies if the properties hold true in all possible states or if there are any violations. Counterexamples may be provided to highlight specific scenarios where the properties fail.
- Analysis and optimization: Model checking can involve various analysis techniques to improve efficiency and scalability. These include techniques such as abstraction, partial order reduction, symmetry reduction, and property-specific optimizations.

A. MODEL CHECKING USAGE IN BLOCKCHAIN

Blockchain model checking involves the systematic analysis of blockchain systems to verify their correctness and identify potential vulnerabilities. Research in this area focuses on formalizing blockchain protocols, consensus mechanisms, and smart contracts to ensure the absence of security flaws, such as double-spending attacks and data inconsistency issues. The importance of blockchain model checking lies in its ability to enhance security, reliability, and performance in blockchain-based applications. By employing formal verification techniques, model checking helps detect vulnerabilities, prevent attacks, and ensure compliance with desired properties, such as consistency, liveness, and fairness. Moreover, it aids in the development of trustworthy blockchain systems, fostering trust among users and stakeholders.

According to Afzaal and his team, crowdsourcing is a computing paradigm that relies on human intelligence for complex tasks. However, it faces security and trust issues. Blockchain technology offers potential solutions, but finding a suitable consensus protocol for crowdsourcing is challenging. To address this, the Trust and Transactions Chain (TTC) protocol is proposed. It selects trusted leaders and validators based on a trust model. The TTC protocol ensures correctness and prevents fraud, as well as Sybil and eclipse attacks. Model checking is used for formal verification, with results analyzed for verification time and visited states [58].

In [19], Wagner and his colleagues showed that the verification of distributed agreement protocols and their applications is challenging due to system complexity and unboundedness. However, recent progress has shown that careful modeling can enable the reduction of verification for unbounded distributed agreement-based systems to small, finite-state systems using model checking. This work extends this reduction approach to doubly-unbounded systems with unbounded data domains tools, called Venus.

In [20], Sadique and his team illustrated an IoT paradigm that aims to improve society with smart and autonomous devices, requiring seamless collaboration. However, the increasing number of connected devices poses identity management challenges for resource constrained IoT devices. Traditional systems are not feasible due to device heterogeneity. As a solution, this paper presents a distributed identity management architecture based on distributed ledger technology (DLT) for edge IoT devices. The proposed model is adaptable to any IoT solution, enabling secure communication. The study reviews consensus mechanisms used in DLT and their connection to IoT research, specifically identity management. The proposed location-based model is generic, distributed, and decentralized. Formal verification tools such as Scyther and SPIN model checker verify the model's security and performance. FobSim, an open-source simulation tool, analyzes deployment performance for fog and edge/user layer DLT. The results demonstrate how the proposed decentralized identity management enhances user data privacy and secure communication in IoT. Similarly, *BlockASP* employs formal modeling and logic to verify that a blockchain system adheres to predefined specifications and properties. It can check security invariants, performance requirements, privacy policies, etc. While Scyther, SPIN and FobSim are specific model checking tools, *BlockASP* provides a more generalized framework that can integrate different model checkers and verification techniques. The key difference is that *BlockASP* utilizes AOP to modularize crosscutting concerns and enable dynamic analysis of blockchain systems. In contrast, Scyther and SPIN employ more traditional model checking approaches. However, *BlockASP* could potentially leverage tools like Scyther and SPIN for performing the actual model checking process once the aspects have been defined. Similarly, FobSim could be used for performance analysis.

The authors in [21] proposed that an energy blockchain is crucial for applications like energy data sharing and distributed power trading in the energy industry's digital transformation. Federated learning (FL) can analyze energy

data while preserving privacy. However, traditional FL lacks trust and faces data poisoning and deception attacks in an energy blockchain environment. To address this, a game theory-based incentive mechanism is proposed to promote collaborative security in FL. An FL model is presented, ensuring privacy and collaborative security. A game model involving energy departments and a supervisory committee is constructed, and an incentive mechanism using smart contracts is designed based on game theory. The mechanism discourages malicious behaviors during iterative training of FL, even with lower model checking accuracy. The proposed mechanism achieves collaborative security by leading the game model to a Nash equilibrium (NE). Security analysis and experimental evaluation demonstrate the feasibility, robustness, reliability, and low complexity of the incentive mechanism in energy blockchains. In contrast, *BlockASP* utilizes AOP and logic-based formal verification to guarantee adherence to security properties.

Smart contracts are crucial in blockchain software development, and their immutability requires thorough verification and validation before deployment [22]. The authors introduced a model-based testing approach for validating Ethereum smart contracts and ensuring their correctness. The process involves four steps: (1) modeling the smart contract and its blockchain environment using UPPAAL Timed Automata, (2) generating abstract test cases with the UPPAAL CO$\surd$ER tool, (3) dynamically executing the generated test cases, and (4) analyzing the test results and generating reports. The approach is exemplified through its application on the Ethereum blockchain, specifically the electronic voting case study. Smart contracts on computer programs that run on blockchains require thorough security checks. Existing tools often yield false positives and negatives for common vulnerabilities. The research study in [23] focuses on the specific vulnerability class of unchecked low-level calls, which is challenging to detect. A proposed approach for Ethereum smart contracts reduces this class to model-checking, employing linear temporal logic (LTL) and an off-the-shelf model checker. After evaluating nearly 200 smart contracts, the approach proved highly effective, detecting zero false positives and negatives.

The researchers in [24] suggested that coalition logic (CL) is a suitable approach for verifying the properties of smart contracts on a blockchain. However, smart contracts can be upgraded by introducing new versions on subsequent blocks. This paper extends a recent formalism that allows reasoning about updated CL models by introducing a temporal modality that connects newer versions to previous ones [17]. This extension is a step towards verifying the properties of smart contracts with upgrades. The paper also discusses the properties of the resulting logic and the complexity of its model-checking problem. A recently introduced use of smart contracts is to execute business processes on blockchains. Composite smart contracts, which execute other contracts through external calls, present security challenges. To address this issue, Mouhamad and his team proposed verifying the security and correctness of composite smart contracts in Ethereum. The approach utilizes finite state machine models and model checking, considering both standard and context-dependent properties. Computation tree logic is used to express the properties, and the nuXmv symbolic model checker verifies the model [25].

Alwhishi and his team [26] worked on the crucial task of verifying the reliability and efficiency of blockchain systems based on smart contracts. However, it remains a challenging task and a topic of active research across various domains. This paper focuses on the verification of these systems, which are modeled using trust protocols under uncertainty. To tackle this problem, the authors propose a verification approach called three-valued model checking. They introduce a new logic, extending the Computation Tree Logic of Trust (TCTL) to the three-valued case (3v-TCTL), which enables reasoning about trust with uncertainty in smart contract-based systems. A transformation approach is also presented to reduce the 3v-TCTL model-checking problem that exists in the classical case [26].

Blockchain technology has gained significant attention, particularly for automating asset transfers [27]. However, the presence of malicious participants in the network creates vulnerabilities in blockchain consensus algorithms, leading to potentially devastating consequences. Surprisingly, despite the flaws observed in various consensus algorithms, there has been no comprehensive verification of any blockchain consensus. This paper addresses this gap by conducting model checking on the consensus algorithm of the Redbelly blockchain. In [28], smart contracts play a crucial role in blockchain applications, ranging from encrypted digital currency to various domains. However, due to the immutability of the blockchain, any bugs or errors in smart contracts can lead to substantial economic losses since they cannot be modified once deployed. Recently, numerous security issues have been identified in smart contracts, highlighting the importance of verifying their correctness before deployment on the blockchain. This work focuses on the verification of smart contracts in Ethereum transactions, using the Spin model checker for formal verification.

Krichen and his colleagues showed that the security of smart contracts is a significant challenge that hinders their advancement. The authors in [28] and [29] identified the urgency of verifying the correctness of smart contracts before their deployment due to the immutability of the blockchain. Despite the progress made in the verification of smart contracts, the authors have acknowledged a lack of comprehensive synthesis of these findings. In contrast to numerous other studies, our approach offers a more robust and real-time solution by capturing dynamic behaviors of the blockchain system, establishing a finite state model of the system that incorporates aspects and dynamic behavior data, and applying formal logic to explore all possible states and transitions. Hence, our research builds upon and extends the work done

in [26], [27], [28], and [29], and other similar works [30], [31], [32], [33], [34], [35], by providing a comprehensive, real-time model-checking framework for blockchain systems.

These studies highlight the importance of thoroughly verifying smart contract correctness and security due to their immutability once deployed on blockchains. However, our *BlockASP* framework provides a robust and generalized approach for blockchain model checking, including smart contracts. It utilizes AOP to modularize crosscutting concerns like security policies. It also can complement existing smart contract verification tools by providing a flexible way to inject security aspects and properties into the contract logic. Formal verification can then validate adherence to these security specifications. For instance, *BlockASP* aspects can encapsulate security requirements while nuXmv or another model checker verifies the contract modeled as a finite state machine, so this prevents false positives. Upgrading contracts can be handled by dynamically weaving new aspects to represent added features. Advice can link old and new contract versions. Composite contracts can be modeled compositionally in *BlockASP* using inter-type declarations to connect contracts, and dynamic weaving then enables checking inter-contract interactions.

B. OBJECT-ORIENTATION (OO) -BASED BLOCKCHAIN MODEL CHECKING

Early research in blockchain model checking predominantly utilized object-oriented approaches to analyze system behavior. OO model-checking techniques enabled the verification of blockchain protocols, transaction validation, and smart contract execution. The researchers in [36] introduced model-checking methods to validate a modeling language called RPOO, which is an object-oriented Petri net. The aim is to enhance the applicability of model checking in model-based software development. To achieve this, the study offered a way to specify object properties without delving into the intricacies of Petri nets, utilizing the Petri nets' semantics solely for constructing the state space of the models. Additionally, they present algorithms for assessing properties expressed in the branching-time temporal logic CTL. The proposed approach emphasizes the explicit representation of the state space, focusing on its OO characteristics.

However, Češka et al. [37] spoke about the state space explosion problem which poses a significant hurdle to infinite-state model checking. To mitigate this challenge, several advanced techniques were proposed to reduce the size of state spaces. Among these techniques, partial-order reduction emerged as a successful approach, particularly in the realm of software systems. Češka and colleagues investigated the utilization of partial-order reduction in the context of object-oriented Petri nets, which incorporate dynamic instantiation, late binding, and garbage collection as distinctive features. The study aims to assess the effectiveness of applying partial-order reduction in managing state space explosion within the OO Petri net paradigm. The limitations of OO paradigms in addressing crosscutting concerns and complex interactions in blockchain systems led to the exploration of aspect-oriented approaches. Our research is designed to address the dynamic and crosscutting nature of concerns in blockchain systems that cannot be effectively handled by OO paradigms [37].

Unlike early research that relied on OO paradigms, the proposed framework leverages AOP to effectively handle the crosscutting concerns inherent in complex blockchain systems. Beyond merely providing static verification, our model aims for real-time capturing of dynamic behaviors to offer a more nuanced and complete understanding of the system's state. Unlike existing work that focuses on explicit state space representation, our approach uses a finite state model enriched with AOP and dynamic behavioral data for comprehensive verification.

Our framework also incorporates formal logic for a rigorous exploration of all states and transitions, a feature that makes it more exhaustive in scope. While prior work like that of Češka et al. focused on tackling the state space explosion problem through specialized techniques such as partial-order reduction, our AOP-based model inherently offers a more efficient state space traversal.

C. ASPECT-ORIENTATION (AO)-BASED BLOCKCHAIN MODEL

AOP provides a promising approach to overcoming the challenges posed by object-oriented paradigms in blockchain model checking. By modularizing crosscutting concerns, AOP enables the separation of blockchain-specific concerns, such as consensus, security, and fault tolerance. This approach allows for more precise analysis, easier maintenance, and scalability in the verification process. Although several attempts of using AOP with model checking such as in [41], [42], [43], and [44], less attention has been paid to using AOP model checking in the context of blockchain. That is where our proposed approach and research comes in to fill the gap.

The authors in [39] stated that AOP is a programming paradigm that enables modularization of crosscutting concerns, such as synchronization policies, resource sharing, and performance optimizations, by separating them from objects as independent aspects. The weaving process, performed by a compiler known as a weaver, integrates aspects and objects to form a cohesive program. However, verifying the correctness of a woven program in AOP presents challenges, as crucial behaviors are heavily influenced by aspect descriptions. To address this issue, this research paper proposes an automatic verification approach utilizing model-checking techniques to detect unexpected behaviors. In [40], the researchers introduce a novel framework for verifying properties of system transaction-level models (TLM) during simulation. TLM is gaining popularity for high-level hardware design, offering improved simulation performance and early prototyping for system-on-chip (SoC) designs. The framework utilizes AOP techniques to automate monitoring

of the design under verification and enable the creation of assertion checkers tailored to TLM requirements. Neither of the above-mentioned works touched significantly on the blockchain environment.

BlockASP adds several unique contributions that distinguish it from other studies on model checking using AOP. While existing research in AOP-based model checking has largely been focused on modularizing crosscutting concerns like synchronization policies, resource sharing, and performance optimizations, it has paid limited attention to the specific context of blockchain systems. For example, although some work has been done on verifying woven programs in AOP or on using AOP techniques in transaction-level models (TLM) for system-on-chip (SoC) designs, these studies have not been specifically tailored for blockchain environments.

Moreover, our work fills this gap by focusing on real-time model-checking in the unique landscape of blockchain systems. Not only do we utilize AOP to modularize crosscutting concerns inherent in blockchain, such as consensus algorithms, security features, and fault tolerance, but we also introduce real-time capabilities for capturing dynamic behaviors [11], [36], [37]. This leads to a more accurate and efficient model-checking process specially designed for the complexities and nuances of blockchain technology. Therefore, our framework offers a more specialized, robust, and real-time solution for blockchain model checking, setting it apart from existing research in significant ways.

III. BLOCKASP: AOP-BASED MODEL CHECKING FRAMEWORK FOR BLOCKCHAIN SYSTEMS

BlockASP architectural solution meticulously engineered to address the unique security challenges inherent to blockchain technologies. Leveraging the modular benefits of AOP [46], *BlockASP* seamlessly integrates with contemporary model-checking algorithms, creating a robust and real-time security assessment toolset that significantly outperforms existing solutions.

Upon the strategic injection of pre-defined AOP aspects into the target blockchain system, *BlockASP* kick-starts an intricate model-checking pipeline. Each aspect comes pre-loaded with state-of-the-art statistical model-checking algorithms. These algorithms are designed to rigorously analyze the real-time, dynamic behaviors of the blockchain system as they unfold. This analysis is backed by a Finite State Machine (FSM) that is programmatically generated to encapsulate not only the system's dynamic behavior but also the operational logic introduced by the injected aspects.

What sets *BlockASP* apart is its adoption of formal logic frameworks, most notably temporal logic, which serves as the underpinning for its model-checking engine. This enables *BlockASP* to systematically traverse every possible state and transition within the blockchain environment, providing a comprehensive assessment of the system's adherence to its specified security and functional parameters. It's worth noting that the use of formal logic ensures the logical rigor and completeness of the verification process, effectively closing any loopholes that could be exploited.

In technical terms, *BlockASP* represents a paradigm shift in how blockchain security is approached, marrying the modular advantages of AOP with the rigorous analytical power of model-checking algorithms. This confluence allows *BlockASP* to offer a highly granular, real-time security assessment tool that is uniquely positioned to meet the ever-evolving challenges of blockchain security.

Figure 1 illustrates the design of the *BlockASP* framework, highlighting the various layers and modules, and their corresponding benefits and challenges. The architecture is based on four primary layers: the Block Analysis Layer, the Block Aspect Layer, the Model Checking Layer, and the Advice Layer. The layers form a sequential process that starts with the integration of AOP with blockchain model checking, followed by identifying and capturing dynamic behaviors using AOP, model checking of dynamic behaviors in a blockchain system, and finally, evaluation and refinement.

The initial phase of the proposed Analysis Layer involves the collection of AOP/OO applications, which serves as the identifier for the crosscutting concerns code requirements. In this layer, the OO application is refactored into AOP applications by identifying the expected results based on the code behavior for subsequent phases. To accomplish this, a tracer is utilized to monitor the code behavior step by step, extracting the crosscutting concern requirements and mapping them to contextual properties in the analysis model phase (context data). This layer comprises the data analysis module, which reads blockchain's transaction data, such as transaction ids, input transactions, output transactions, and malicious or hacked transactions, etc.

The Analysis Layer involves gathering a collection of web service (WS)-based blockchain applications along with their specific application requirements. This information serves as a foundation for understanding the context and needs of various blockchain applications. After that, a context-aware model is established to analyze the WS-based blockchain applications. This model should also capture the relationships between different application crosscutting concerns, such as performance, security, and reliability, and the contextual factors that influence them [55]. In a traditional approach, the AspectJ Weaver traces states of objects using context-related data, where *BlockASP* leverages AspectJ, to trace the states of objects in the blockchain applications. This allows you to check and manipulate block states in the context of the overall application. By doing so, you can track and analyze context-related data throughout the application lifecycle.

The Block Aspect Layer introduces blockchain to analyzing crosscutting concern and its relationship with the context that we encapsulated in aspect, and exposure for context; it presents the states of objects for each piece of class and for each transaction's class. The layer provides immutable and transparent functionality. It comprises the Data Hashing Module, which generates unique hash values for each

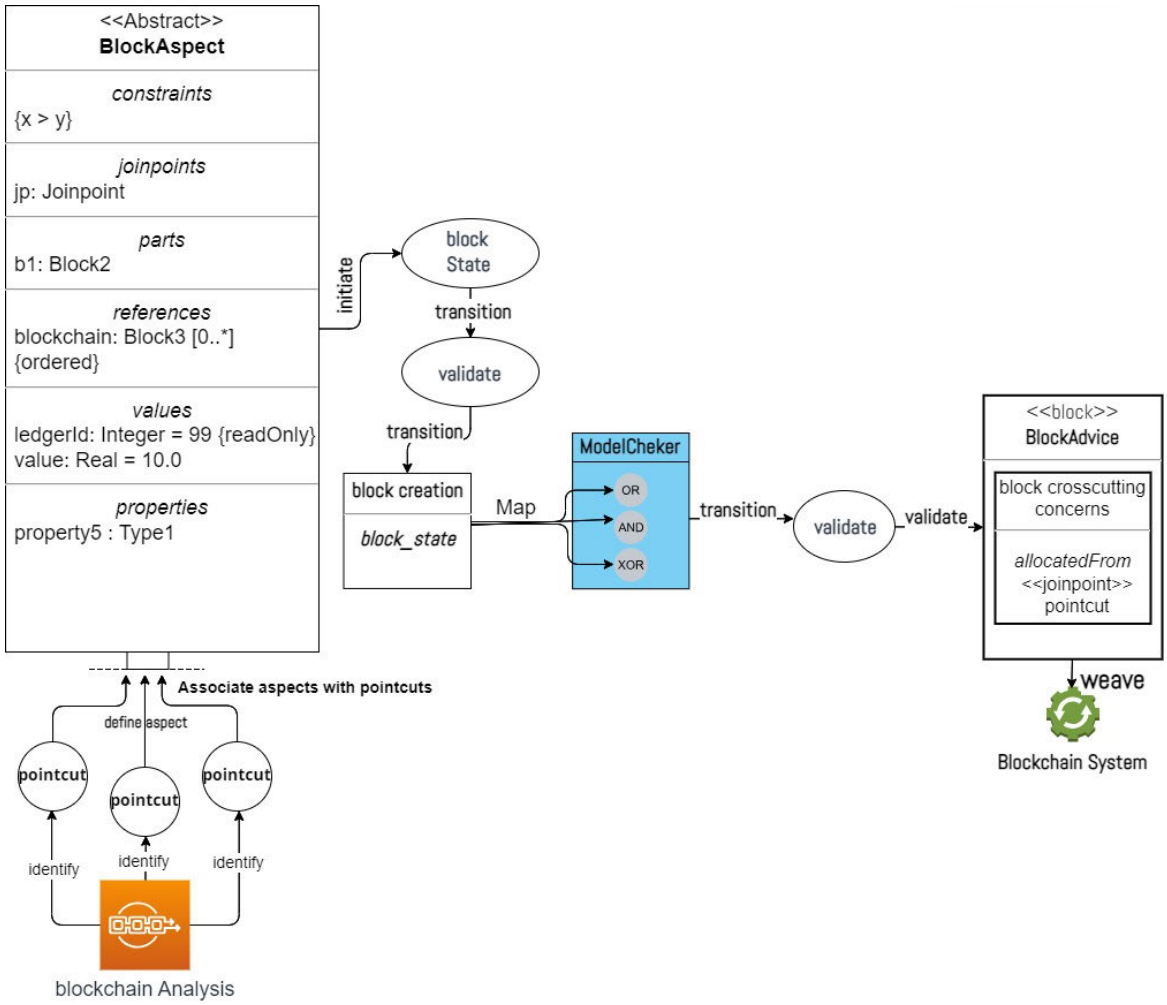

FIGURE 1. The overall sequential process of the proposed approaches.

data item, which retrieves data and its associated metadata from blockchain transactions. In our previous examples, the ledger was implemented as a *Map<String, Double>* inside the blockchain class, where the keys represent the account identifiers (e.g., names), and the values represent the block values.

The Model Checker Layer facilitates the practice of statistical model checking, wherein it encompasses the ledger responsible for identifying the code's behavior based on extracted attributes and parameters. The ledger is responsible for documenting all transactions that have taken place within the system. It also determines and applies the most suitable verification technique, such as propositions, depending on the specific details of the data request. The utilization of a data-driven methodology enhances the efficacy of transaction preservation and optimizes the implementation of check measures by taking into account the contextual information of the data request.

The Advice Layer functions as the intermediary between the blockchain interface and the underlying modules. It enables transaction requesters to search for and request specific data from the Model Checking Layer. The system facilitates the registration and updating of transaction data and policies on both the Blockchain Aspect Layer and the Model Checking Layer.

A. APPLYING BLOCKASP: WORKFLOW DESCRIPTION

Upon obtaining context data from the Analysis Layer, it is then passed into two observer patterns: AOP observer *A* and OOP observer *B*. These observer patterns are utilized to log and save the data into log files [49]. The AOP observer traces the states of all classes and extracts the context data during runtime, as mentioned above. The results obtained from the observer patterns serve as inputs for the subsequent step of statistical model checking. In this step, the observer pattern data is subjected to a specific rule that detects adaptive code behavior based on the principles of AOP. AO pointcuts and joinpoints are placed to check the state of each object. The meaning of valid state case is that the results are identical for each piece of original code with AOP.

For instance, in a cryptocurrency exchange system, *BlockASP* offers a crawler for accessing the data from a cryptocurrency exchange, where all transaction information can be publicly accessed [53]. It executes in three phases: 1) User provides a Java List of transaction block hash strings and defines where the data will be stored in its finished csv format. The crawler aspects pull the transaction IDs for the given data blocks from the online exchange API, verify uniqueness and optionally verify no hacked transaction IDs are included in the pulled transactions. 2) Next, the parser will concatenate all the transactions and poll the exchange API for each transaction's metadata. 3) The pre-processor then creates a transaction entry from the metadata of a transaction, capturing or calculating the following features: fraudulent = binary flag for indicating the transaction is fraudulent or not; txid = transactionId hash; input_transaction_size = monetary size of the cumulative input transaction; output_transaction_size = monetary size of the cumulative output transaction, etc.

The *BlockASP* system integrates transaction data into the blockchain using the transaction aspect at the Aspect layer. It characterizes transaction data attributes and properties at the model checking layer, encapsulating elements such as purpose, duration, and data sharing compensation. These attributes are subsequently symbolized by model-checking propositions to apply fitting modeling based on the specific nature of the data context request. The Model Checking layer is used to validate the data of the code upon logging in aspects, which specifies the data proposition request. The system validates context data using the Quality-of-Service Module (QoSM). QSoM sends context requests to the model-checking layer where the data propositions are triggered, therefore it verifies properties and attributes for data sharing, and if validated, the quality model selects the appropriate technique (e.g., anonymization, obfuscation, or differential privacy).

BlockASP's QoSM is implemented as a paradigm for dynamically modifying the Java application's context data during runtime, known as aspect weaving. Quality characteristics and sub-characteristics for software products and systems are defined in the ISO/IEC 25010 standard, on which the quality model is based. Functionality, dependability, usefulness, efficiency, maintainability, and portability are prioritized throughout the code. Aspects may be used to quantify any desirable attribute you care about. There is an aspect-based metric that corresponds to each requirement. Aspects are modules that include solutions to problems that affect several areas. Pointcuts, expressions that match certain branches in the system's execution flow, may be defined by an aspect. Advices are actions that are carried out at junctions when the pointcuts are a match. The QoSM evaluates the proposed checking model for changing the context data of a Java application within runtime aspect weaving. QoSM aspect implements the quality model in Java and AspectJ. This code outlines a proposed quality model for a Java application that uses AOP to dynamically weave context data during runtime. This aspect deals with model checking, hypothesis testing, and context data measurement. A model checker takes a model and a property as inputs, and outputs either a claim that the property is true or a counterexample falsifying the property [56], [57].

In a cryptocurrency system that leverages *BlockASP*, when a request for data access is granted, the extended abstract model checking aspect classes initiate the smart model checking process. This allows the user access to both the encrypted data and its associated metadata situated on the blockchain aspect layer. The encrypted data and its metadata are subsequently retrieved from the blockchain Aspect Layer and decrypted in real time. This real-time processing ensures a seamless and prompt data access experience for the user. The data is securely transmitted back to the user via the extended security crosscutting concern aspect, specifically targeting the data quality model. Along with the data, any relevant provenance and consent information is also dynamically supplied. This comprehensive process ensures secure data access and delivery while maintaining transparency and data integrity within the BlockASP-based cryptocurrency system.

ModelChecker, *HypothesisTester*, *ContextDataMeasurer*, and *CyclomaticComplexityCalculator* are the four modules defined by *BlockASP*. Each module includes a shorthand and some recommendations on how to improve a certain element of the model. For instance, the *ModelChecker* feature checks whether a model satisfies a specific property of the system by a pointcut and some recommendations. A pointcut and recommendation are provided by the *HypothesisTester* aspect for the hypothesis testing function, which verifies if the proposed method and model modify the Java application's context data during runtime. The *ContextDataMeasurer* feature provides a pointcut and recommendation for the context data measurement function, which compares the data's size before and after the recommended method and model are applied. The cyclomatic complexity of the suggested method and model may be decided with the help of the *CyclomaticComplexityCalculator* feature, which provides shorthand and some guidance for the cyclomatic complexity calculation function.

In a cryptocurrency system, data and attributes relevant to the system are defined in the code, including the system's state before and after applying the recommended strategy and model, as well as the environments and platforms that are compatible with the solution. The program then evaluates the suggested procedure and model based on the criteria and metrics established by the QoSM. Particularly, the *ModelChecker* aspect uses a pointcut and some suggestions to determine whether a given model matches a certain system characteristic. The model-checking process uses a formal verification strategy to determine whether a model is suitable for a given system. This is given a model and a property, which may prove the property to be true or produce a counterexample to show that it is untrue. If we use the model checking function to verify that we can, for example, edit the Java application's context data during runtime aspect weaving,

then we know that the suggested method and model will perform as expected. Moreover, the *HypothesisTester* module provides a shorthand and suggestion for the hypothesis testing function, which checks whether the suggested approach and model alter the runtime context data of the Java program. The hypothesis testing function uses a module checking layer aspect to evaluate whether there is sufficient evidence to reject or accept a null hypothesis. A null hypothesis is a statement that does not make any presumptions regarding the connection between two data sets. A hypothesis testing function takes in the null hypothesis and an alternative hypothesis and returns either a claim that the null hypothesis is true or a claim that the alternative is true. The hypothesis testing feature may be used to determine whether the proposed approach and model are reliable by examining their ability to meet the challenges of experimentation and refinement.

B. BLOCKASP ARCHITECTURE: COMPONENTS

Figure 2 depicts the *BlockASP* components. In practical implementations of blockchain systems, blocks possess the capability to incorporate supplementary data and exhibit more intricate configurations. The blockchain class plays a pivotal role within the blockchain system, which is consisted of a series of interconnected blocks that are responsible for maintaining the record of transactions, and offers functionalities for appending blocks and verifying the integrity of the blockchain. The keys within the map are assumed to serve as identifiers for users or entities within the system, while the values represent the quantity of a specific blockchain asset held by each entity. Upon the creation of a new instance of a blockchain, the constructor undertakes the task of initializing both the chain and the ledger. In addition, it appends a genesis block to the blockchain. The genesis block is the initial block within a blockchain, functioning as the foundational element from which the blockchain commences. The ledger class encapsulates a ledger that tracks the value status of the system's blocks. The value status gets updated with each transaction in the *update*() method.

Let B be the set of all blocks, T be the set of all transactions, U be the set of all users, and L be the ledger. $Blockchain = \{B_1, B_2, \ldots, B_n\}$ that maintain a record of transactions T. The class offers functionalities for appending blocks and verifying the blockchain's integrity. Mathematically, the initialization function.

$$Blockchain\,(B, L) \leftarrow init\,(Genesis\ Block)\,.\ addGenesisBlock(B) \leftarrow (index = 0, timestamp, prevHash = 0) \tag{1}$$

The function $addBlock(T)$ adds a new block b to the blockchain. The new b contains a timestamp t, a list of transactions T, and the hash value h of the most recent block. Formally, $b = \{t, T, h\}$. The function *addGenesisBlock* is responsible for creating the genesis block b_0, which has an index value of 0, a timestamp t_0, no transactions $T = \emptyset$, and a previous hash value of "0". Formally, $b_0 = \{0, t_0, \emptyset, \text{"0}''\}$.

The blockchain encompasses various methodologies that contribute to the operational aspects of the blockchain. The *addGenesisBlock*() function is accountable for the creation of the genesis block as shown in Equation 1. This block possesses an index value of 0, a timestamp reflecting the current system time, an absence of transactions, and a previous hash value of "0" due to the absence of any preceding blocks. The *getLatestBlock*() function is employed to retrieve the most recently appended block in the blockchain. Another essential technique is the *addBlock(List<Transaction> transactions)* method. This procedure is accountable for the incorporation of novel blocks into the blockchain. *BlockASP* generates a novel block that includes a timestamp reflecting the present moment, designated transactions, and the hash value of the most recent block serving as *previousHash*. Furthermore, the ledger is updated to accurately reflect the transactions that are incorporated within the recently appended block. The method *updateLedger (List<Transaction> transactions),* i.e., L', is utilized to modify the ledger in accordance with the provided transactions. In the ledger, the sender's value is reduced and the recipient's value is increased by the amount involved in each transaction. The method used to validate the integrity of the chain is the *isChainValid*() method. Equation 3 process guarantees that the computed hash of each block is equivalent to its current hash, and that the hash of the previous block corresponds to the hash of its preceding block. If any block does not pass these checks, the method declares the chain as invalid. The method *consensusAlgorithmIsValid*() is responsible for verifying the adherence of each block within the blockchain to the Proof of Work consensus algorithm. The algorithm exhibits the characteristic whereby the hash value of every block commences with the sequence "0000". The Proof of Work condition can be represented in Equation 4. If any block fails to adhere to this condition, the blockchain is considered invalid.

$$L' = L - SenderValue + RecipientValue \tag{2}$$

$$isChainVaild = \begin{cases} 1, & if\ Hash(B_i) = PreviousHash(B_{i+1}) \forall_i \\ 0, & otherwise \end{cases} \tag{3}$$

$$startsWith(Hash(B_i),\ \text{"0000}'') = 1 \tag{4}$$

$$check(Blockchain) = \prod_{i=1}^{n} isValid(T_i),\ \text{where } T_i \text{ is the } i^{th} \text{ transaction in the blockchain} \tag{5}$$

The Transaction component defines a transaction in the blockchain, which involves a sender, recipient, and an amount. It also has a *isValid*() method for validating the transaction based on sender's values and the transaction's digital signature. However, the methods *getValue*() and *verifySignature*() are placeholders and need to be implemented.

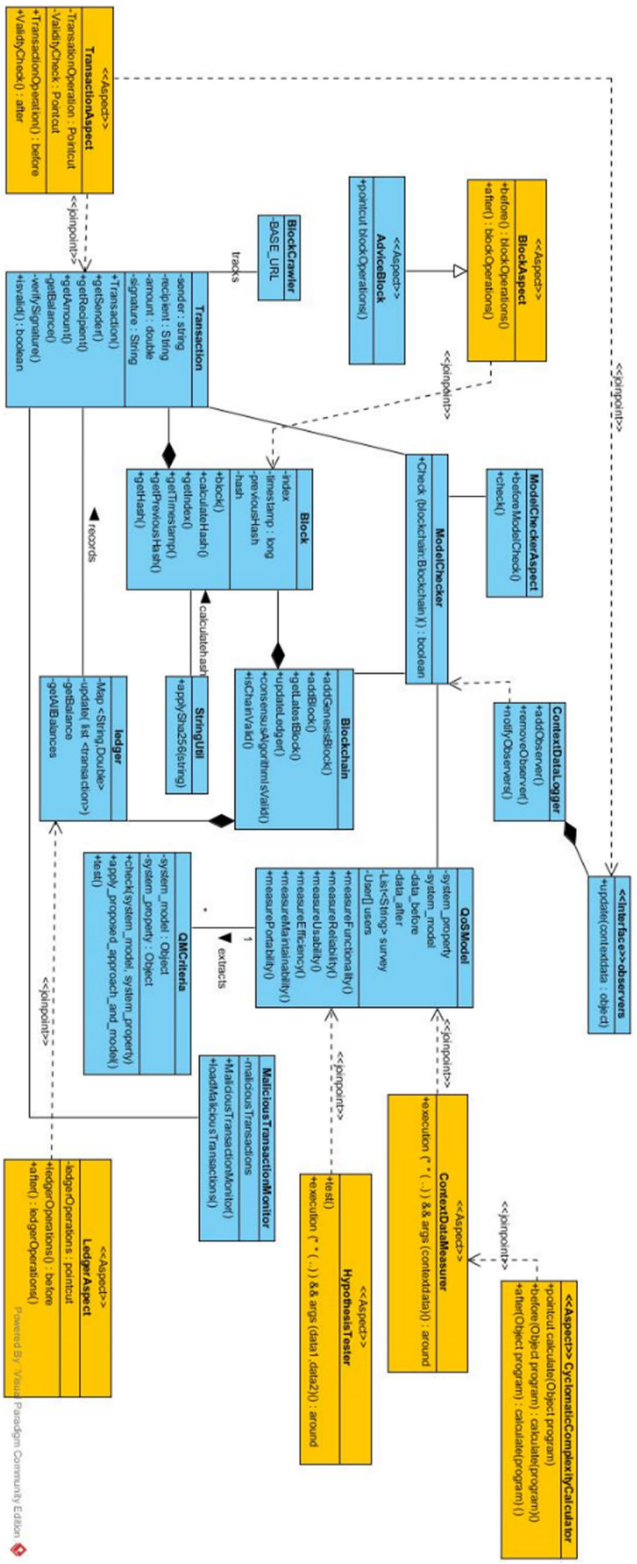

FIGURE 2. BlockASP Components.

The *ContextDataLogger* component acts as a publisher in an observer pattern. It can register observers and notify them with certain context data. The Ledger class encapsulates a ledger that tracks the value of each transaction in the system. The *BlockCrawler* component acts as a utility to fetch raw block data from an online source, here specifically from blockchain.info. It fetches the data based on the provided block hashes, parses it into JSON, and stores the transactions in a CSV file.

The *ModelChecker* class checks the validity of the blockchain and the transactions within each block. It uses a static method *check(Blockchain blockchain)* which takes a blockchain object as its argument. It starts by iterating over all the blocks in the blockchain. Within each block, it goes through every transaction using a nested for-each loop. It then checks the validity of each transaction by calling the *isValid*() method on the transaction object. If this method returns false for any transaction, indicating that the transaction is invalid, the *check*() method will immediately return false, signifying that the blockchain is invalid.

The *ModelChecker* then checks if all blocks in the blockchain are correctly linked. It does this by comparing the hash of each block (except for the last one) with the *previousHash* attribute of the following block. If these hashes do not match for any pair of blocks, this indicates that the chain of blocks has been broken or tampered with. In such a case, the *check*() method will return false. The *ModelChecker* verifies that the consensus algorithm of the blockchain is working correctly. This is done by calling the *consensusAlgorithmIsValid*() method on the blockchain object. If this method returns false, the *check*() method will also return false, indicating that the blockchain is not adhering to its consensus rules.

BlockASP uses tracing crosscutting concerns, i.e., *TraceAspect*, to access the blockchain's transaction interactions and can also serve to log the block's transaction calls for monitoring purposes. It monitors and enforces the compliance of the block's data using the model checking layer. It gets executed when a transaction action takes place in a blockchain program, which enables the program to trace the flow of blockchain, thus finding bugs and gathering usage statistics. This aspect could be extended to pass context data to an observer for further processing. The *BlockAspect, AdviceAspect, and LedgerAspect* encapsulate the interactions and activities occurring in the blockchain system, selecting the proper properties and attributes of the transactions. *BlockASPect* is the logic of the Block Analysis Layer. Its purpose is to wrap around 'block operations', i.e., any activity that involves a block, like registering transaction data, setting data attributes, approving data access, and auditing data. *AdviceBlock* extends a class called *BlockAspect*. The aspect contains a pointcut definition called *blockOperations*, which specifies a joinpoint in the program. This joinpoint is a well-defined moment in the execution of a block's transaction, such as a method call or variable access. The pointcut expression *call(* Blockchain.addBlock(..))* specifies that this pointcut will match any call to the *addBlock* method of the Blockchain class. *LedgerAspect* contains a pointcut definition called *ledgerOperations*. The pointcut expression *call(* Blockchain.updateLedger(..))* specifies that this

pointcut will match any call to the *updateLedger* method of the Blockchain component. This provides a way to monitor the operations related to the ledger in a decoupled manner. It logs messages to the console before and after the ledger update operation, serving as an auditing or tracing tool. This aspect can be invaluable for debugging, auditing, or for gathering runtime information without polluting the core business logic code with these concerns. The before advice is executed before any joinpoints matched by the *ledgerOperations* pointcut, and the after advice is executed after any joinpoints matched by the *ledgerOperations* pointcut. In this case, both advice definitions print a message to the standard output method. The role of this aspect is to provide additional behavior to the program by executing code before and after calls to the *updateLedger* method of the blockchain class. This can be useful for logging, debugging, or other purposes. For example, in this case, the aspect logs messages, indicating when the ledger is being updated and when it has been updated. This can help developers understand when and how often this method is being called during program execution. *TransactionAspect* provides logging functionality that crosscuts the core logic of the transaction operations, helping trace the transaction activity in the system.

Figure 3 shows the flow of *BlockAspect–.Block 1*, for instance, symbolizes the group of transactions in which the blockchain system registers transaction data. This process is handled by the *BlockAspect* module of the Block Analysis Layer. Upon the execution of these transactions on the blockchain network, Block 1 is established. *Block 2* signifies the transactions where *AdviceAspect* sets the data attributes, leveraging the Block Aspect Layer. Like its predecessor, this transaction is added to the blockchain network. *Block 2* is then interlinked with Block 1, keeping the continuity and integrity of the chain. Block 3 embodies the transactions that involve requests for block data access, facilitated by the Block Analysis Layer. Upon approval of the transaction request, this transaction is incorporated into the blockchain, resulting in the creation of *Block 3*, which is then linked to *Block 2*. This process is handled by *LedgerAspect*. *Block 4* is the transactions relevant to *BlockAspect*, such as auditing data at *LedgerAspect*. These transactions are recorded on the blockchain as QoSM. Subsequently, a link to Block 3 is established, ensuring the blockchain's ongoing sequence by *LedgerAspect*.

QoSM consists of *QMCriteria* and QoS models, which verify the system property before and after applying the proposed approach and model. *QMCriteria* performs this verification by invoking the *check()* method on the system, which may alter the system model or its properties according to the proposed approach and model. Subsequently, *QMCriteria* compares the system property before and after the changes to measure the impact of the proposed approach and model. The null hypothesis is a statistical concept that states that there is no significant difference between the defined groups. In this case, the null hypothesis would be that applying the proposed approach and model does not significantly affect the context

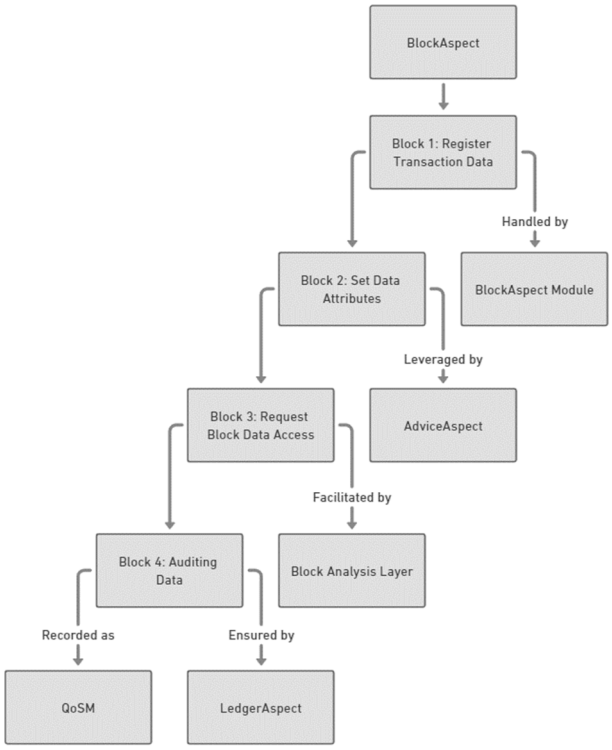

FIGURE 3. BlockAspect flowchart.

data. QoSM handles assessing the QoS of a system, which can be represented as Equation 6.

It works to evaluate the system's functionality by creating a system model and a system property and test the system's reliability. Two sets of context data are produced, being the application's state before and after the model's application. This gives an estimate of the system's efficiency. A timer tracks the execution time of the proposed model, which is used to assess efficiency and figure out system maintainability by calculating the proposed approach and model's cyclomatic complexity. It is a measure of a program's control flow complexity, and it can assess the system's portability by determining the number of environments and platforms that the proposed approach and model support.

C. COMPARING BLOCKASP TO TRADITIONAL OO APPROACHES

When conducting a comparison between *BlockASP* and conventional OO methodologies like Model-Driven Architecture (MDA) [50], [51], various significant metrics are considered, as depicted in Table 1. The metrics encompassed in this framework consist of Functionality, Reliability, Usability, Scalability, Computational Complexity, Efficiency, Transparency, Maintainability, Portability, Trust and Reputation, and Incentive Mechanism.

BlockASP offers several distinct advantages over traditional OO approaches, such as MDA, by harnessing the inherent benefits of blockchain technology. The system offers a decentralized and tamper-resistant approach to managing

data on the blockchain. This provides significant improvements and reliability in the design, analysis, and implementation of blockchain applications. This is accomplished by facilitating a distinct division of responsibilities, fostering a significant level of adaptability for multiple uses, and guaranteeing efficient scalability. *BlockASP* demonstrates exceptional proficiency in managing extensive applications, owing to its decentralized architectural framework. While all systems exhibit similar levels of computational complexity and cost, *BlockASP* stands out due to its notable transparency. The provision of comprehensive visibility of all transactions serves to augment the system's trustworthiness. By harnessing the capabilities of blockchain technology, *BlockASP* ensures the integrity, usability, and scalability of data. The implementation of a QoSM mechanism enables transaction owners to exercise control over the access to block data, granting them the capability to define and enforce access rules and conditions. The assurance of this functionality is provided by the model checking layer. The *BlockASP* layer handles capturing the provenance of every interaction and securely storing the properties and attributes of each transaction. Furthermore, it documents the permissions provided to individuals or entities seeking access to data and enables users to actively control and withdraw these permissions in a flexible manner. *BlockASP* offers a robust mechanism for preserving the provenance and consent information of block data. This mechanism allows for traceability and accountability of data transactions, facilitating auditing processes and ensuring compliance with relevant regulations.

BlockASP demonstrates a high level of portability by effectively integrating with various blockchain systems, thereby facilitating smooth data management across diverse platforms. Trust and reputation can be strengthened through the elimination of intermediaries, which in turn reduces the potential for tampering and provides a transparent audit trail. Furthermore, *BlockASP* incorporates an incentivized mechanism that promotes the sharing of transaction data among users, thereby cultivating a collaborative atmosphere. In contrast, the MDA approach, despite its ability to handle transaction data, gives rise to centralized points of failure. The control that users possess over their transactions within such systems is frequently constrained, resulting in concerns regarding usability and scalability. Furthermore, the MDA methodology exhibits a deficiency in terms of comprehensive transparency, thereby introducing intermediaries such as data brokers. This inclusion of intermediaries may give rise to concerns regarding trust, supplementary expenses, and potential delays in the transactional procedures.

TABLE 1. Comparison of metrics in different *BlockASP* and DMA approaches.

Metric	MDA	BlockASP
Functionality	Medium	High
Reliability	Low	High
Usability	Medium	High
Scalability	Low	High
Computational Complexity	Low	High
Efficiency	Medium	Medium
Transparency	Low	High
Portability	High	High
Maintainability	Low	High
Trust and Reputation	Medium	High
Incentive Mechanism	None	Present

IV. CONCLUSION AND FUTURE WORK

This work proposed an innovative blockchain model checking approach utilizing aspect-oriented programming to better handle dynamism and modularize crosscutting concerns. The *BlockASP* framework was designed comprising four layers - Analysis, Aspect, Model Checking, and Advice to integrate AOP with blockchain model checking. The application of *BlockASP* was demonstrated in a case study of a cryptocurrency system, outlining the workflow from analyzing context data, to statistical model checking using aspects, to quality-of-service assessment. This showcased how BlockASP can enable real-time verification and monitoring of dynamic blockchain behaviors. The superiority of *BlockASP* over traditional OO approaches like MDA was validated through comparison across metrics including functionality, reliability, transparency etc. *BlockASP* leverages blockchain's inherent benefits like decentralization, tamper-resistance, transparency and incentives. Thus, this research contributes to advancing formal verification for blockchain systems by addressing limitations of existing model checking methods.

While the present study establishes a foundational framework with *BlockASP*, several avenues remain for further technical refinement and practical implementation. The integration of machine learning algorithms for real-time anomaly detection could enhance the framework's capability to identify vulnerabilities and improve resilience. From a usability standpoint, developing an intuitive user interface coupled with automated toolchains would streamline the model-checking process, making *BlockASP* accessible to non-expert users. On the architectural side, extending *BlockASP* to be compatible with multi-layer and sharded blockchain designs will add versatility and scalability, thereby enabling its use in increasingly complex systems.

In the context of platform diversity, benchmarking *BlockASP* across multiple blockchain platforms, such as Ethereum, Hyperledger, and Binance Smart Chain, would provide valuable insights into its adaptability and efficiency. Advanced optimizations—such as state space reduction via abstraction or partial order reduction—should also be investigated to expedite the model-checking process.

Lastly, consolidating the theoretical aspects of *BlockASP* by formalizing its algorithmic complexity and computational overhead would lay a robust foundation for future academic and industrial applications. By advancing in these targeted directions, *BlockASP* has the potential to evolve into a universally adopted framework for secure and efficient blockchain model verification. This confluence of aspect-oriented programming, model verification, and blockchain technology stands to gain substantially from these focused initiatives.

ACKNOWLEDGMENT

The author would like to thank Prince Sultan University for their support and cooperation. Authors would like to take the opportunity also to appreciate the support and cooperation given by Southern Illinois University and Yarmouk University.

REFERENCES

[1] S. Khezr, M. Moniruzzaman, A. Yassine, and R. Benlamri, "Blockchain technology in healthcare: A comprehensive review and directions for future research," *Appl. Sci.*, vol. 9, no. 9, p. 1736, Apr. 2019.
[2] H. T. M. Gamage, H. D. Weerasinghe, and N. G. J. Dias, "A survey on blockchain technology concepts, applications, and issues," *Social Netw. Comput. Sci.*, vol. 1, no. 2, pp. 1–15, Mar. 2020.
[3] W. Nam and H. Kil, "Formal verification of blockchain smart contracts via ATL model checking," *IEEE Access*, vol. 10, pp. 8151–8162, 2022.
[4] A. Turrini, "An introduction to quantum model checking," *Appl. Sci.*, vol. 12, no. 4, p. 2016, Feb. 2022.
[5] R. Wang, Z. Tang, J. Gao, Z. Gao, and Z. Wang, "Probabilistic model-checking based reliability analysis for failure correlation of multi-state systems," *Qual. Eng.*, vol. 32, no. 4, pp. 566–582, Oct. 2020.
[6] A. G. Gad, D. T. Mosa, L. Abualigah, and A. A. Abohany, "Emerging trends in blockchain technology and applications: A review and outlook," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 9, pp. 6719–6742, Oct. 2022.
[7] O. A. Abdulhameed, A. Y. Yousuf, and R. H. Abbas, "Aspect oriented programming: Concepts, characteristics and implementation," *Periodicals Eng. Nat. Sci.*, vol. 7, no. 4, pp. 2022–2033, 2020.
[8] P. Boeing, M. Leon, D. Nesbeth, A. Finkelstein, and C. Barnes, "Towards an aspect-oriented design and modelling framework for synthetic biology," *Processes*, vol. 6, no. 9, p. 167, Sep. 2018.
[9] M. Cieslak, A. N. Seleznyova, P. Prusinkiewicz, and J. Hanan, "Towards aspect-oriented functional-structural plant modelling," *Ann. Botany*, vol. 108, no. 6, pp. 1025–1041, Oct. 2011.
[10] R. Laddad, "Aspect-oriented programming will improve quality," *IEEE Softw.*, vol. 20, no. 6, pp. 90–92, Nov. 2003.
[11] 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.
[12] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen, *Systems and Software Verification: Model-Checking Techniques and Tools*. Berlin, Germany: Springer, 2013.
[13] M. Frappier, B. Fraikin, R. Chossart, R. Chane-Yack-Fa, and M. Ouenzar, "Comparison of model checking tools for information systems," in *Formal Methods and Software Engineering*. Berlin, Germany: Springer, 2010, pp. 581–596.
[14] A. M. AlSobeh and S. W. Clyde, "TransJ: An abstract independent-framework for weaving crosscutting concern into distributed transactions," *Comput. Technol. Appl.*, vol. 7, no. 4, p. 173, 2016, doi: 10.17265/1934-7332/2016.04.001.
[15] A. AlSobeh and S. Clyde, "Unified conceptual model for joinpoints in distributed transactions," in *Proc. ICSE*, vol. 14, 2014, pp. 8–15.
[16] A. M. AlSobeh and S. W. Clyde, "Transaction-aware aspects with TransJ: An initial empirical study to demonstrate improvement in reusability," *ICSEA*, vol. 2016, p. 59, Aug. 2016.
[17] A. Cunha, "Bounded model checking of temporal formulas with Alloy," in *Proc. Int. Conf. Abstract State Mach., Alloy, B, TLA, VDM, Z*. Berlin, Germany: Springer, 2014, pp. 303–308.
[18] A. A. Magableh and A. M. R. AlSobeh, "Securing software development stages using aspect-orientation concepts," *Int. J. Softw. Eng. Appl.*, vol. 9, no. 6, pp. 57–71, Nov. 2018.
[19] C. Wagner, N. Jaber, and R. Samanta, "Enabling bounded verification of doubly-unbounded distributed agreement-based systems via bounded regions," *Proc. ACM Program. Lang.*, vol. 7, pp. 172–200, Apr. 2023.
[20] K. M. Sadique, R. Rahmani, and P. Johannesson, "DIdM-EIoTD: Distributed identity management for edge Internet of Things (IoT) devices," *Sensors*, vol. 23, no. 8, p. 4046, Apr. 2023.
[21] Y. He, M. Luo, B. Wu, L. Sun, Y. Wu, Z. Liu, and K. Xiao, "A game theory-based incentive mechanism for collaborative security of federated learning in energy blockchain environment," *IEEE Internet Things J.*, early access, Jun. 5, 2023, doi: 10.1109/JIOT.2023.3282732.
[22] M. A. Hammami, M. Lahami, and A. J. Maalej, "Towards a dynamic testing approach for checking the correctness of Ethereum smart contracts," in *Proc. Int. Conf. Risk Secur. Internet Syst.* Cham, Switzerland: Springer, 2022, pp. 85–100.
[23] P. Gill, I. Ray, A. L. Takami, and M. Tripunitara, "Finding unchecked low-level calls with zero false positives and negatives in Ethereum smart contracts," in *Proc. Int. Symp. Found. Pract. Secur.* Cham, Switzerland: Springer, 2022, pp. 305–321.
[24] R. Galimullin and T. Agotnes, "Coalition logic for specification and verification of smart contract upgrades," in *Proc. Int. Conf. Princ. Pract. Multi-Agent Syst.* Cham, Switzerland: Springer, 2022, pp. 563–572.
[25] M. Almakhour, L. Sliman, A. E. Samhat, and A. Mellouk, "A formal verification approach for composite smart contracts security using FSM," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 35, no. 1, pp. 70–86, Jan. 2023.
[26] G. Alwhishi, J. Bentahar, and A. Elwhishi, "Three-valued model checking smart contract systems with trust under uncertainty," in *Proc. Int. Conf. Deep Learn., Big Data Blockchain*. Cham, Switzerland: Springer, 2022, pp. 119–133.
[27] N. Bertrand, V. Gramoli, I. Konnov, M. Lazic, P. Tholoniat, and J. Widder, "Holistic verification of blockchain consensus," 2022, *arXiv:2206.04489*.
[28] Z. Yang, M. Dai, and J. Guo, "Formal modeling and verification of smart contracts with spin," *Electronics*, vol. 11, no. 19, p. 3091, Sep. 2022.
[29] M. Krichen, M. Lahami, and Q. A. Al-Haija, "Formal methods for the verification of smart contracts: A review," in *Proc. 15th Int. Conf. Secur. Inf. Netw. (SIN)*, 2022, pp. 1–8.
[30] G. S. Ilgi, D. Kayali, P. Olawale, B. Demir Erdem, K. Dimililer, and Y. Kirsal-Ever, "Formal verification for security technologies in the blockchain with artificial intelligence: A survey," in *Proc. Innov. Intell. Syst. Appl. Conf. (ASYU)*, Sep. 2022, pp. 1–6.
[31] X. Hai and J. Liu, "PPDS: Privacy preserving data sharing for AI applications based on smart contracts," in *Proc. IEEE 46th Annu. Comput., Softw., Appl. Conf.*, Jun. 2022, pp. 1561–1566.
[32] R. B. Fekih, M. Lahami, M. Jmaiel, A. Ben Ali, and P. Genestier, "Towards model checking approach for smart contract validation in the EIP-1559 Ethereum," in *Proc. IEEE 46th Annu. Comput., Softw., Appl. Conf. (COMPSAC)*, Jun. 2022, pp. 83–88.
[33] P. Bose, D. Das, Y. Chen, Y. Feng, C. Kruegel, and G. Vigna, "SAILFISH: Vetting smart contract state-inconsistency bugs in seconds," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2022, pp. 161–178.
[34] R. Heckel, Z. Erum, N. Rahmi, and A. Pul, "Visual smart contracts for DAML," in *Proc. Int. Conf. Graph Transformation*. Cham, Switzerland: Springer, 2022, pp. 137–154.
[35] W. S. Park, H. Lee, and J.-Y. Choi, "Formal modeling of smart contract-based trading system," in *Proc. 23rd Int. Conf. Adv. Commun. Technol. (ICACT)*, Feb. 2021, pp. 48–52.
[36] C. L. Rodrigues, D. D. S. Guerrero, and J. C. A. de Figueiredo, "Model checking in object-oriented Petri nets," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, Oct. 2004, pp. 4977–4982.
[37] M. Ceška, L. Haša, and T. Vojnar, "Partial-order reduction in model checking object-oriented Petri nets," in *Computer Aided Systems Theory—EUROCAST*. Berlin, Germany: Springer, 2003, pp. 265–278.
[38] S. Porru, A. Pinna, M. Marchesi, and R. Tonelli, "Blockchain-oriented software engineering: Challenges and new directions," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2017, pp. 169–171.
[39] N. Ubayashi and T. Tamai, "Aspect-oriented programming with model checking," in *Proc. 1st Int. Conf. Aspect-Oriented Softw. Develop.*, Apr. 2002, pp. 148–154.
[40] M. Kallel, Y. Lahbib, A. Baganne, and R. Tourki, "Monitoring transaction level SystemC models using a generic and aspect-oriented framework," *Int. J. Comput. Aided Eng. Technol.*, vol. 4, no. 3, pp. 229–249, 2012.
[41] W. Ruanthong and P. Muenchaisri, "Model checking for aspect-oriented software evolution," in *Proc. 4th WSEAS Int. Conf. Softw. Eng., Parallel Distrib. Syst.*, 2005, pp. 1–6.
[42] M. A. P. Toledano, A. N. Martinez, J. M. M. Rodriguez, and C. Canal, "Making aspect oriented system evolution safer," in *Proc. ECOOP Workshop Reflection, AOP Meta-Data Softw. Evol.*, 2006, pp. 23–34.
[43] Y. Koga and Y. Tanabe, "A practical method for software model checking using AOP," IEICE, China, Tech. Rep., 2012.
[44] D.-X. Xu, O. El-Ariss, W.-F. Xu, and L.-Z. Wang, "Aspect-oriented modeling and verification with finite state machines," *J. Comput. Sci. Technol.*, vol. 24, no. 5, pp. 949–961, Sep. 2009.
[45] A. A. AlSobeh, "Improving reuse of distributed transaction software with transaction-aware aspects," *ProQuest*, vol. 2, pp. 185–191, Mar. 2015.

[46] A. M. R. AlSobeh and A. A. Magableh, "An aspect-oriented with BIP components for better crosscutting concerns modernization in IoT applications," in *Proc. Comput. Sci. Inf. Technol.*, Aug. 2018, pp. 1–11.
[47] A. Nusayr and J. Cook, "Using AOP for detailed runtime monitoring instrumentation," in *Proc. 7th Int. Workshop Dynamic Anal.*, 2009, pp. 8–14.
[48] X. Wen, H. Zheng, and Z. Yang, "A formal method for software architecture analysis based on aspect orientation," *Int. J. Wireless Mobile Comput.*, vol. 14, no. 2, pp. 191–196, 2018.
[49] J. W. Yoder, F. Balaguer, and R. Johnson, "From analysis to design of the observation pattern," *Urbana*, vol. 51, p. 61801, May 2017.
[50] T. Górski and J. Bednarski, "Applying model-driven engineering to distributed ledger deployment," *IEEE Access*, vol. 8, pp. 118245–118261, 2020.
[51] M. Fahmideh, J. Grundy, A. Ahmed, J. Shen, J. Yan, D. Mougouei, P. Wang, A. Ghose, A. Gunawardana, U. Aickelin, and B. Abedin, "Engineering blockchain based software systems: Foundations, survey, and future directions," 2021, *arXiv:2105.01881*.
[52] A. M. R. AlSobeh, R. Hammad, and A. K. Al-Tamimi, "A modular cloud-based ontology framework for context-aware EHR services," *Int. J. Comput. Appl. Technol.*, vol. 60, no. 4, pp. 339–350, 2019.
[53] A. M. AlSobeh, S. AlShattnawi, A. Jarrah, and M. M. Hammad, "Weavesim: A scalable and reusable cloud simulation framework leveraging aspect-oriented programming," *Jordanian J. Comput. Inf. Technol.*, vol. 6, no. 2, pp. 1–20, 2020.
[54] A. Alsobeh and A. Shatnawi, "Integrating data-driven security, model checking, and self-adaptation for IoT systems using BIP components: A conceptual proposal model," in *Proc. Int. Conf. Adv. Comput. Res.* Cham, Switzerland: Springer, 2023, pp. 533–549.
[55] A. M. R. AlSobeh and A. A. Magableh, "Architectural aspect-aware design for IoT applications: Conceptual proposal," *Int. J. Comput. Sci. Inf. Technol.*, vol. 10, no. 6, pp. 01–11, Dec. 2018.
[56] A. A. Magableh and A. M. R. AlSobeh, "Aspect-oriented software security development life cycle (AOSSDLC)," in *Proc. Comput. Sci. Inf. Technol.*, Aug. 2018, pp. 25–26.
[57] A. M. R. Alsobeh, A. A. A. R. Magableh, and E. M. AlSukhni, "Runtime reusable weaving model for cloud services using aspect-oriented programming," *Int. J. Web Services Res.*, vol. 15, no. 1, pp. 71–88, Jan. 2018.
[58] H. Afzaal, M. Imran, and M. U. Janjua, "Formal verification of fraud-resilience in a crowdsourcing consensus protocol," *Comput. Secur.*, vol. 131, Aug. 2023, Art. no. 103290.

ANAS M. R. ALSOBEH was born in Jordan. He received the B.Sc. and M.Sc. degrees in computer information systems from Yarmouk University, Jordan, in 2007 and 2010, respectively, and the Ph.D. degree in computer science from Utah State University, USA, in 2015. With a keen interest in academia and a passion for information technology, he joined Southern Illinois University Carbondale, IL, USA, as an Assistant Professor. He currently plays a pivotal role with the Information Technology Program. Prior to this, he lent their expertise to the Department of Information Systems, Yarmouk University, even serving as the Director of the Refugees, Displaced Persons, and Forced Migration Studies Center. His research interests include software design and modeling, data analysis, web applications, cybersecurity analysis, cloud computing, and the social analysis of IT. He has a distinguished record of publications and presentations, contributing to the body of knowledge in his field. Some of his noteworthy publications include *The Effectiveness of a Counseling Program Based on Psychosocial Support* (Springer, 2023), *Integrating Data-Driven Security, Model Checking, and Self-adaptation for IoT Systems Using BIP Components: A Conceptual Proposal Model* (Springer, 2023), and *A Modular Cloud-Based Ontology Framework for Context-Aware EHR Services* (InderScience, 2019). His research efforts have been recognized and supported through various national and international research grants, including several from H2020, HOPES, Erasmus+, and Nuffic. As a dedicated lifelong learner, he recently certified with the Data Science and Machine Learning: Making Data-Driven Decisions Program, Institute for Data, Systems, and Society (IDSS), MIT, and the AWS Cloud Practitioner Program by Amazon Web Services. His current research interests include exploring and advancing the cybersecurity systems design and modeling techniques by AOP. He is an esteemed member of the Academic and Research Community.

AWS A. MAGABLEH received the bachelor's degree in software engineering from Hashemite University, in 2006, the master's degree in software engineering from University Malaysia (UM), in 2008, and the Ph.D. degree from the National University of Malaysia (UKM), in 2015. He is currently an Associate Professor with the Information Systems Department, Yarmouk University. He is very passionate about learning and development (L&D). He has immersed in the training industries with Nokia, Microsoft, and Huawei. He has many publications in international conferences and refereed journals; these papers focus on software analysis and design, AOP, OO, Web, and UML.