Intelligent Multimedia Processing & Communication Systems Journal

IMPCSJ

J IMPCS (2025) 21: 13-24
DOI 10.71856/IMPCS.2025.1217575

Research Paper

SmartSLA: A Graphical Modeling-Based Approach to SLA Management with Automatic Code Generation on Blockchain Platforms

Jalal Fazilat[1], Leila Samimi Dehkordi[2*], Delaram Nikbakht Nasrabadi[3], Abbas Horri[4]

1. MSc. Student, Department of Computer Engineering, Shahrekord University, Shahrekord, Iran.
2. Assistant Professor, Department of Computer Engineering, Shahrekord University, Shahrekord, Iran.
**Corresponding Author*, samimi@sku.ac.ir
3. Master Graduated, Department of Computer Engineering,, Shahrekord University, Shahrekord, Iran.
4. Assistant Professor, Department of Computer Engineering, Shahrekord University, Shahrekord, Iran.

Article Info

Article history:
Received: 11 Aug 2025
Accepted: 21 Sep 2025

Keywords:
Blockchain,
Cloud Computing,
Model Driven Engineering,
Service Level Agreement,
Smart Contract.

ABSTRACT

SmartSLA is an innovative model-driven framework that automates the generation and deployment of blockchain-based Service Level Agreements (SLAs) in cloud environments. It addresses the limitations of manual and centralized SLA management—such as lack of transparency, high complexity, and susceptibility to human error—by integrating Model-Driven Engineering (MDE) with blockchain technology. The framework consists of three core components: (1) an Ecore-based metamodel that formally defines SLA elements and policies, (2) a graphical modeling editor for intuitive SLA design, and (3) an automated code generator that produces executable Solidity contracts for Ethereum deployment. The framework's applicability was evaluated through eight industrial case studies across diverse cloud domains, including networking, storage, IoT, and disaster recovery. Results demonstrate full automation of the SLA lifecycle with substantial reduction in design effort and consistent model-to-code transformation accuracy. Structural comparisons with five existing modeling languages confirm balanced design quality, achieving moderate maintainability, high understandability, and sufficient extensibility. By bridging SLA specification and blockchain enforcement, SmartSLA provides a unified, scalable, and transparent solution for automated SLA management, strengthening operational reliability and advancing the integration of MDE and blockchain in industrial cloud ecosystems.

I. Introduction

In the past decade, cloud computing has emerged as a critical infrastructure for modern industries and service-oriented enterprises, offering unprecedented scalability, flexibility, and cost efficiency [1]. As organizations increasingly rely on cloud platforms, ensuring service quality, transparency, and accountability has become a fundamental requirement [2].

Service Level Agreements (SLAs) are the primary mechanism for formalizing quality expectations—such as availability, performance, and reliability—between providers and consumers [3]. However, traditional SLA management remains largely manual and centralized, limiting its effectiveness in dynamic and distributed environments [4, 5]. These limitations reduce transparency, hinder automation, and increase the risk of operational failures in industrial applications.

Blockchain-based smart contracts have been proposed as a decentralized and tamper-proof approach to enforce SLAs [6]. While promising, their practical adoption is hindered by the technical expertise required for smart contract programming and deployment [7]. Bridging the gap between high-level SLA specification and low-level blockchain enforcement therefore remains a critical challenge.

To address this challenge, we propose **SmartSLA**, an innovative model-driven framework that automates the transformation of SLA specifications into executable smart contracts. Leveraging Model-Driven Engineering (MDE), SmartSLA enables SLA logic to be expressed at a high level of abstraction and automatically translated into Solidity code. The framework comprises three key components: (1) an Ecore-based metamodel capturing SLA concepts, (2) a graphical modeling environment that facilitates SLA design for both technical and non-technical users, and (3) an automated code generator targeting Ethereum-compatible blockchains. This integration reduces programming errors, promotes standardization, and supports reliable SLA enforcement across diverse domains.

Unlike prior studies that focus exclusively on either SLA modeling languages or smart contract description frameworks, SmartSLA unifies both perspectives into a single, end-to-end solution. To evaluate its novelty and effectiveness, the study is guided by the following research questions:

RQ1: Can SmartSLA effectively automate the generation of smart contracts from high-level SLA specifications across diverse cloud domains?

RQ2: To what extent does SmartSLA reduce design effort and improve reliability compared to manual or template-based approaches?

RQ3: How does the SmartSLA metamodel compare with existing SLA and smart contract languages in terms of maintainability, understandability, and extensibility?

By addressing these questions, the paper highlights the scientific and practical contributions of SmartSLA to the automation of SLA lifecycle management. The remainder of the paper is organized as follows: Section II introduces the conceptual background, Section III reviews related work, Section IV presents the SmartSLA framework, Section V discusses evaluation results, and Section VI concludes with future directions.

II. Background

This section lays the conceptual foundation of the SmartSLA framework by examining its three core pillars: smart contracts, service-level agreements, and model-driven engineering.

A. Smart Contracts

Smart contracts are self-executing programs deployed on blockchain platforms such as Ethereum. They automatically enforce predefined business rules without the need for intermediaries, leveraging blockchain immutable and distributed ledger to ensure transparent and verifiable execution [6, 8]. In SLA enforcement, smart contracts can continuously monitor metrics such as uptime, latency, or error rates, and autonomously trigger actions—like applying penalties or initiating alerts—upon detecting violations [9].

Although languages such as Solidity enable the development of smart contract logic, writing such contracts remains prone to programming errors and requires high technical proficiency [10, 11]. This complexity often limits their adoption in real-world industrial and service settings.

In the SmartSLA framework, smart contracts form the core enforcement layer, and are generated automatically from high-level models, reducing human error and technical barriers while ensuring SLA terms are enforced in a trustworthy and decentralized manner.

B. Service Level Agreements (SLAs)

SLAs are formal contracts that define the service quality expectations and obligations between cloud providers and consumers [3]. These agreements typically cover quantifiable metrics such as availability, response time, and throughput, as well as enforcement policies and penalty structures [2].

However, conventional SLA management is still heavily manual, centralized, and reactive, making it slow to detect and respond to violations—especially in multi-tenant and hybrid cloud environments [4]. Such inefficiencies reduce transparency and may erode trust between service parties [5]. The increasing complexity of distributed systems has amplified the need for SLA enforcement mechanisms that are automated, auditable, and trustworthy [12, 13].

The SmartSLA framework addresses this need by reengineering SLA structure through formal modeling, thus promoting automation, improving clarity, and enabling integration with intelligent service platforms.

C. Model-Driven Engineering (MDE)

Model-Driven Engineering (MDE) is a development methodology that abstracts software systems into formal models to facilitate automated code generation, structural analysis, and design validation [7]. Central to MDSE is the creation of a metamodel, which formally defines domain-specific constructs and their relationships—for instance, SLA components such as parties, performance metrics, and compliance policies [14].

By enabling high-level design and transformation into executable code, MDSE empowers domain experts to define sophisticated service agreements without needing deep programming knowledge [15]. In SmartSLA, this is realized through an Ecore-based metamodel and a graphical modeling tool built with Sirius [16], enabling the intuitive construction of SLA models.

These models are then automatically translated into executable Solidity code [17], bridging the conceptual and implementation layers of SLA enforcement. This approach reduces development effort, enhances reliability, and ensures consistency across SLA definitions deployed in blockchain-backed cloud infrastructures.

III. Related works

The integration of blockchain technology with Service Level Agreement (SLA) management has attracted significant attention in recent years. This section reviews state-of-the-art approaches related to blockchain-based SLA enforcement, smart contract modeling, and model-driven frameworks, highlighting their contributions and limitations in relation to the SmartSLA framework.

Nguyen et al. [18] introduced PenChain, a blockchain-based platform that enables automatic SLA enforcement with embedded penalty rules. Their system uses smart contracts to execute SLAs and ranks service providers based on compliance and reputation, demonstrating its effectiveness in precision agriculture and automotive industries. Alzubaidi et al. [19] proposed a formal SLA representation model, IRAFUTAL, enabling blockchain-based SLA lifecycle management via Hyperledger Fabric. Their framework supports negotiation, monitoring, billing, and enforcement, addressing the complexity of multi-phase SLA management.

In response to the lack of standardization in asynchronous service agreements, Oriol et al. [20] proposed a quality model aligned with ISO/IEC 25010 and a domain-specific language for asynchronous SLAs using WS-Agreement standards. Their solution, integrated into AsyncAPI and extended through tooling, facilitates SLA specification and enforcement in IoT and cyber-physical systems.

Maatougui et al. [21] proposed a component-based contractual approach for designing and formally specifying self-adaptive systems with respect to Quality of Service (QoS) contracts. Their method leverages Model-Driven Engineering to model system structure and behavior and employs the Maude formal language to generate executable specifications that enable runtime monitoring and adaptation. By clearly separating user-defined QoS requirements from internal system parameters, their model simplifies complexity and enhances reusability. They validated their approach through a firefighting system case study, demonstrating how adaptation strategies are triggered to ensure continuous QoS compliance in dynamic contexts.

Cambronero et al. [22] introduced CloudCost, a UML-based profile designed to model cloud infrastructures and user interactions with respect to Service Level Agreements, aiming to improve cloud provider profitability. Their approach distinguishes between regular and high-priority users, integrating parameters such as resource costs, discounts, and compensations into the SLA model. They also developed MSCC, a modeling tool that supports the creation, validation, and simulation of cloud scenarios using the Simcan2Cloud simulator. Through a comprehensive case study involving different workloads and infrastructure configurations, they demonstrated how pricing strategies and user types influence provider income, offering insights into effective SLA-driven resource management.

Several survey-based studies provide broader overviews. Saghaier et al. [23] and Hamdi et al. [24] analyzed SLA monitoring solutions across various sectors, such as cloud computing and 6G networks, underscoring the potential of smart contracts in detecting violations and managing compensation. Mahapatra et al. [25] proposed a secure blockchain framework for IoT-Fog-Cloud environments, addressing critical challenges such as trust and authentication.

Azzahra and Nugraha [26] implemented a smart contract-based SLA management system for IT services in higher education, illustrating benefits such as improved monitoring and reduced processing time, albeit with higher operational costs. Tang et al. [27] extended this idea to the tourism industry with TSLA, a framework that leverages oracles and smart contracts to detect provider misbehavior and enforce penalties.

Souei et al. [28] developed a distributed directory for smart contracts based on a unified description language (UDL-SC), enabling semantic-based search and selection of contracts based on legal, performance, and gas usage attributes. In another model-driven engineering approach, Hamdaqa et al. [10] introduced a reference model for smart contracts in 2020 by analyzing the characteristics of three platforms: Hyperledger Composer, Azure Blockchain Workbench, and Ethereum. The proposed model was designed as a framework that enables developers to model

and generate the structural code of smart contracts across different blockchain platforms. To validate their approach, they applied the framework to three case studies, demonstrating its capability to generate executable code for the target platforms. Furthermore, the authors extended the reference model into a more advanced version, iContractML 2.0 [11]. In addition to supporting the generation of smart contract behavior, this version also facilitates the creation of structural and deployment artifacts by employing templates for commonly used functions.

In cloud service discovery, Nabli et al. [13] proposed the Cloud Services Description Ontology (CSDO) using Linked-USDL, supporting the publication and selection of cloud services. Similarly, Kamel et al. [12] applied reactive system theory to model and verify SLA lifecycle behavior in cloud-based environments, emphasizing formal verification.

Furthermore, studies like those by Battula et al. [29] on fog computing and Cedillo et al. [30] in AAL environments confirm the growing trend toward SLA automation using blockchain. However, these solutions often focus on specific domains, lack comprehensive modeling environments, or require significant technical expertise.

Makwe et al. [31] propose a broker-based SLA framework for IaaS clouds that improves provider selection and monitoring, reducing violation rates and enhancing resource utilization. Booth et al. [32] present a blockchain-based library that generates smart contracts from SLAs for IoT monitoring, ensuring reliable violation detection in healthcare scenarios. Muntaha et al. [33] design a hybrid blockchain framework for SLA management in 5G networks, combining Hyperledger and Ethereum to improve resource sharing and enforcement. Solis et al. [34] introduce MICAAL, a DSL for modeling microservices in Ambient Assisted Living systems, supporting modularity and scalability in IoT healthcare.

Unlike previous work, SmartSLA provides an integrated, model-driven framework featuring a metamodel, a graphical modeling editor based on Sirius, and automated smart contract code generation in Solidity. This design bridges the gap between SLA specification and deployment, making the process accessible to non-experts while ensuring consistency, extensibility, and operational automation. Table I provides a comparison of related work with our proposed solution (SmartSLA).

TABLE I Comparison of Related Work

Study	Application Domain	Blockchain	Modeling Type
[10, 11]	General	✓	Ecore to Solidity
[12]	Cloud	×	Reactive systems
[13]	Cloud	×	Linked-USDL
[18]	Precision Agriculture, Automotive Manufacturing	✓	Smart Contracts
[19]	IoT	✓	Formal Modeling
[20]	Asynchronous SLA	×	DSL, AsyncAPI
[21]	Self-adaptive systems	×	Model-Driven Engineering (Maude)
[22]	Cloud Infrastructure	×	UML, Simcan2Cloud
[23]	Multiple domains (Cloud, IoT, 6G)	✓	Survey / Comparative Analysis
[24]	General (Survey-based)	✓	Survey, Comparative Analysis
[25]	IoT-Fog-Cloud	✓	Security Framework
[26]	Higher Education	✓	Smart Contracts
[27]	Tourism	✓	Smart Contracts + Oracle
[28]	General	✓	UDL-SC
[29]	Fog Computing	✓	Various
[30]	Ambient Assisted Living (AAL)	✓	Various
[31]	IaaS Cloud, CSP Selection	×	Broker model
[32]	IoT, Patient Monitoring	✓	Java Library
[33]	5G Resource Sharing	✓	Game theory
[34]	Ambient Assisted Living	×	DSL, Ecore2Service
SmartSLA	Cloud SLA	✓	Graphical model to Solidity

IV. SmartSLA: A Model-Driven Framework for SLA Code Generation

In complex and dynamic cloud-based environments, managing Service Level Agreements is essential not only for ensuring service quality but also for maintaining operational continuity and cost efficiency—key concerns in industrial and service domains. However, conventional approaches to SLA management often suffer from fragmentation, manual configuration steps, and limited adaptability to rapidly evolving service structures.

To address these limitations, we introduce SmartSLA, a model-driven framework that facilitates the standardized and automated definition, customization, and execution of SLAs. The framework consists of three key components: a **metamodel** that formally defines the structure of smart SLAs, a **graphical editor** that simplifies SLA modeling for technical and non-technical users alike, and a **code generation engine** that automatically produces executable smart contracts for deployment in blockchain environments.

By integrating these components, SmartSLA supports seamless SLA management across cloud-based platforms, and promotes reduced design complexity, higher reusability, and quicker deployment—all of which are highly relevant for industrial decision-making systems, enterprise platforms, and service coordination in large-scale infrastructures.

A. Core Concepts

SmartSLA is founded on three core concepts that underpin its modeling and automation capabilities:

- Metamodeling: At the heart of the framework lies a formal Ecore-based metamodel that serves as the foundation for the domain-specific modeling language. This metamodel defines the core elements of SLAs—including service objectives, penalty rules, compliance terms, and provider-consumer roles—in a structured and extensible way. It enables model validation and reuse across diverse service settings.
- Smart Contracts: Using blockchain technology, SmartSLA converts SLA models into smart contracts, which are self-executing programs that enforce SLA conditions such as uptime guarantees or compensation clauses. This ensures transparency, auditability, and operational reliability without requiring manual oversight [6].
- Model Transformation and Code Generation: Through model-to-text transformation techniques, SmartSLA automatically generates Solidity code for deployment on Ethereum-compatible platforms. This reduces development time and technical effort while allowing organizations to incorporate SLA enforcement into automated, industrial-grade workflows.

By combining these pillars, the SmartSLA framework bridges the gap between high-level service design and low-level contract execution, providing a robust, industry-aligned tool for SLA management in distributed cloud infrastructures.

B. The SmartSLA Metamodel

The formal definition outlining the structure and various constituent parts of a smart SLA is provided by the SmartSLA metamodel, which aligns with the Ecore standard. This metamodel offers a precise outline of the classes, detailing their relationships, assigned responsibilities, and defining attributes. It essentially forms the vital foundation required for building valid SmartSLA models. A visual depiction of the proposed SmartSLA metamodel structure can be found in Fig. 1.

The SLAContract class serves as the root of the metamodel, representing the main container for all SLA-related elements. It stores general contract metadata (such as name, duration, and version) and aggregates all core components, including service objectives, parties, constraints, enforcement mechanisms, resolution policies, and audit logs.

Party represents the actors involved in the agreement—namely the provider and consumer. It includes attributes such as address and reputationScore, and is connected to the contract, notifications, and change requests.

The abstract class SLAConstraint generalizes various SLA condition types. Its concrete subclasses include:

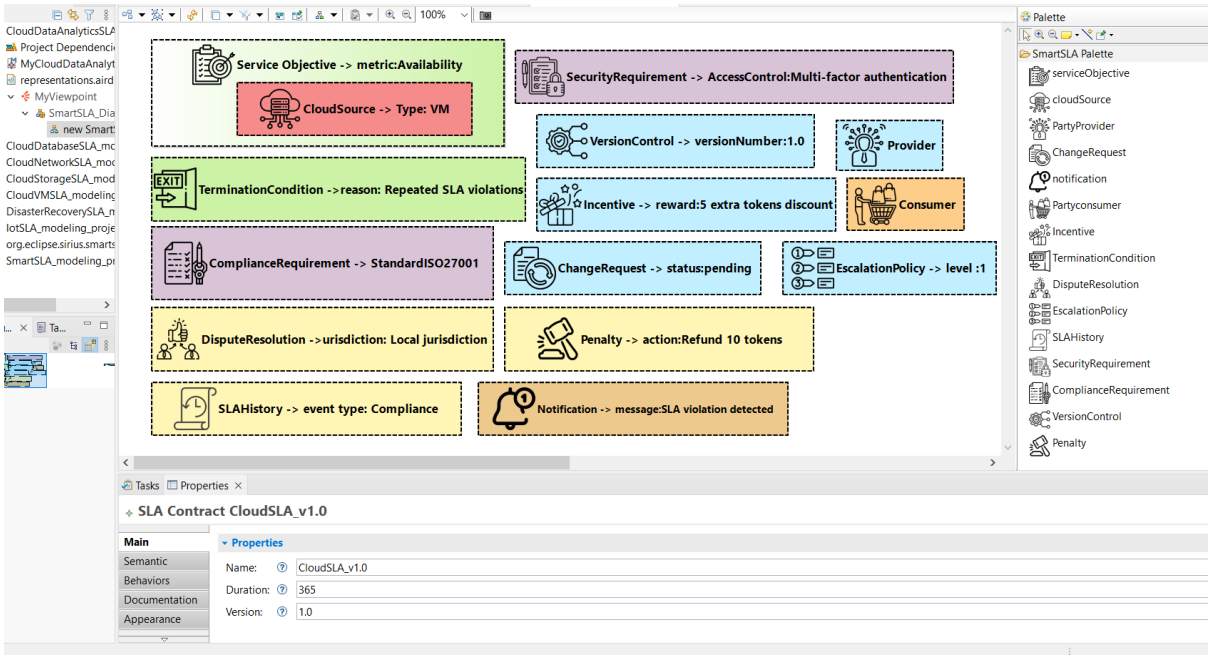

Fig. 2. The Graphical Editor Implemented in Sirius

- SecurityRequirement, which defines encryption, access control, and audit needs;
- ComplianceRequirement, which specifies external standards and verification methods;
- ServiceObjective, which defines measurable targets (e.g., performance or availability) and is linked to a specific CloudSource, describing the underlying infrastructure (such as a VM, storage, or IoT device) using the CloudSourceType enumeration.

The EnforcementElement abstract class captures rule-based enforcement logic and is inherited by:

- Penalty, which applies predefined actions upon contract violation;
- Incentive, which defines rewards for compliance;
- Notification, which alerts parties under specified conditions.

To model SLA dynamics and evolution, the ChangeRequest class captures proposed modifications, while VersionControl maintains version history. The SLAHistory class records past events (e.g., violations or updates) and is linked to the contract via audit logs.

Conflict and termination handling are modeled through the abstract ResolutionElement, which is extended by:

- TerminationCondition, defining reasons and consequences for SLA cancellation;
- DisputeResolution, specifying procedures and legal aspects for resolving conflicts;
- EscalationPolicy, indicating how service breaches are escalated based on severity and response time.

Overall, the SmartSLA metamodel provides a modular and extensible structure that covers key aspects of SLA management, supporting formal modeling, enforcement automation, and traceability.

C. Graphical Editor

To promote accessibility for users with limited technical or programming backgrounds, a graphical modeling editor was developed as part of the SmartSLA framework. Serving as an interactive interface for SLA model creation, the editor enables users to define, visualize, and manage SLA structures directly based on the SmartSLA metamodel. It was implemented using the Sirius framework, which supports the development of domain-specific graphical editors within the Eclipse ecosystem.

The editor provides a user-friendly, drag-and-drop environment where users can construct SLA models by placing visual elements—such as Party, ServiceObjective, or Penalty—onto a design canvas. Each element corresponds directly to a metamodel class and can be configured using a dedicated properties panel. For instance, a user can define a ServiceObjective with a response time metric and a performance threshold (e.g., 300ms), then link it to a specific CloudSource, such as a serverless function.

To streamline the modeling process, the editor includes a **tool palette** containing all available SLA elements, each with a distinctive icon for clarity. Users define relationships between elements visually, while the editor automatically enforces conformance with the underlying metamodel. Additionally, validation mechanisms ensure that models are semantically and structurally correct, and the editor can support the automatic generation of preliminary smart contract code from compliant models.

This visual interface significantly reduces complexity, enhances modeling efficiency, and encourages broader adoption of formal SLA specification methods across industrial and service-oriented domains. Fig. 2 illustrates the editor's interface and its alignment with the core SmartSLA concepts.

Built on the Sirius framework, the editor provides notable flexibility for tailoring visual representations and adeptly handles differing levels of complexity. Its key benefits include significantly reducing design complexity for users without a technical background, substantially accelerating the SLA creation process, and ensuring strict adherence to the SmartSLA metamodel, which is vital for guaranteeing the structural soundness of the resulting models.

D. Integration and Code Generation

The strength of the SmartSLA framework really lies in how seamlessly the metamodel and the graphical editor are integrated. The metamodel formally defines the abstract syntax of the modeling language. In parallel, the editor provides a user-friendly interface, as a concrete syntax of the SmartSLA language, allowing the creation of instances (the models). The underlying model-driven framework (Sirius/Ecore) facilitates this integration, ensuring the workflow is cohesive and smooth.

Users design an SLA visually in the editor, and each graphical element corresponds directly to an instance of a metamodel-defined class or relationship. The editor enforces the rules of the metamodel using validation, stopping users from creating structurally incorrect or inconsistent SLA definitions. This model can then be utilized for automated code generation.

The conversion of a SmartSLA model into executable smart contract code is usually carried out using model-to-text languages like Epsilon Generation Language (EGL). As depicted in Fig. 3, this automated generation approach greatly lessens both the manual effort involved and the chance of introducing errors during the creation of deployable smart contracts from the higher-level SLA definition.

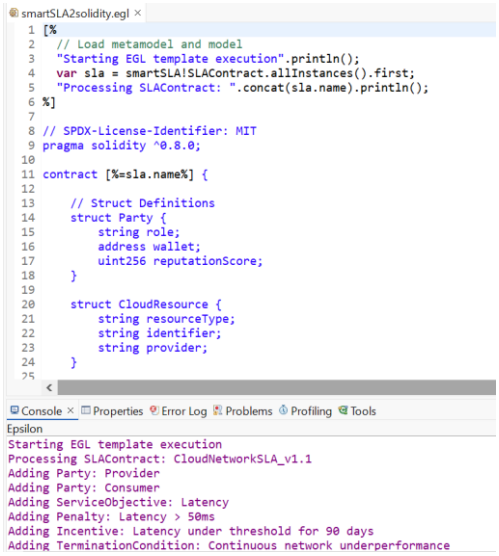

Fig.3. Transformation from SmartSLA model to Smart Contract Code via EGL program.

This integrated approach reduces design errors, increases efficiency, and makes SLA modeling accessible to a wider range of stakeholders, from technical developers to cloud service managers.

E. SmartSLA Model Deployment

The final stage of the SmartSLA framework focuses on the deployment of automatically generated smart contracts onto blockchain platforms for real-time SLA enforcement [8]. This phase transforms the modeled SLA specifications into operational assets that can autonomously monitor service performance and enforce contractual terms without manual intervention [9].

Smart contracts generated from the SmartSLA models can be deployed on platforms such as Ethereum, where they continuously track compliance with defined service objectives. Upon detecting a violation or fulfillment, the contracts trigger predefined actions—such as penalties, notifications, or rewards—ensuring that SLA terms are enforced in a consistent and transparent manner.

The deployment environment includes integrated audit logging mechanisms, enabling full traceability and post-execution verification of SLA activities [11]. Designed with ease of integration in mind, the deployment process requires minimal manual configuration, making it suitable for seamless adoption in industrial cloud infrastructures [15].

This deployment capability completes the end-to-end lifecycle of SLA management—spanning from abstract modeling and validation to executable contract generation and live operation. By supporting fully automated, traceable, and scalable SLA enforcement, SmartSLA offers a practical and efficient solution for managing service reliability in modern distributed systems.

V. Evaluation

To assess the effectiveness of the SmartSLA framework, a two-fold evaluation was conducted focusing on both its applicability in diverse cloud service domains (Section 5.1) and its structural modeling characteristics in comparison with existing SLA metamodels (Section 5.2). This evaluation aims to demonstrate the framework's practical utility, flexibility, and degree of automation in supporting SLA definition and enforcement across industrial-scale service environments.

A. Applicability Evaluation

To answer RQ1 and RQ2, eight case studies were modeled and automatically transformed into Solidity contracts, confirming the end-to-end automation capability of the SmartSLA framework. These case studies were chosen to reflect real-world scenarios where SLA modeling and automation are essential for operational efficiency, reliability, and service quality assurance. Each case involved designing a complete SLA model using the SmartSLA graphical editor, validating its structure through EMF representation, and automatically generating executable Solidity smart contracts through an EGL-based code generator.

The use cases are as follows:

1. Cloud Data Analytics: SLAs were designed to manage data processing quality, focusing on throughput, latency, and accuracy. This domain is critical for real-time decision-making and data-intensive industrial applications such as predictive maintenance or fraud detection in financial services.

2. Cloud Database: Here, the SLA addressed transactional integrity, availability, and recovery time. Database services form the backbone of many enterprise applications and require high reliability and performance guarantees, especially in sectors like banking and e-commerce.

3. Cloud Network: An SLA centered on latency and packet loss, suitable for time-sensitive systems such as VoIP, video conferencing, and industrial IoT communication frameworks. This case demonstrates the need for tightly controlled service metrics in latency-critical environments.

4. Cloud Storage: We modeled SLAs prioritizing data durability, access time, and encryption compliance. These SLAs are highly relevant in legal, medical, and financial industries where secure data retention and retrieval are paramount.

5. Cloud Infrastructure (IaaS): SLAs focused on virtual machine availability and resource provisioning speed. This case supports general-purpose compute environments, which are foundational for most enterprise digital infrastructure.

6. Disaster Recovery: Key SLA terms included Recovery Point Objective (RPO) and Recovery Time Objective (RTO). This use case is critical in sectors requiring business continuity, such as healthcare, public safety, and enterprise risk management.

7. Internet of Things (IoT): The SLA included metrics for sensor responsiveness and data availability. This case

supports industrial and smart city environments where high uptime and real-time sensor data are essential.

8. Serverless Computing: SLAs were modeled around function execution time, scalability, and cost optimization. Serverless environments are vital in cost-sensitive deployments like mobile backends, APIs, and microservices.

In all cases, users leveraged the graphical interface to construct SLA models using SmartSLA's drag-and-drop tooling.

Table II presents the numbers of classes, attributes, and relationships that were defined in each case model. Importantly, each model was used to generate executable smart contract code via a 265-line EGL script. This automated translation from graphical SLA design to Solidity code showcases the system's end-to-end automation capability. While the number of lines of code (LOC) varies across case studies, the consistent generation of deployable contracts underlines the robustness of the approach.

TABLE II Comparison of SLA Model Complexity Metrics across Eight Case Studies

Case Study	#C	#R	#A	LOC
Data Analytics	18	13	57	125
Cloud Database	16	11	51	125
Cloud Network	14	9	41	120
Cloud Storage	14	9	41	120
Cloud Infrastructure	14	9	41	120
Disaster Recovery	15	10	46	135
Internet of Things	18	13	57	135
Serverless Computing	17	12	54	135

The variation in generated LOC corresponds directly to model complexity. Regarding RQ2, the evaluation shows that SmartSLA significantly reduces manual coding effort and minimizes human errors by automating contract generation.

To further illustrate the modeling effort and complexity across case studies, Fig. 4 presents a comparative bar chart visualizing the number of classes, relationships, attributes, and generated lines of code (LOC) for each cloud domain. The diagram reveals that case studies such as IoT, Cloud Analytics, and Serverless Computing exhibit higher structural complexity, with correspondingly increased LOC in their generated smart contracts. Conversely, scenarios like Cloud Network and Cloud Storage show relatively lower complexity and shorter generated code. This visualization supports the quantitative findings presented in Tables 2, demonstrating the SmartSLA framework's capacity to scale across a range of modeling demands—from minimal to highly detailed SLA definitions—while maintaining automation consistency.

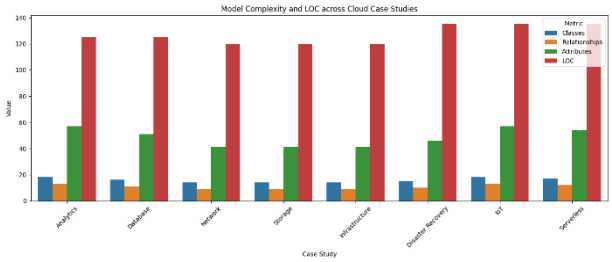

Fig. 4. Comparative visualization of model complexity and generated code size across eight case studies.

This comprehensive applicability evaluation confirms that SmartSLA is not only theoretically sound but also practically effective in diverse cloud-based industrial contexts. The combination of usability, automation, and domain versatility positions SmartSLA as a promising tool for smart SLA design and enforcement in modern service-oriented infrastructures.

To complement the applicability assessment of SmartSLA, we measured the deployment cost of the generated smart contracts on the Ethereum platform. Eight representative SLA scenarios were deployed using the Remix IDE with a JavaScript VM. Table III summarizes the reported gas usage, transaction cost, and net execution cost for each case study. The results show that deployment costs range from approximately 1.3M gas (Cloud Network) to 4.1M gas (Cloud Storage). This variation is aligned with the structural complexity of each SLA model, as contracts with more objectives and compliance requirements consume more gas during initialization. Overall, the results confirm that SmartSLA produces contracts with manageable overhead while preserving flexibility across diverse application domains.

TABLE III Deployment costs for eight case studies

Case Study	Gas	Transaction Cost	Execution Cost
Data Analytics	2,744,848	2,386,824	2,203,028
Cloud Database	3,880,848	3,374,650	3,122,104
Cloud Network	1,677,876	1,459,022	1,318,504
Cloud Storage	4,153,580	3,611,808	3,333,494
Cloud Infrastructure	2,294,582	1,995,288	1,829,256
Disaster Recovery	3,031,938	2,636,467	2,466,451
Internet of Things	2,165,891	1,883,383	1,725,925
Serverless Computing	2,830,524	2,461,325	2,280,849

B. Metamodel Structural Comparison

To address RQ3, we conducted a quantitative comparison of the SmartSLA metamodel against five existing SLA and smart contract languages. Notably, these languages fall into two distinct categories: (I) SLA modeling languages, such as AsyncSLA [20], CloudCost [22], and QaSAS [21], which are designed to express and reason about service-level requirements in cloud environments, and (II) smart contract

description languages, such as iContractML [10] and iContractML 2.0 [11], which focus on the specification of blockchain-based contractual logic but do not address SLA concerns directly.

Unlike these existing solutions, SmartSLA aims to bridge the gap between SLA modeling and smart contract generation, offering an integrated, model-driven solution that covers the entire lifecycle—from abstract SLA specification to executable blockchain code. To underscore this contribution, we compare the structural properties of SmartSLA's metamodel with those of the aforementioned languages.

The comparison uses both primary metrics (e.g., number of classes, attributes, relationships) and derived structural metrics, such as maximum depth of inheritance (DITmax), maximum fan-out (Fanoutmax), number of predecessor nodes (PRED), number of inherited features (INHF), and total number of features (NTF). Furthermore, three high-level quality criteria—**maintainability**, **understandability**, and **extensibility**—were calculated using the following standard formulas:

$$Maintainability = \frac{NC + NA + NR + DIT_{Max} + Fanout_{Max}}{5} \quad (1)$$

$$Understandability = \frac{\sum_{K=1}^{NC} PRED + 1}{NC} \quad (2)$$

$$Extensibility = \frac{INHF}{NFT} \quad (3)$$

Here, a lower value in maintainability suggests better modularity and ease of management, a higher value in understandability indicates clearer inheritance structure, and a higher extensibility score reflects better support for model evolution through inheritance reuse. Table III presents the updated metric values for all six metamodels, including the revised SmartSLA values after incorporating abstract parent classes and reducing attribute redundancy.

TABLE IV Comparison of SmartSLA metamodel with five other metamodels

Metric	[20]	[22]	[21]	[11]	[10]	SmartSLA
NC	12	18	15	16	15	**16**
NA	16	25	16	26	23	**40**
NR	10	9	17	10	7	**15**
DITmax	1	1	1	2	1	**1**
Fanoutmax	3	3	4	3	3	**7**
PRED	1	1	1	3	3	**1**
INHF	5	3	1	11	10	**8**
NTF	26	34	33	36	30	**57**
Maintainability	8.6	11.2	10.6	11.4	10.4	**15.8**
Understandability	1.3	1.389	1.13	1.5	1.6	**1.563**
extensibility	0.192	0.09	0.03	0.305	0.333	**0.14**

Structural Analysis

With 16 classes (NC), SmartSLA sits within a moderate complexity range, indicating an appropriate level of abstraction for capturing SLA-specific constructs. However, it significantly surpasses others in attribute richness, with 40 attributes (NA)—the highest among all six—underscoring its focus on detailed specification of service-level properties. The number of relationships (NR = 15) is also relatively high, reflecting a comprehensive linking of elements within the model.

The inheritance structure is intentionally flat, with a DITmax of 1 and only one predecessor per class (PRED = 1). This design choice promotes simplicity and makes the model easier to understand, especially for stakeholders unfamiliar with deep inheritance chains. In contrast, *iContractML 2.0* shows a DITmax of 2, and both *iContractML* versions have PRED = 3, indicating slightly more complex hierarchies.

SmartSLA's Fanoutmax of 7—the highest among the compared metamodels—results from the central SLAContract class referencing multiple associated components (e.g., objectives, penalties, security requirements). This design emphasizes centralized integration, making the contract structure semantically clear and modular.

Quality Criteria

As shown in Fig. 5, the Maintainability score of *SmartSLA* (15.8) is the highest in the comparison, due to its combination of high NA and $Fanout_{max}$. Although a higher number of attributes and references increases the metric's numerical value, it does not necessarily indicate poor structure. Instead, it reflects the design's emphasis on rich detail and centralized coordination within the SLAContract class. This moderate maintainability suggests a trade-off between structural compactness and expressive modeling capacity.

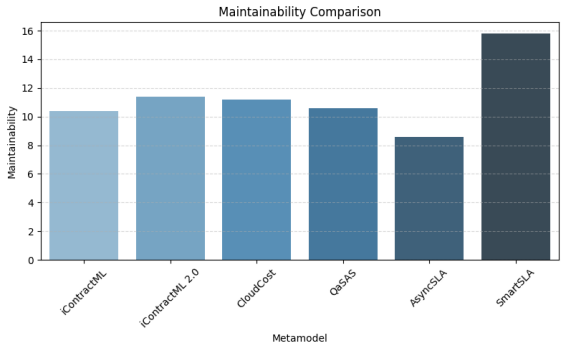

Fig. 5. Comparison of metamodels based on Maintainability.

Fig. 6 reveals that the Understandability score of *SmartSLA* (1.563) is among the top three, nearly matching *iContractML* [11] (1.6) and surpassing *CloudCost* [22] and *QaSAS* [21]. Since the inheritance hierarchy is shallow, with minimal predecessor depth, the class structure remains straightforward. This enhances clarity for new users,

especially those unfamiliar with complex metamodel hierarchies, and facilitates ease of learning and onboarding.

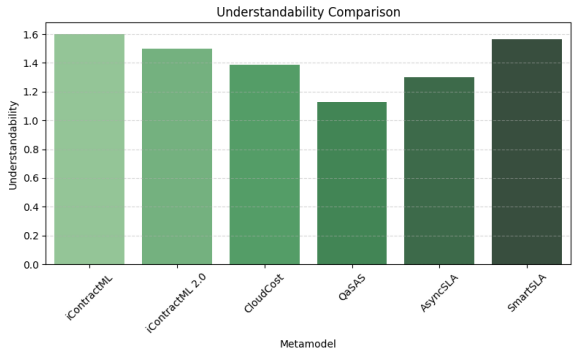

Fig. 6. Comparison of metamodels based on Understandability.

Fig. 7 illustrates that On **Extensibility**, *SmartSLA* achieves a moderate score (0.14), better than *CloudCost* [22] (0.09) and *QaSAS* [21] (0.03), though lower than *iContractML* [11] (0.333). This reflects the limited reliance on inheritance for feature reuse, which is a known trade-off of the intentionally shallow class hierarchy. The extensibility could be improved in future versions by incorporating more systematic inheritance strategies for reusable SLA components.

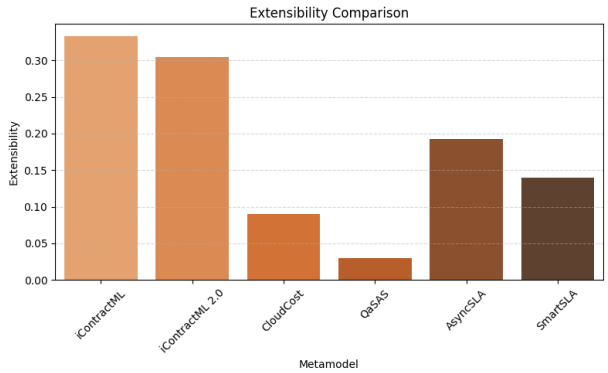

Fig. 7. Comparison of six metamodels based on Extensibility.

Summary and Implications

The SmartSLA metamodel demonstrates a balanced quality across key software criteria, showcasing its unique strengths and thoughtful design trade-offs. Its maintainability is moderate, because of a high number of attributes and a centralized structure with a moderate maximum fan-out. This reflects a focus on detailed and expressive SLA specifications.

For understandability, SmartSLA ranks among the top compared to other metamodels, thanks to its streamlined, flat structure. By avoiding deep inheritance layers, it keeps relationships between core SLA elements clear and straightforward, making it easier for non-professional designers to work with the framework.

Its extensibility is moderate, suggesting room for improvement in leveraging inheritance to make the metamodel more adaptable. Future enhancements could focus on refining this aspect to support more flexible reuse of components without sacrificing simplicity.

What sets SmartSLA apart from other frameworks are three standout features:

- **Blending SLAs with Smart Contracts:** Unlike other languages that focus solely on either SLA modeling or smart contract logic, SmartSLA seamlessly integrates both. This unique combination enables automated, blockchain-based enforcement of service agreements, streamlining the entire SLA lifecycle.
- **Simplified Design for Clarity:** The flat structure minimizes complexity, making the connections between key elements—like service goals, penalties, and parties—more intuitive. This clarity reduces the learning curve and supports broader adoption across technical and non-technical users.
- **Detailed and Precise Modeling:** With a rich set of attributes, SmartSLA captures fine-grained details of SLAs, from performance metrics to compliance rules. This thoroughness ensures that the metamodel can handle intricate service agreements with precision.

Overall, the evaluation confirms that SmartSLA meets its design objectives by combining automation, structural soundness, and cross-domain applicability, directly addressing the research questions posed in Section I.

VI. Conclusion

A complete solution for the automated administration of Service Level Agreements (SLAs) in cloud-based industrial environments is offered by the SmartSLA framework. SmartSLA overcomes the main drawbacks with traditional SLA management, including manual implementation, a lack of transparency, and specification complexity, by merging model-driven engineering with blockchain technology. The key components SmartSLA, i.e., a comprehensive metamodel, an intuitive graphical editor, and a robust code generator, enable non-professional developers to simply specify, validate, and deploy executable smart contracts on platforms such as Ethereum.

To demonstrate the applicability of the SmartSLA framework eight mostly-common case studies in various domains such as data analytics, cloud storage, IoT, and disaster recovery were studied. By automatically converting high-level models into Solidity smart contracts, SmartSLA enhanced accuracy, decreased manual labor, and offered domain-specific flexibility in all scenarios. The evaluation of structural comparisons with three SLA modeling languages and two smart contract metamodels show that SmartSLA has a balanced design, with moderate maintainability, high understandability, and sufficient extensibility.

Significantly, SmartSLA makes SLA automation accessible to both technical and non-technical stakeholders by bridging the gap between low-level blockchain enforcement and high-level SLA modeling. It encourages operational scalability, traceability, and reliability in cloud

infrastructures by supporting the entire SLA lifecycle, from visual design to on-chain execution.

The evaluation results provide empirical answers to the research questions: SmartSLA successfully automates contract generation across domains (RQ1), reduces design effort and improves reliability (RQ2), and achieves a balanced metamodel structure with high understandability and moderate extensibility compared to existing approaches (RQ3).

To further advance the SmartSLA framework, several directions can be pursued. Improving the adaptability of the metamodel by introducing modular components, reusable patterns, and enhanced inheritance strategies would strengthen extensibility and allow customization for specific industry requirements while maintaining overall system stability.

Another promising line of development involves enriching operational intelligence. Incorporating real-time monitoring and advanced analytics would enable SmartSLA to promptly react to service changes, while AI-driven prediction techniques could support proactive detection and resolution of potential SLA violations before they materialize.

Expanding the platform to support a broader range of blockchain technologies—including Hyperledger, Polkadot, and lightweight Ethereum Layer-2 solutions—would significantly enhance deployment flexibility and address the diverse preferences of industrial adopters. At the same time, exploring compliance- and regulation-oriented SLA specifications could open opportunities in highly regulated sectors such as finance, healthcare, and government. In parallel, integrating SmartSLA with DevOps pipelines and delivering a user-friendly web-based graphical editor would improve accessibility and streamline practical adoption.

Importantly, the current evaluation has already quantified the deployment cost of SmartSLA contracts across eight representative case studies, demonstrating that the overhead remains manageable and proportional to the structural complexity of the SLA models. Building on this, future work will extend the evaluation to runtime operations. In particular, measuring the gas consumption of interactive functions such as *applyPenalty*, *addChangeRequest*, and *logHistory* under realistic workloads will provide a more comprehensive picture of the operational overhead of SmartSLA.

With these enhancements, SmartSLA is well positioned to evolve into a robust and versatile framework for transparent, automated, and scalable SLA management—an ideal fit for the rapidly changing cloud environments of today.

REFRENCES

[1] P. Patel, A. H. Ranabahu, and A. P. Sheth, “Service level agreement in cloud computing,” 2009.

[2] H. Zhou, X. Ouyang, Z. Ren, J. Su, C. de Laat, and Z. Zhao, “A blockchain based witness model for trustworthy cloud service level agreement enforcement,” in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 1567–1575. doi:10.1109/INFOCOM.2019.8737580

[3] J. Skene, D. D. Lamanna, and W. Emmerich, “Precise service level agreements,” in *Proceedings of the 26th International Conference on Software Engineering*, 2004, pp. 179–188. doi: 10.5555/998675.999422

[4] A. R. Da Silva, “Model-driven engineering: A survey supported by the unified conceptual model,” *Computer Languages, Systems & Structures*, vol. 43, pp. 139–155, 2015. doi: 10.1016/j.cl.2015.06.001

[5] J. P. de Brito Gonçalves, R. L. Gomes, R. da Silva Villaca, E. Municio, and J. Marquez-Barja, “A service level agreement verification system using blockchains,” in *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*, 2020, pp. 541–544. doi: 10.1109/ICSESS49938.2020.9237735

[6] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, “Blockchain,” *Business & Information Systems Engineering*, vol. 59, no. 3, pp. 183–187, 2017. doi: 10.1007/s12599-017-0467-3

[7] Y. Ait Hsain, N. Laaz, and S. Mbarki, “Ethereum’s smart contracts construction and development using model driven engineering technologies: a review,” *Procedia Computer Science*, vol. 184, pp. 785–790, 2021. doi: 10.1016/j.procs.2021.03.097

[8] W. Tan, H. Zhu, J. Tan, Y. Zhao, L. Da Xu, and K. Guo, “A novel service level agreement model using blockchain and smart contract for cloud manufacturing in industry 4.0,” *Enterprise Information Systems*, vol. 16, no. 12, p. 1939426, 2022. doi: 10.1080/17517575.2021.1939426

[9] R. B. Uriarte, H. Zhou, K. Kritikos, Z. Shi, Z. Zhao, and R. De Nicola, “Distributed service-level agreement management with smart contracts and blockchain,” *Concurrency and Computation: Practice and Experience*, vol. 33, no. 14, p. e5800, 2021. doi: 10.1002/cpe.5800

[10] M. Hamdaqa, L. A. P. Metz, and I. Qasse, "icontractml: A domain-specific language for modeling and deploying smart contracts onto multiple blockchain platforms," in Proceedings of the 12th System Analysis and Modelling Conference, 2020, pp. 34-43. doi: 10.1145/3419804.3421454

[11] M. Hamdaqa, L. A. P. Met, and I. Qasse, “iContractML 2.0: A domain-specific language for modeling and deploying smart contracts onto multiple blockchain platforms,” *Information and Software Technology*, vol. 144, p. 106762, 2022. doi: 10.1016/j.infsof.2021.106762

[12] O. Kamel, A. Chaoui, G. Diaz, and M. Gharzouli, “SLA-driven modeling and verifying cloud systems: A bigraphical reactive systems-based approach,” *Computer Standards & Interfaces*, vol. 74, p. 103483, 2021. doi: 10.1016/j.csi.2020.103483

[13] H. Nabli, R. Ben Djemaa, and I. Amous Ben Amor, “Cloud services description ontology used for service selection,” *Service Oriented Computing and Applications*, vol. 16, no. 1, pp. 17–30, 2022. doi: 10.1007/s11761-021-00328-y

[14] 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. doi: 10.1504/IJWGS.2018.095647

[15] K. Upadhyay, R. Dantu, Y. He, S. Badruddoja, and A. Salau, "Can't understand SLAs? Use the smart contract," in *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, 2021, pp. 129–136. doi: 10.1109/TPSISA52974.2021.00015

[16] A. Breckel, J. Pietron, K. Juhnke, and M. Tichy, "A domain-specific language and interactive user interface for model-driven engineering of technology roadmaps," in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2020, pp. 162–170. doi: 10.1109/SEAA51224.2020.00035

[17] I. Jiménez-Pastor, A. Garmendia, and J. de Lara, "Scalable model exploration for model-driven engineering," *Journal of Systems and Software*, vol. 132, pp. 204–225, 2017. doi: 10.1016/j.jss.2017.07.011

[18] T.-V. Nguyen, L.-S. Lê, S. A. Shah, S. Hameed, and D. Draheim, "PenChain: A blockchain-based platform for penalty-aware service provisioning," *IEEE Access*, vol. 12, pp. 1005–1030, 2023. doi: 10.1109/ACCESS.2023.3344038

[19] A. Alzubaidi, K. Mitra, and E. Solaiman, "SLA representation and awareness within blockchain in the context of IoT," *IET*, 2024. doi: 10.1049/PBPC027E_ch6

[20] M. Oriol, A. Gómez, and J. Cabot, "AsyncSLA: Towards a service level agreement for asynchronous services," in *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, 2024, pp. 1781–1788. doi: 10.1145/3605098.3636074

[21] E. Maatougui, C. Bouanaka, and N. Zeghib, "Towards a meta-model for quality-aware self-adaptive systems design," in *Proceedings of the 3rd International Workshop on Interplay of Model-Driven and Component-Based Software Engineering co-located with ACM/IEEE 19th International Conference on Model*, 2016. url: https://ceur-ws.org/Vol-1723/2.pdf

[22] M.-E. Cambronero, A. Bernal, V. Valero, P. C. Cañizares, and A. Núñez, "Profiling SLAs for cloud system infrastructures and user interactions," *PeerJ Computer Science*, vol. 7, p. e513, 2021. doi: 10.7717/peerj-cs.513

[23] R. Sghaier, C. El Hog, R. Ben Djemaa, and L. Sliman, "A review on SLA monitoring based on blockchain," in *International Conference on Intelligent Systems Design and Applications*, 2023, pp. 458–467. doi: 10.1007/978-3-031-64650-8_46

[24] N. Hamdi, C. El Hog, R. Ben Djemaa, and L. Sliman, "A survey on SLA management using blockchain based smart contracts," in *Intelligent Systems Design and Applications (ISDA 2021)*, 2021. doi: 0.1007/978-3-030-96308-8_132

[25] A. Mahapatra, K. Mishra, S. K. Majhi, and R. Pradhan, "Blockchain in evolving computing paradigms: A beginner's guide for review and future directions," in *2023 IEEE 11th Region 10 Humanitarian Technology Conference (R10-HTC)*, 2023, pp. 595–602. doi: 10.1109/R10-HTC57504.2023.10461800

[26] Z. F. Azzahra and I. G. B. B. Nugraha, "Service-level agreement management with blockchain-based smart contract to improve the quality of IT service management," in *Proceedings of the 2023 12th International Conference on Software and Computer Applications*, 2023, pp. 260–266. doi: 10.1145/3587828.3587867

[27] W. Tang, J. Zhang, and R. Guo, "A blockchain-based for trustworthy tourism service level agreement," in *2023 3rd International Conference on Computer Science and Blockchain (CCSB)*, 2023, pp. 195–199. doi: 10.1109/CCSB60789.2023.10398825

[28] W. B. S. Souei, C. El Hog, R. Ben Djemaa, L. Sliman, and I. A. Ben Amor, "Towards smart contract distributed directory based on the uniform description language," *Journal of Computer Languages*, vol. 77, p. 101225, 2023. doi: 10.1016/j.cola.2023.101225

[29] S. K. Battula, S. Garg, R. Naha, M. B. Amin, B. Kang, and E. Aghasian, "A blockchain-based framework for automatic SLA management in fog computing environments," *The Journal of Supercomputing*, vol. 78, no. 15, pp. 16647–16677, 2022. doi: 10.1007/s11227-022-04545-w

[30] P. Cedillo, E. Insfran, and S. M. Abrahao Gonzales, "Monitoring cloud services through models at runtime: A case in an ambient assisted living environment," *Journal of Object Technology*, vol. 21, no. 4, pp. 1–19, 2022. doi: 10.5381/jot.2022.21.4.a1

[31] A. Makwe, D. Sukheja, K. Ohri, and P. Kanungo, "SLA aware CSP selection and resource monitoring framework for infrastructure as a service cloud," *Service Oriented Computing and Applications (SOCA)*, 2025. doi: 10.1007/s11761-025-00471-w.

[32] A. Booth, A. Alqahtani, and E. Solaiman, "IoT Monitoring with Blockchain: Generating Smart Contracts from Service Level Agreements," *Managing Internet of Things Applications across Edge and Cloud Data Centres, IET*, 2024. doi: 10.48550/arXiv.2408.15016.

[33] S. T. Muntaha, Q. Z. Ahmed, F. A. Khan, Z. D. Zaharis, and P. I. Lazaridis, "Hybrid Blockchain-Based Multi-Operator Resource Sharing and SLA Management," *IEEE Open Journal of the Communications Society*, vol. 6, pp. 362–377, 2025. doi: 10.1109/OJCOMS.2024.3523362.

[34] W. V. Solis, P. Cedillo, and A. Kertesz, "MICAAL: A Domain-Specific Language for Microservices in Ambient Assisted Living," *IEEE Access*, vol. 13, pp. 56255–56272, 2025. doi: 10.1109/ACCESS.2025.3555831.