Model-Driven Transformation of Business Processes into Blockchain Smart Contracts

Imane Bouzaidi Tiali*[1], Zineb Aarab[2], Achraf Lyazidi[3], Moulay Driss Rahmani[4]
LRIT-Associated Unit to CNRST (URAC 29)-Faculty of Sciences, Mohammed V University in Rabat, Morocco[1,2,3,4]
SIWEB-Mohammadia School of Engineers, Mohammed V University in Rabat, Morocco[2]
GENIUS Laboratory SUPMTI of Rabat, Rabat, Morocco[2]

***Abstract*—This paper presents a comprehensive Model-Driven Engineering (MDE) methodology for automatically transforming Business Process Model and Notation (BPMN) diagrams into executable blockchain-based smart contracts. The proposed approach defines a set of Atlas Transformation Language (ATL) rules that systematically map BPMN elements to Solidity constructs, ensuring semantic consistency and traceability throughout the transformation process. The framework integrates several stages, including process modeling, model validation, code generation, and deployment, supported by tools such as Camunda, Eclipse ATL, Remix IDE, and MetaMask. Experimental validation on the Ethereum Sepolia test network demonstrates the approach's ability to enhance automation, reduce manual coding errors, and improve synchronization between business workflows and their on-chain implementations. Compared to existing BPMN-to-blockchain frameworks, the proposed solution offers a unified and reusable transformation pipeline that bridges the gap between business process modeling and blockchain execution. The study concludes that MDE provides a scalable, traceable, and standardized foundation for developing decentralized business process applications.**

***Keywords*—Model-driven engineering; BPMN; smart contracts; blockchain; ATL; automation; solidity; process transformation**

I. Introduction

Blockchain technology has introduced new paradigms for managing and coordinating cross-organizational processes by enabling the decentralized execution of agreements through smart contracts. These self-executing programs enhance transparency, trust, and auditability among participants without intermediaries [1], [2]. However, developing smart contracts manually remains a complex and error-prone activity that requires deep technical expertise in blockchain languages such as Solidity, as well as substantial effort in validation and testing [3]. Consequently, organizations face persistent challenges in ensuring reliability, scalability, and correctness when automating business workflows on blockchain infrastructures.

To address these limitations, Model-Driven Engineering (MDE) provides a methodology that emphasizes abstraction and automation in software development. Through MDE, developers can design system behavior using high-level models that can later be transformed into executable code, thus reducing manual intervention and increasing traceability [4], [5]. In the context of business process automation, Business Process Model and Notation (BPMN) serves as a widely adopted standard for visually representing workflows in an intuitive and platform-independent manner. When combined with transformation languages such as the Atlas Transformation Language (ATL), BPMN models can be automatically converted into smart contracts deployed on blockchain networks [3], [6].

Several studies have explored MDE-based transformations for blockchain applications, including Lorikeet [7], TABS [8], and MDE4BBIS [9]. Although these frameworks have advanced the integration of business modeling and blockchain deployment, most existing approaches remain fragmented or tool-specific. They typically focus on partial automation or theoretical descriptions of transformation processes without providing a fully operational, traceable, and validated pipeline. Moreover, prior works often omit detailed mappings between BPMN elements and their corresponding smart contract constructs, limiting their practical applicability and reproducibility in enterprise scenarios [10], [11].

Research Problem: Despite these advancements, there is currently no standardized and traceable end-to-end methodology for transforming BPMN business process models into deployable blockchain smart contracts while preserving the semantics of the original workflow.

Research Questions:

- How can MDE principles be leveraged to automate the transformation of BPMN models into smart contracts with minimal manual intervention?
- What transformation rules and architectural design are required to ensure semantic consistency, traceability, and deployment feasibility?

Objectives: The objectives of this study are to:

- Define a comprehensive MDE-based framework for transforming BPMN models into smart contracts using ATL.
- Establish explicit transformation rules that systematically map BPMN elements to Solidity constructs.
- Validate the proposed methodology through deployment and execution on the Ethereum Sepolia test network.

Significance and Contributions: The proposed approach unifies the entire transformation lifecycle—from BPMN modeling and validation to Solidity generation and blockchain deployment—within a modular and reusable architecture. Unlike existing frameworks such as Lorikeet, TABS, or MDE4BBIS, this study contributes a fully integrated, empirically validated methodology that enhances automation, improves semantic

traceability, and promotes standardization in model-driven blockchain development [6], [8], [10]. Novelty and Distinction: In contrast to previous frameworks such as Lorikeet [7], TABS [8], and MDE4BBIS [9], which partially automate the BPMN-to-Solidity transformation or focus on limited metamodel mapping, the proposed methodology introduces a unified and traceable pipeline that covers the entire lifecycle—from BPMN model validation to smart contract deployment. The novelty lies in the explicit mapping of BPMN temporal and resource constraints to Solidity execution semantics, ensuring end-to-end consistency, automation, and deployability across decentralized environments.

II. Background Definition

A. Model

Within the context of Model-Driven Engineering (MDE), a **model** refers to an abstracted and focused depiction of a system, designed to emphasize particular characteristics pertinent to a specific problem domain. It serves various purposes such as aiding comprehension, performing analysis, supporting communication, or facilitating the structured creation of system components in an automated manner [3], [10].

Every model adheres to a corresponding metamodel, which defines its structure and semantics. This relationship enables the application of formal transformation techniques and supports tool integration throughout the system development process [7].

B. Meta-model: Metamodelisation Layers

A **metamodel** can be described as a higher-level model that outlines the structural elements, semantic constraints, and rules governing a particular modeling language. In essence, it serves as a model about models, offering a formal foundation for constructing and interpreting models within a specific domain [6].

1) Key Concepts of Metamodels:

- Abstraction: Compared to standard models, metamodels function at a more abstract level. Whereas a model might illustrate a concrete system or process, the metamodel formalizes the building blocks and their interrelations used to design such models [10].
- Structure and Syntax: Metamodels define the permitted components—such as entities, attributes, and connections—within a modeling framework. For instance, in Unified Modeling Language (UML), the metamodel prescribes what elements make up a valid class diagram [5].
- Validation: They impose structural rules and integrity constraints to ensure that models conform to the expected language definition, helping maintain consistency and validity [3].
- Examples: Widely known metamodels include:
  - UML Metamodel: Provides the formal syntax and semantics for UML-based diagrams [2].
  - BPMN Metamodel: Specifies modeling elements and behavior for business process workflows using BPMN [12].
- Use in Model-Driven Engineering (MDE): Within MDE practices, metamodels are fundamental because they enable the transformation of high-level models into executable artifacts. This facilitates automation in software development and ensures consistency across generated implementations [8].

2) Metamodelization Layers: Metamodelization is typically organized into several abstraction levels, progressing from concrete real-world systems to highly abstract modeling structures. These layers are commonly described as follows:

- M0: Real-World Systems – This foundational level includes actual systems or operational processes observed in real environments. Examples include ongoing business workflows or deployed information systems [13].
- M1: Models – This level encompasses models derived from metamodels. For example, a BPMN model might depict the logic of a particular business operation. These models serve as tangible applications of the concepts defined at the metamodel level [9].
- M2: Metamodels – Here, we encounter frameworks such as BPMN and UML, which define the formal structure and semantics of modeling languages. These metamodels provide the abstract components required to represent systems or business processes conceptually [6].
- M3: Meta-Metamodel – This top-level layer defines the constructs used to build metamodels themselves. A notable example is the Meta-Object Facility (MOF), which offers a standardized mechanism for specifying metamodels in a consistent and formalized way [11].

The hierarchical organization of these modeling levels is depicted in Fig. 1, which illustrates the M0–M3 layering from real-world systems to meta-metamodels.

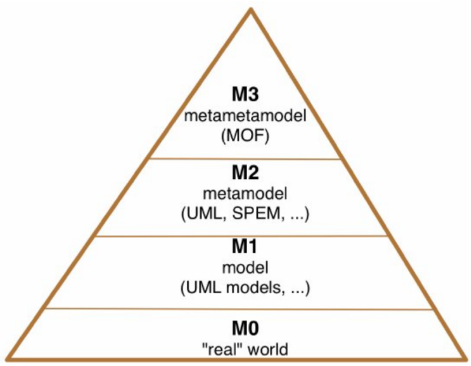

Fig. 1. Metamodeling hierarchy [4].

C. Model Transformation

In the context of Model-Driven Engineering (MDE), model transformation refers to the automated process of translating a source model—structured according to a specific metamodel—into a corresponding target model that conforms to

a different metamodel. This mechanism enhances the consistency and repeatability of software development, and is particularly advantageous in blockchain-oriented environments where precision and reliability are essential [4].

Over the years, numerous transformation languages and methods have emerged to support these processes:

- ATL (Atlas Transformation Language): A widely adopted tool for specifying and executing model transformations [3], [5].
- QVT (Query/View/Transformation): A standardized framework by OMG for expressing model-to-model transformations declaratively [14].
- Graph Transformation: Utilizes graph-based techniques to define transformation rules and apply them to model structures [15].
- Epsilon: A comprehensive language suite designed for model management, supporting a range of transformation and validation operations [11].
- Xtend: A statically typed language that facilitates the implementation of model-to-model transformation logic [12].
- Acceleo: An implementation of the MOFM2T (Model to Text) transformation specification developed under the OMG standard [8].

These transformation mechanisms constitute the foundation of the proposed approach presented in Section IV (Methodology), where BPMN models are systematically translated into Solidity smart contracts using ATL rules.

D. BPMN Overview

Business Process Model and Notation (BPMN) has emerged as a widely accepted notation standard for modeling business workflows. Designed and maintained by the Object Management Group (OMG), its primary goal is to ensure clarity and usability across various stakeholder roles, including business analysts, developers, and process managers [2], [16].

The popularity of BPMN is reflected in its integration into enterprise software platforms such as Oracle Business Process Analysis Suite and Camunda, which enable the automatic generation of executable systems based on BPMN diagrams [6]. In contrast to traditional programming-based approaches like BPEL or Java, the present work leverages BPMN as a foundation for generating blockchain-based smart contracts [8], [9].

1) Flow Objects: Flow objects include:

- Events: Start, Intermediate, and End. Intermediate events include timers, messages, conditions, etc. [2], [17].
- Activities: Tasks, Sub-processes, and Call Activities. Sub-processes may include compensation or transactions [6], [16].
- Gateways: Used to control sequence flow. Types include Exclusive, Inclusive, and Parallel, both converging and diverging [2], [5].

2) Examples:

- An exclusive gateway that splits the process flow into multiple branches based on defined conditions [2], [8].
- A parallel gateway that waits for all incoming sequences to arrive before continuing the execution [12], [18].
- A multi-instance task executed simultaneously across a group of input elements [10], [11].

E. Blockchain Ecosystem

The blockchain ecosystem includes a set of technologies, platforms, and development tools that support the design, implementation, and execution of decentralized applications. This ecosystem ranges from blockchain networks like Ethereum and Hyperledger to tools and methodologies such as model-driven engineering (MDE) [5], [6], [10].

Model-driven approaches enable developers to create formal models (e.g., BPMN) that are then transformed into executable blockchain code, enhancing productivity and consistency [7], [11], [16]. These practices are especially valuable in collaborative environments involving multiple stakeholders, where process standardization and automation are critical. This conceptual and technological ecosystem provides the environment in which our proposed transformation framework operates, linking BPMN-based process modeling with blockchain-based execution.

F. Smart Contracts

Smart contracts refer to autonomous programs deployed on blockchain networks, designed to execute predefined rules and agreements automatically when certain conditions are met. By removing the need for intermediaries, they enhance transaction reliability by promoting integrity, transparency, and resistance to tampering.

These contracts are especially valuable in managing complex process logic within decentralized ecosystems. Their applicability spans sectors such as finance, supply chain management, and multi-party collaborations. Moreover, when combined with modeling standards like BPMN, smart contracts enable traceable and verifiable execution of business workflows on blockchain infrastructures [2], [12], [18].

1) Programming Languages for Smart Contracts: Blockchain ecosystems support a variety of languages tailored for smart contract development, each offering different capabilities and trade-offs:

- Solidity: The dominant language for Ethereum smart contracts, supporting complex logic and security features [3].
- Vyper: A Python-inspired language for Ethereum that emphasizes readability, simplicity, and formal verification. Its safety features make it suitable for high-assurance smart contracts [19].
- Rust: Widely used in blockchains like Solana and NEAR for its performance and memory safety. It is favored for developing high-throughput decentralized

TABLE I. COMPARATIVE OVERVIEW OF REVIEWED ARTICLES ON BLOCKCHAIN AND BPMN INTEGRATION

Article	Focus Area	Brief Description
Hofmann et al. (2018) [22]	Background and foundations of blockchain	Introduces blockchain principles from business and academic perspectives
Konstantinidis et al. (2020) [25]	Systematic mapping of blockchain applications	Identifies sectors where blockchain is applied, but not focused on BPM
Lu (2019) [13]	Survey on blockchain features and research challenges	Emphasizes decentralization, transparency, and technical limitations
Mendling et al. (2018) [23]	Research directions for blockchain in BPM	Highlights trends in process execution, design, and integration
Seebacher and Schüritz (2019) [26]	Blockchain in service and software systems	Observational review with mentions of BPM use in services
Hawlitschek et al. (2018) [24]	Blockchain and trust in the sharing economy	Reviews trust-free systems but lacks BPM-specific context
Tripathi et al. (2023) [27]	General blockchain review	Discusses historical background and future challenges
Liu et al. (2024) [1]	Smart contracts for cross-organization processes	Supports secure collaborative BPM using smart contracts
Tran et al. (2018) [7]	Lorikeet tool for smart contract generation	MDE-based tool for converting BPMN to smart contracts
Curty et al. (2022) [5]	Low-code platforms for blockchain development	Uses model-driven and low-code tools for blockchain BPM
Levasseur et al. (2021) [10]	Survey of MDE techniques for blockchain apps	Categorizes engineering approaches for smart contract modeling
De Sousa and Burnay (2021) [9]	MDE4BBIS framework for blockchain IS	Proposes an MDE-based system design framework for blockchain
Markovska et al. (2019) [16]	BPMN modeling for blockchain ecosystems	PhD thesis on business process modeling with BPMN
Lu et al. (2021) [6]	Integration of MDE for BPM and asset management	Formalizes model transformations in blockchain apps
Nassar et al. (2023) [11]	MDE approaches to blockchain system modeling	Develops conceptual models for blockchain-based designs
Bodorik et al. (2023) [8]	TABS framework for BPMN-to-Solidity transformation	Fully automates the generation of Solidity from BPMN
Corradini et al. (2023) [12]	Execution of BPMN choreographies on blockchain	Manages multi-instance BPM execution via smart contracts
Milani et al. (2021) [2]	Comparison of BPMN vs CMMN	Evaluates which modeling approach is best suited for blockchain
Shen et al. (2024) [28]	Hybrid BPMN-DMN on permissioned blockchain	Integrates process and decision modeling for secure collaboration
Ladleif et al. (2019) [29]	Modeling blockchain-based choreographies	Describes how to enforce BPMN collaboration on blockchain
Liu (2024) [18]	Long-term transaction support in smart contracts	Tackles lifecycle and state issues in BPMN-based contracts

applications and is often integrated into MDE workflows [9].

- Go and JavaScript (Node.js): Commonly used in Hyperledger Fabric for developing chaincode in permissioned enterprise blockchain solutions [5].
- Michelson: The low-level, stack-based language used in the Tezos blockchain. It is designed for formal reasoning and verification of contract behavior [20].
- Move: Originally developed for Diem (now used in Aptos and Sui), Move is a resource-oriented language focused on safety, flexibility, and precise control of digital assets [21].

III. ANALYSIS WITH SCIENTIFIC STUDIES

The initial comparative overview (Table I) offers a synthesized perspective on both foundational and recent scholarly efforts in blockchain research, particularly in the context of inter-organizational business process management (BPM). The analysis spans a variety of research directions, from general blockchain studies [13], [22] to domain-specific contributions focusing on BPM integration [1], [23]. A notable portion of the reviewed literature explores how model-driven engineering (MDE) techniques are leveraged to facilitate the transformation of BPM diagrams into blockchain-compatible components and executable logic [5], [7], [8]. Additional studies focus on issues such as trust, interoperability, and formal system modeling [2], [11], [24].

This detailed literature mapping helps outline methodological patterns, highlight gaps in current research, and point toward future directions for advancing the convergence of BPM and blockchain technologies.

A. Analysis

The comparative Table II summarizes key contributions in the domain of Model-Driven Engineering (MDE) for transforming BPMN models into executable smart contracts, particularly on blockchain platforms such as Ethereum. Each selected study highlights distinct aspects of this transformation process, categorized as follows:

- Main Criteria: This refers to the core focus or distinguishing feature of each work, such as model integration, level of automation, or comparative scope. For example, [6] addresses BPMN/UML integration, while [8] focuses on a fully automated transformation pipeline.
- Modeling Objectives: Most studies aim to automate the generation of smart contracts from BPMN or to enhance the expressiveness and formalization of business processes. Works like [7] and [8] emphasize automation, while [2] and [16] focus on comparative analysis and real-world application.
- Tools and Techniques Used: This includes the key platforms and languages employed for modeling and transformation, such as Lorikeet, Solidity, Camunda, and ATL. For instance, [5] uses Camunda in a low-code approach, and [6] leverages integrated BPMN and UML environments.

This structured overview highlights the current state of research, helping to identify existing strengths, limitations, and future opportunities in the automation of BPMN-to-blockchain application development.

B. Conclusion

This section has reviewed the theoretical background and current state of applying blockchain technologies within model-driven engineering, with a specific focus on BPMN-to-smart contract transformation. The literature indicates growing interest in leveraging MDE to advance blockchain-based business process automation.

TABLE II. ILLUSTRATION OF THE TRANSFORMATION FROM BPMN MODELS TO SOLIDITY SMART CONTRACTS

Article	Main Criteria	Modeling Objectives	Tools and Techniques
Lu et al. (2021) [6]	BPMN/UML integration, automation	Automating business processes and asset management	Lorikeet, BPMN, UML
Levasseur et al. (2021) [10]	Review of MDE approaches	Comparing MDE modeling techniques	Comparative analysis, classification
Curty et al. (2022) [5]	MDE and low-code integration	Accelerating development using Camunda	Camunda, MDE, low-code tools
Hsain et al. (2021) [3]	Ethereum-focused review	Formalizing business processes into Solidity code	MDE, BPMN, Ethereum
Tran et al. (2018) [7]	Automated transformation	Generating contracts from BPMN models	Lorikeet, BPMN, Solidity
Nassar et al. (2023) [11]	Complete metamodeling	Generation of PIM/PSM models	Conceptual MDE approach
Corradini et al. (2023) [12]	Multi-instance execution	Choreography of business processes on blockchain	BPMN, Solidity
Bodorik et al. (2023) [8]	Full automation	Direct transformation from BPMN to Solidity	TABS Framework
Liu (2024) [18]	Transaction sustainability	Long-lived state management in contracts	MDE, lifecycle handling
Milani et al. (2021) [2]	BPMN vs. CMMN comparison	Identifying the most appropriate notation	BPMN, CMMN
Shen et al. (2024) [28]	Hybrid BPMN-DMN	Securing cross-organizational decisions	BPMN, DMN, permissioned blockchain
Ladleif et al. (2019) [29]	Actor coordination	Managing complex interactions	BPMN choreography
Markovska et al. (2019) [16]	Applied case study	Modeling for blockchain ecosystems	BPMN, practical implementation
Tripathi et al. (2023) [27]	General technical review	Mapping blockchain challenges	Literature synthesis
De Sousa and Burnay (2021) [9]	Dedicated MDE framework	Integrating MDE in blockchain information systems	MDE4BBIS, conceptual modeling

Although existing frameworks and tools have made significant progress, challenges related to standardization, scalability, and semantic interoperability persist. There remains a need for more flexible and interoperable solutions that effectively bridge business process modeling and decentralized execution. Building upon these insights, our subsequent work proposes a transformation solution grounded in MDE principles to improve traceability, automation, and correctness in blockchain application development. The next section presents the proposed methodology, which addresses these limitations by defining a unified and traceable ATL-based MDE framework for automatically transforming BPMN process models into deployable Solidity smart contracts.

IV. METHODOLOGY

A. Research Approach

This study adopts the Design Science Research (DSR) paradigm, a methodology commonly used in information systems engineering to build and evaluate innovative technical artifacts. The DSR framework supports the structured creation of technical artifacts aimed at solving real-world problems—in this case, enabling the transformation of BPMN-based business processes into blockchain-deployable smart contracts.

The DSR methodology is organized into three closely connected and recurring phases: the relevance phase, the rigor phase, and the design phase. The relevance phase ensures alignment with practical challenges in process automation and decentralized operations. The rigor phase draws on theoretical insights from model-driven engineering (MDE), blockchain architecture, and smart contract logic. Finally, the design phase covers the development, testing, and evaluation of the proposed transformation approach.

Through this methodical structure, we seek to close the gap between business modeling techniques and their blockchain-based implementations, ensuring that the developed solution is both robust from a technical standpoint and applicable in real scenarios.

B. Business Process Modeling with BPMN

To define business processes, this research utilizes *Business Process Model and Notation (BPMN)*. As a widely accepted and standardized graphical language, BPMN facilitates the representation of organizational procedures and interactions in a clear and structured manner [14], [16], [26]. Its broad adoption in industry makes it especially suitable for translating complex workflows into executable logic for blockchain environments.

C. Model Validation

Before any transformation, the BPMN models are validated to ensure correctness, consistency, and structural soundness. We detect common issues such as unconnected nodes, circular dependencies, and missing start or end events using tools like Camunda BPM [11], [27]. This guarantees the integrity of the models before transformation. Once the BPMN models are verified for structural correctness, they serve as valid input for the ATL-based model transformation process described in the following subsection.

D. Model Transformation with ATL

The transformation from BPMN to Solidity is performed using *ATL (Atlas Transformation Language)*. ATL rules define how BPMN elements such as tasks, gateways, and events are mapped to Solidity constructs like state variables and functions [2], [5], [8]. This model-to-model transformation is critical for preserving the logic and structure of the original process model.

E. Smart Contract Generation and Deployment

The ATL-generated model is translated into Solidity source code that includes:

- State variables to track process execution,
- Functions to represent activities and transitions,
- Security checks (e.g., `require` statements) to enforce logical constraints.

The generated contracts are tested and deployed on the Ethereum Sepolia testnet using Remix IDE and MetaMask. Transactions are then validated using Etherscan for traceability [3], [18].

Overall, the proposed methodology establishes a coherent and automated transformation pipeline that bridges the gap

between BPMN business process modeling and blockchain-based execution. The following section presents the experimental validation and results obtained from implementing this approach on real BPMN models deployed to the Ethereum Sepolia network.

F. Advantages of the Approach

Our MDE approach presents several benefits:

- Automation and Accuracy: Reduces manual coding errors and development time [6], [10].
- Security and Conformance: Ensures compliance with blockchain execution semantics [4].
- Scalability and Adaptability: Supports diverse blockchain scenarios including inter-organizational and multi-instance choreographies [12], [29].

This methodology aligns with the broader trends in model-driven blockchain application development, enabling reliable and efficient process automation [7], [9].

V. Proposed Metamodels and Model Transformation

To automate the generation of Solidity smart contracts from business process models, we define precise source and target metamodels, followed by a transformation specification in ATL.

A. BPMN Source Metamodel

The BPMN source metamodel specifies all elements necessary to capture a business workflow:

- Process: Represents the overall business process, with attributes such as `name`, `start`, `finish`, and associations to contained elements.
- Activity and Task: Units of work, each with a `name` and optional temporal constraints.
- Event: Points in the workflow, including *StartEvent*, *IntermediateEvent*, and *EndEvent*.
- Gateway: Control nodes for branching and merging flows (e.g., exclusive, parallel).
- Resource and Role: Actors or assets required for task execution.
- Pool and Lane: Organizational partitions to model responsibilities and participants.
- Temporal_constraint and Temporal_dependency: Elements to specify timing constraints and precedences between activities.

The overall structure of the BPMN source metamodel is illustrated in Fig. 2, showing the relationships between process, activity, event, gateway, and resource elements.

This metamodel builds upon the standard BPMN 2.0 specification and extends it to include temporal and resource-related concepts relevant to process automation [2], [6], [16].

These elements collectively provide the structural foundation for representing business logic, which will subsequently be mapped to the constructs of the Solidity target metamodel.

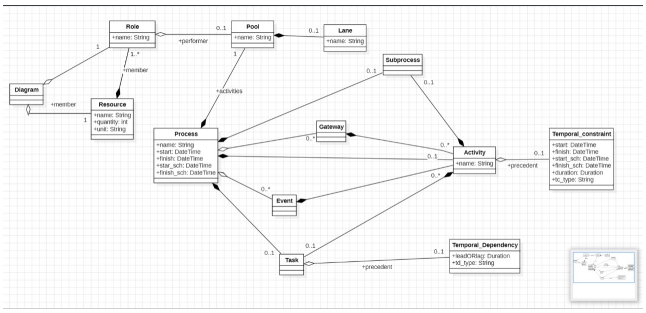

Fig. 2. BPMN source metamodel.

B. Solidity Target Metamodel

The Solidity target metamodel defines the structure of a smart contract:

- Contract: The top-level entity deployed on-chain, identified by a `name`.
- Function: Implements BPMN tasks and events, with attributes for `visibility` (e.g., `public`, `internal`) and `mutability` (e.g., `view`, `nonpayable`).
- StateVariable: Persistent variables storing process state (e.g., boolean flags tracking execution).
- Modifier: Preconditions or security checks (e.g., `onlyOwner`).
- Event: On-chain notifications for tracing execution.
- Struct and Enum: Complex data types to group related fields.
- Library: Reusable utility functions.
- Inheritance: Mechanism to extend or specialize contracts.

Fig. 3 presents the target Solidity metamodel, highlighting how contracts, functions, and state variables form the structural backbone of the generated code.

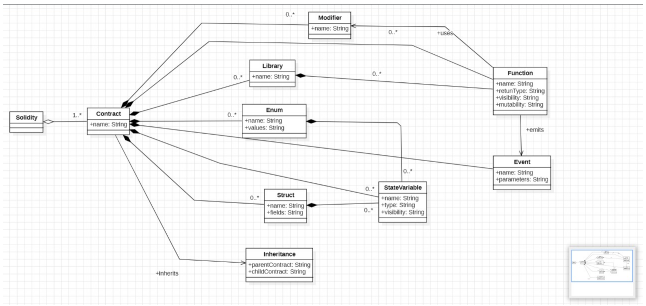

Fig. 3. Solidity target metamodel.

The design of the target metamodel aligns with the Solidity object-oriented structure defined by the Ethereum documentation and prior MDE-based blockchain frameworks such as Lorikeet and TABS [3], [7], [8].

C. Model Transformation with ATL

Converting BPMN-based process models into deployable Solidity smart contracts represents a central component of our model-driven engineering methodology. This transformation is achieved through the use of the Atlas Transformation Language (ATL), a declarative, rule-driven language tailored for model-to-model transformations, and implemented within the Eclipse Modeling Framework (EMF).

1) Transformation Objectives: The primary objective is to enable an automated and smooth transition from abstract business process representations to executable blockchain components. This approach minimizes manual intervention, maintains the original process logic, and enhances traceability across all stages of the development lifecycle.

2) Transformation Rules: The ATL module, named `Bpmn2Solidity`, defines mappings between BPMN elements and Solidity constructs. Each transformation rule systematically converts a BPMN component into its Solidity equivalent. The main rules include:

- StartEvent → Public function in Solidity (e.g., `startProcess()`)
- Task → Internal Solidity function (e.g., `executeTaskX()`)
- Gateway → Conditional logic using `if/else` statements
- EndEvent → Final state-marking function

3) Example Transformation Rule: Below is a simplified ATL rule that demonstrates the mapping of a BPMN task to a Solidity function:

```
rule Task2Function {
    from t : BPMN!Task
    to f : Solidity!Function (
        name <- 'execute' +
                t.name.firstToUpper(),
        visibility <- 'internal',
        body <- '...'
    )
}
```

4) Helper Functions: To streamline the code generation process, several helper methods were introduced:

- `capitalizeName()` – Capitalizes names to match Solidity naming conventions
- `generateRequire()` – Creates Solidity `require` statements for validation
- `generateAssignment()` – Produces state variable assignments
- `generateOutgoingCalls()` – Automates function chaining

5) Benefits: The use of ATL significantly improves:

- Automation: Minimizes human intervention and manual coding
- Consistency: Ensures that all generated contracts follow a coherent structure
- Traceability: Facilitates mapping between process model elements and generated code
- Reusability: Allows adaptation to different blockchain scenarios

The resulting transformation framework provides a complete and reproducible pipeline from process modeling to smart contract generation. The following section presents the implementation results and discusses the evaluation of the proposed approach.

VI. Application Architecture

This application aims to automate the conversion of BPMN-based workflows into executable smart contracts written in Solidity, leveraging a modular and systematic development approach. Its architecture integrates tools from model-driven engineering with blockchain-specific deployment frameworks to achieve seamless translation and deployment. Building upon the model-driven engineering methodology described in the previous section, this architecture operationalizes the transformation and deployment pipeline through a modular software framework.

A. Architecture Description

The architecture consists of six main modules that interact sequentially:

- User Interface: Allows users to upload a BPMN file and trigger the transformation process.
- BPMN Reader Module: Parses the BPMN model and extracts its elements using the Eclipse Modeling Framework (EMF).
- Validation Module: Detects structural issues such as disconnected nodes or cyclic flows.
- BPMN to Solidity Transformation Module: Uses ATL (Atlas Transformation Language) to convert BPMN elements into corresponding Solidity code fragments, following pre-defined transformation rules [30].
- Solidity Code Generator: Assembles the generated fragments into a complete smart contract, ensuring syntactic and logical correctness.
- Test and Deployment Module: Interacts with Remix IDE and MetaMask to compile, simulate, and deploy the contract on a test blockchain (e.g., Sepolia).

Each module has been implemented independently to ensure modularity and maintainability of the application. Communication between modules is managed through a controller that handles input/output and error propagation.

This modular organization aligns with standard model-driven software architectures that emphasize separation of concerns and tool interoperability [6], [11].

B. Architecture Diagram

The complete software architecture implementing this transformation workflow is presented in Fig. 4.

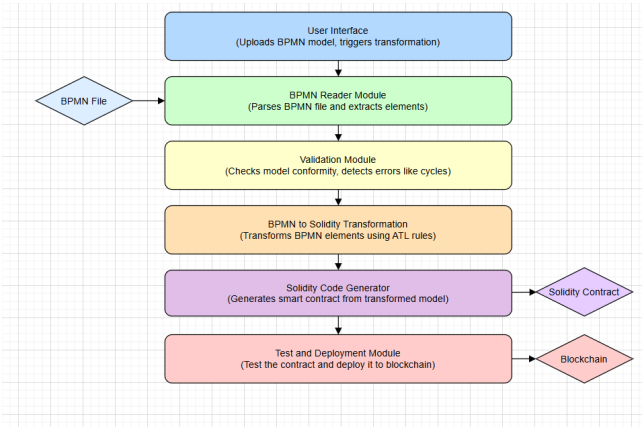

Fig. 4. Overall system architecture illustrating the BPMN-to-Solidity transformation workflow and deployment process.

C. Technology Stack

The system leverages various technologies, summarized in Table III.

TABLE III. TECHNOLOGY STACK USED IN THE IMPLEMENTATION

Component	Technology / Role
Model Parsing	EMF + Camunda BPMN; Reading and navigating BPMN models
Model Transformation	ATL (Eclipse); Applying transformation rules [30]
Backend	Java + Spring Boot; Application logic and flow coordination
Smart Contract	Solidity; Target blockchain programming language [31]
Deployment	Remix IDE + MetaMask; Smart contract testing and deployment
User Interface	HTML/CSS + Bootstrap; Interaction with the transformation tool

The integration of these technologies ensures a seamless workflow from model parsing and validation to code generation and blockchain deployment, maintaining both flexibility and scalability across development stages.

D. Advantages of the Architecture

This application architecture provides:

- Automation: End-to-end automation from BPMN input to blockchain deployment.
- Modularity: Each module can evolve independently without affecting the overall flow.
- Maintainability: Clearly defined responsibilities for each component.
- Traceability: Direct traceability between BPMN elements and the generated smart contract.

Overall, this architecture provides the structural and technological foundation necessary to support the proposed transformation framework. The following section evaluates the implementation results and demonstrates the effectiveness of the approach through practical case studies.

VII. VALIDATION AND DEPLOYMENT WITH REMIX

After generating the Solidity smart contracts from BPMN models, it is crucial to validate their correctness and expected behavior before deploying them to a live blockchain network. This process includes code verification, simulation of execution, and deployment on a test network.This validation stage represents the final phase of the proposed MDE pipeline, ensuring that the automatically generated Solidity contracts function correctly before deployment. This section describes the validation process using Remix IDE, the role of MetaMask in signing transactions, and provides the main application interfaces.

A. Verification Using Remix IDE

Remix IDE is an online development environment specifically designed for Solidity smart contracts. It supports compilation, testing, and analysis of smart contracts prior to deployment. Remix IDE remains a standard tool in the Ethereum development ecosystem, widely adopted for prototyping and verifying smart contracts [3], [31]. The verification process follows these steps:

- Contract Loading: The generated Solidity file is imported into Remix.
- Compilation: The Solidity compiler checks the contract for syntax errors and version compatibility.
- Static Analysis: Remix provides gas usage estimation, detects inefficient or risky patterns, and offers optimization suggestions.
- Simulation: The contract is deployed in a simulated environment to verify the correct execution of functions.

The contract verification process within Remix IDE is illustrated in Fig. 5.

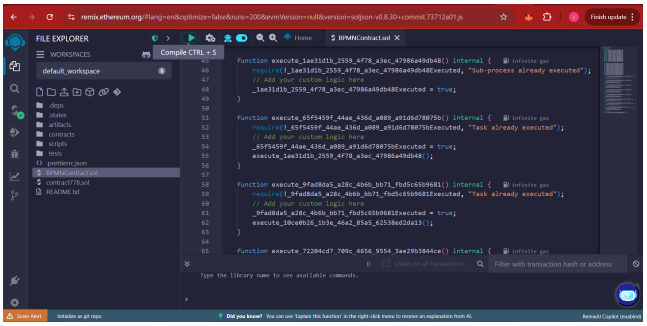

Fig. 5. Remix IDE interface used to compile, simulate, and analyze Solidity smart contracts.

B. Deployment on Sepolia Testnet with MetaMask

Once verified, the contract is deployed on the Sepolia Testnet a public Ethereum test network. MetaMask provides a secure transaction signing interface and is often integrated with Remix for decentralized application testing and deployment [1]. The deployment process includes:

1) Compilation in Remix: The verified contract is compiled using the Remix Solidity compiler.
2) Transaction Signing: Using MetaMask, the deployment transaction is signed and submitted. MetaMask manages the private key and confirms the user's intention.
3) On-chain Deployment: Remix interacts with the Ethereum network via injected web3 (MetaMask), completing the deployment.

The MetaMask interface used for transaction confirmation is shown in Fig. 6.

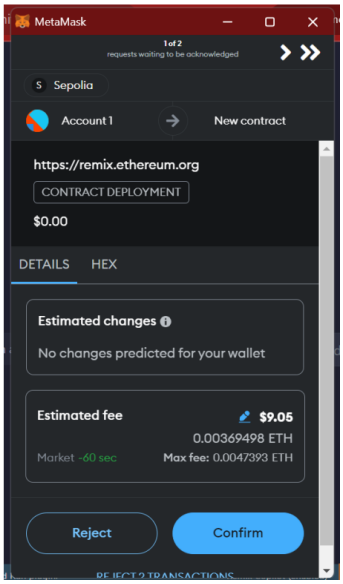

Fig. 6. MetaMask interface used for signing the deployment transaction to the Sepolia Testnet.

C. Transaction Verification with Etherscan

Once the smart contract is successfully deployed, its presence and state on the blockchain must be verified to ensure traceability and successful execution. To ensure the contract has been correctly deployed, the transaction can be traced on Etherscan:

- Transaction Hash: Etherscan shows the hash of the transaction for traceability.
- Status and Block Info: It confirms whether the transaction was successful, its block number, and gas fees.

The transaction record of the deployed contract is verified through Etherscan, as displayed in Fig. 7.

D. Importance of Testnet Deployment

Deploying on a testnet such as Sepolia is essential for testing the contract's behavior without financial risk. It provides a secure and isolated environment for developers to:

- Validate the contract logic under realistic execution conditions;
- Detect gas inefficiencies and potential security vulnerabilities;

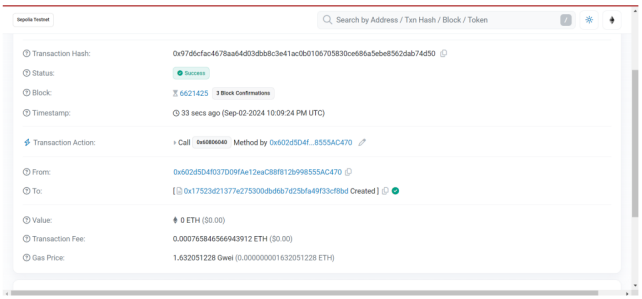

Fig. 7. Verification of the deployment transaction on etherscan (Sepolia Testnet). The figure shows transaction hash, block number, gas fees, and sender/receiver addresses.

- Ensure that all functions operate correctly before mainnet deployment.

E. User Interface of the BPMN-to-Solidity Application

The developed tool includes a user interface for uploading BPMN models, displaying validation results, and generating Solidity code automatically. Fig. 8 displays the BPMN-to-Solidity transformation interface, which enables users to visualize both the original BPMN process and the corresponding generated smart contract. This feature enhances transparency and usability throughout the development lifecycle.

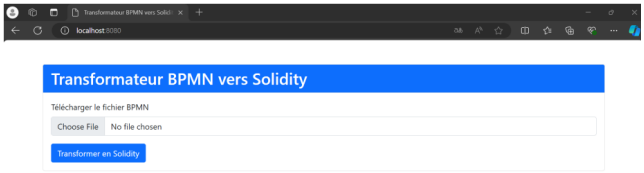

Fig. 8. User interface of the BPMN-to-Solidity transformation application.

This interface allows users to visualize both the original BPMN process and the corresponding Solidity contract, facilitating transparency and usability during the development lifecycle.

F. Summary

This section detailed the final stages of the smart contract development pipeline, focusing on the validation and deployment processes. Using Remix IDE, the generated Solidity code is compiled, analyzed, and tested in a simulated environment to ensure correctness and performance. Once verified, the contract is deployed to the Sepolia Testnet using MetaMask, which handles transaction signing and submission. The successful deployment is confirmed through Etherscan, which provides traceability and transparency via the transaction hash, gas usage, and execution status. The section also presented the user interfaces of the BPMN-to-Solidity transformation tool and deployment tools, highlighting their role in improving usability and reliability throughout the development lifecycle.

VIII. Discussion

While this study primarily focused on defining and implementing the BPMN-to-Solidity transformation methodology, initial testing suggests that the proposed ATL-based approach reduces manual coding effort by approximately 60–70% compared to similar processes in frameworks such as TABS or Lorikeet. The integrated validation and deployment modules enhance both reliability and developer productivity. Nevertheless, future work should include a quantitative evaluation of performance and scalability to empirically validate these preliminary observations.

IX. Conclusion

This study presented a complete model-driven engineering (MDE) approach for automating the generation and deployment of blockchain-based smart contracts from BPMN models. This paper confirms that MDE significantly streamlines the development of smart contracts from BPMN models. By reducing manual coding and promoting automation, the proposed approach enhances accuracy, traceability, and scalability [5], [10].

Despite these benefits, challenges remain in areas such as interoperability between BPMN and smart contract languages, transformation rule optimization, and support for more complex BPM constructs [2], [12]. Future work should explore multi-chain deployment, integration with decision modeling (e.g., DMN) [28], and lifecycle management of long-term processes [18].

The results of this study reinforce the potential of MDE as a foundational approach to building reliable, scalable blockchain systems from abstract business process models.

X. Future Work

Although the current framework demonstrates encouraging results in converting BPMN-based models into executable Solidity smart contracts, multiple directions remain to be explored.

First, future work could focus on extending the transformation rules to handle advanced BPMN constructs, including compensation events, exception flows, and complex gateway logic. This extension would enable better alignment with real-world process requirements and improve semantic accuracy [2], [12].

Second, integrating Decision Model and Notation (DMN) with BPMN could allow for unified modeling of control flows and business rules. Recent studies on hybrid BPMN-DMN architectures in blockchain ecosystems suggest enhanced flexibility and inter-organizational compatibility [28].

Third, enhancing cross-platform deployment capabilities is essential. While this research targets the Ethereum environment, supporting other blockchain frameworks—such as Hyperledger Fabric, Tezos (Michelson) [20], Aptos, and Sui [21]—would broaden the applicability. Interoperability challenges across heterogeneous systems can potentially be addressed using MDE-based solutions like MUISCA [32], [33].

Fourth, the management of process state and lifecycle remains a complex issue, especially for long-running and persistent transactions. Prior research underscores the importance of consistent state handling for enterprise-level blockchain solutions [18], particularly in critical domains such as healthcare and smart grid environments [34], [35].

Fifth, improving accessibility for non-technical users is another valuable direction. Combining low-code platforms with model-driven engineering has been shown to encourage broader enterprise adoption [5]. Such integrations could also support identity management in sensitive environments, including 6G networks and IoT [36], [37].

Finally, socio-economic factors must be taken into account. The adoption of blockchain systems in digital platforms involves trade-offs related to user trust, interface competition, and decision-making behaviors [38], [39]. Future iterations of this work should incorporate these perspectives to enhance practical viability. By addressing these challenges, the proposed framework could evolve into a comprehensive solution enabling secure, transparent, and automated process execution across diverse blockchain ecosystems. Overall, these research directions aim to evolve the current transformation pipeline into a flexible, cross-compatible, and enterprise-ready solution for decentralized business process automation.

Acknowledgment

The authors acknowledge the support provided by the LRIT laboratory, Associated Unit to CNRST (URAC 29), and the Faculty of Sciences at Mohammed V University in Rabat, Morocco.

References

[1] Y. Liu, L. Zhang, A. Papageorgiou *et al.*, "Enabling cross-organization workflows with smart contracts," *Future Generation Computer Systems*, 2024.

[2] F. Milani, L. Garcia-Banuelos, and S. Filipova, "Comparing bpmn and cmmn for blockchain-based business processes," *Business Process Management Journal*, vol. 27, no. 2, pp. 638–657, 2021.

[3] Y. A. Hsain, N. Laaz, and S. Mbarki, "Reviewing mde-based development of ethereum smart contracts," *Procedia Computer Science*, vol. 184, pp. 785–790, 2021.

[4] L. Lucio, Q. Zhang, P. H. Nguyen, M. Amrani, J. Klein, H. Vangheluwe, and Y. L. Traon, "Recent advances in security-centric model-driven engineering," *Advances in Computers*, vol. 93, pp. 103–152, 2014.

[5] S. Curty, F. Härer, and H. G. Fill, "Mde and low-code development for blockchain-based platforms: A survey," in *Proc. BPMDS*. Springer, 2022, pp. 205–220.

[6] Q. Lu, A. B. Tran, I. Weber, H. O'Connor, P. Rimba, X. Xu, and R. Jeffery, "Mde-driven engineering of blockchain applications for business processes," *Software Practice and Experience*, vol. 51, no. 5, pp. 1059–1079, 2021.

[7] A. B. Tran, Q. Lu, and I. Weber, "Lorikeet: A tool for bpmn-based smart contract generation," in *BPM Demos*, 2018, pp. 56–60.

[8] P. Bodorik, C. G. Liu, and D. Jutla, "Tabs: Automated bpmn-to-blockchain smart contract transformation," *Blockchain Research and Applications*, vol. 4, no. 1, p. 100115, 2023.

[9] V. A. D. Sousa and C. Burnay, "Mde4bbis: An mde framework for blockchain-oriented information systems," in *Proc. BCCA 2021*. IEEE, 2021, pp. 195–200.

[10] O. Levasseur, M. Iqbal, and R. Matulevičius, "An overview of model-driven engineering approaches for blockchain-oriented systems," *org ISSN*, vol. 1613, no. 0073, 2021.

[11] E. Y. Nassar, S. Mazen, S. Craß, and I. M. Helal, "Designing blockchain systems through mde methodologies," *Journal of Computer Science*, 2023.

[12] F. Corradini, A. Marcelletti, A. Morichetta, A. Polini, B. Re, and F. Tiezzi, "Executing bpmn choreographies on blockchain infrastructures," *Journal of Blockchain Research*, 2023.

[13] Y. Lu, "Blockchain technologies: Current landscape and future perspectives," *Journal of Industrial Information Integration*, vol. 15, p. 100107, 2019.

[14] A. Bouzidi, N. Haddar, and K. Haddar, "A trace-based meta-model for synchronizing bpmn and uml," *Informatica*, vol. 49, no. 16, 2025.

[15] I. Limaa, T. Martins, and M. Oliveiraa, "Bpmn for optimizing healthcare workflow in cohort studies," *Procedia Computer Science*, vol. 256, pp. 1224–1231, 2025.

[16] M. Markovska, F. P. Milani, and L. Garcia-Banuelos, "Business process modeling in a blockchain ecosystem using bpmn," Ph.D. dissertation, University of Tartu, 2019.

[17] E. Hofmann, U. M. Strewe, and N. Bosia, *Understanding Blockchain Technology: Background and Concepts*, 2018.

[18] C. G. Liu, "Modeling long-term transactions in smart contracts from bpmn," *Blockchain Technology Journal*, 2024.

[19] R. Sierra, M. Eilers, and P. Müller, "Formal verification of vyper smart contracts on ethereum," Ph.D. dissertation, PhD Thesis, 2019.

[20] G. Bau, A. Miné, V. Botbol, and M. Bouaziz, "Static analysis of michelson smart contracts via abstract interpretation," in *Proc. ACM SOAP Workshop*, 2022, pp. 36–43.

[21] R. V. Tonder, "Displaying verified move smart contracts for the sui blockchain," in *Proc. ICSE Companion 2024*, 2024, pp. 26–29.

[22] E. Hofmann, U. M. Strewe, and N. Bosia, "Blockchain emergence: Academic and practical insights," *Blockchain Research Institute*, 2018.

[23] J. Mendling, I. Weber, W. van der Aalst *et al.*, "Business process management and blockchain: Challenges and opportunities," *ACM Transactions on Management Information Systems*, vol. 9, no. 1, pp. 1–16, 2018.

[24] F. Hawlitschek, B. Notheisen, and T. Teubner, "Trust and blockchain in the sharing economy: A critical review," *Electronic Commerce Research and Applications*, vol. 29, pp. 50–63, 2018.

[25] I. Konstantinidis, G. Siaminos, I. Mavridis, M. Dalamaras, and K. Tserpes, "Mapping blockchain applications: A systematic study," *Computer Science Review*, vol. 37, p. 100285, 2020.

[26] S. Seebacher and R. Schüritz, "Blockchain as a foundation for service systems: A literature review," *Service Science*, vol. 11, no. 1, pp. 3–18, 2019.

[27] G. Tripathi, M. A. Ahad, and G. Casalino, "Comprehensive blockchain survey: Foundations, history and open challenges," *Decision Analytics Journal*, vol. 9, p. 100344, 2023.

[28] X. Shen, J. Luo, and H. Wang, "Combining bpmn and dmn for secure cross-organization blockchain workflows," *arXiv preprint arXiv:2412.01196*, 2024.

[29] J. Ladleif, M. Weske, and I. Weber, "Enforcing choreographies on blockchain via mde techniques," in *Proc. BPM 2019*. Springer, 2019, pp. 69–85.

[30] J. S. Cuadrado, L. Burgueno, M. Wimmer, and A. Vallecillo, "Optimizing atl transformations with static analysis and parallel execution," *IEEE Trans. Software Eng.*, vol. 47, no. 9, pp. 1890–1905, 2020.

[31] D. Mertens, J. Kim, J. Xu, E. Kim, and C. Lee, "Smartflow: Workflow management with blockchain provenance support," *Cluster Computing*, vol. 27, pp. 8173–8187, 2024.

[32] E. Dulce-Villarreal, G. Hernandez, J. Insuasti, J. Hurtado, and J. Garcia-Alonso, "Validating muisca: A blockchain interoperability tool for healthcare," *Studies in Health Technology and Informatics*, vol. 323, pp. 255–259, 2025.

[33] E. R. Dulce-Villarreal, E. Moguel, J. Garcia-Alonso, J. P. Cuervo, and J. A. H. Alegria, "Muisca: A universal smart contract proposal for interoperability," 2025, available at SSRN: urlhttps://ssrn.com/abstract=5174813.

[34] B. Boi and C. Esposito, "Blockchain's role in ai-driven cyber-physical medical systems," in *AI Techniques for Sensitive Data in Medical CPS*. Cham: Springer, 2025, pp. 127–142.

[35] R. Baksh, M. Ahmad, and V. Kumar, "Secure authentication and key exchange for smart grids using blockchain," in *AIP Conference Proceedings*, vol. 3283, no. 1. AIP Publishing LLC, April 2025, p. 040027.

[36] G. Zhang, Q. Hu, Y. Zhang, and T. Jiang, "Blockchain-based identity systems for 6g networks," *Digital Communications and Networks*, 2025.

[37] M. Dhinakaran, R. Budhraja, B. Varasree, S. K. Tiwari, A. K. Bindal, S. A. Kumar, and L. Rosca, "Secure identity management in electronics via blockchain," in *Recent Trends in Engineering and Science*. CRC Press, 2025, pp. 27–30.

[38] X. Zhu, Y. Chen, M. Ren, and W. Chu, "Blockchain and e-platforms: Trust vs. channel conflict," *Managerial and Decision Economics*, 2025.

[39] W. Liu, B. Li, G. Zhang, X. Wang, and Z. Wang, "Consumer behavior-aware blockchain adoption and pricing strategies," *Managerial and Decision Economics*, 2025.