BLOCKSTAND

Deliverable D1 – Gap Analysis

Soumya Kanti Datta, Digiotouch

Funded by
the European Union

Blockchain based smart contracts are self-executing with the terms of the agreement directly written into code. This feature allows for the automatic execution of contract terms without the need for intermediaries, thereby reducing the time and cost associated with manual execution. Due to the cryptographic and tamper-proof nature of blockchain, smart contracts offer enhanced security.

The workplan of this project is composed of two tasks. The first task comprises of a preparatory work in terms of gap analysis of common requirements and data formats, templates for smart contract in current standardisation efforts during the first month. Such thorough gap analysis will lead to deeper understanding on how the current situation is preventing interoperability across smart contract tools. This gap analysis has been described in detail below.

A gap analysis is performed from a software engineering and smart contract application development perspective on current standardization efforts for blockchain-based smart contracts. Interoperability: One significant gap lies in interoperability among different blockchain platforms. Currently, there is no universal standard for smart contracts that can seamlessly work across multiple blockchains. Software developers often need to rewrite, or develop from scratch, or adjust smart contracts when deploying them on different blockchain platforms. Initiatives like the InterWork Alliance[1] (IWA) are aiming to address this by developing standards for tokenization and smart contract interoperability. It looks into identifying where current standards, practices, and technologies do not fully support the optimal development, deployment, and interoperability of smart contracts. This analysis will focus on common requirements, data formats, templates, and integration architectures.

Common Requirements for Smart Contract Development

Current state: Standards for developing smart contracts vary widely among blockchain platforms, leading to issues in security, efficiency, and interoperability. Notable frameworks like Ethereum and Hyperledger Fabric provide their own development environments and languages (e.g., Solidity for Ethereum[2] and Chaincode[3] for Hyperledger), which are not interoperable. ISO/TC 307[4] is one international effort aimed at standardization in blockchain and distributed ledger technologies, but specific guidelines for smart contract development are still underdeveloped.

Gaps identified:

- Security standards: There is no universally accepted security standard for smart contracts, despite frequently reported vulnerabilities and breaches. Especially, smart contract applications are vulnerable to various exploits like reentrancy attack[5] (for solidity smart contracts), integer overflows, and unauthorized access. A common set of security standards, including security patterns and anti-patterns, is crucial. A baseline knowledge of security considerations for intermediate Solidity programmers are maintained by ConsenSys Diligence[6] in cooperation with the Ethereum community.
- Development methodology: Lack of standardized development methodologies leads to inefficiencies and errors. A standard process model for the lifecycle of smart contract development could include stages such as requirement gathering, design, coding, testing, deployment, and maintenance.

[1] https://interwork.org/
[2] Solidity is a programming language designed for developing smart contracts that run on Ethereum, https://soliditylang.org/
[3] A chaincode typically handles business logic agreed to by members of the network, so it may be considered as a "smart contract," https://hyperledger-fabric.readthedocs.io/en/release-1.3/chaincode.html#what-is-chaincode
[4] ISO/TC 307 Blockchain and distributed ledger technologies, https://www.iso.org/committee/6266604.html
[I]It occurs when a function makes an external call to another untrusted contract, then the untrusted contract makes a recursive call back to the original function in an attempt to drain funds.
[6] https://consensys.github.io/smart-contract-best-practices/

- Error handling: There is no widespread practice on how to handle execution errors in smart contracts. This varies from one blockchain to another, leading to unpredictable behaviour.
- Upgradeability: Smart contracts are designed to be immutable. However, there should be standardized methods for upgrading contracts or correcting them in response to bugs or evolving business needs and scenarios.
- Interoperability: With multiple blockchain platforms in use, each supporting different smart contract languages and environments, cross-chain interoperability is minimal. Standardizing aspects of smart contract interfaces will facilitate interoperability.
- Legal enforceability: While efforts like the Uniform Law Commission's smart contract model code[7] explore legal aspects, a gap remains in standardizing how legal terms can be translated into unambiguous code within smart contracts.

Data Formats, Structures, and Schemas

Current state: Smart contracts interact with several types of data, but there is a significant lack of standardized data formats or schemas. Each blockchain has its own way of handling data, which can range from simple key-value stores to more sophisticated storage solutions. This variation complicates data integration and consistency across different blockchain systems.

Gaps Identified:

- Standard data structures: There is a need for universally recognized data structures that can be used across various blockchain technologies. These would facilitate easier data sharing and manipulation within and across blockchains. Existing standards like ERC-20[8] for tokens on Ethereum offer some level of data format consistency. However, a gap exists in creating a more comprehensive standard encompassing different data types (e.g., integers, strings, timestamps) used in various smart contract functionalities.
- Encoding standards: There is a lack of standardization in how data is encoded and decoded across different blockchains, which affects the data exchange and interoperability.
- Schema definitions: Clear definitions and standards for schema design specific to blockchain use cases are lacking. Standard schemas would ensure that data stored in smart contracts is consistent, reliable, and predictable.
- Versioning and updatability: Handling updates to data structures without breaking existing contracts is a challenge. Standard protocols for versioning of data schemas used in smart contracts could help manage evolutionary changes.
- Data Provenance: While standards like W3C Provenance[9] address data tracking, a gap exists in standardizing how data provenance can be embedded within smart contracts to ensure data integrity and auditability.

Templates for Smart Contract Development

Templates are standardized blueprints (e.g., website template) that can be used to deploy smart contracts that meet specific criteria, ensuring compliance, interoperability, and reliability.

Gaps Identified:

[7] https://www.uniformlaws.org/viewdocument/guidance-note-regarding-the-relatio?CommunityKey=2c04b76c-2b7d-4399-977e-d5876ba7e034&tab=librarydocuments
[8] https://ethereum.org/en/developers/docs/standards/tokens/erc-20/
[9] https://www.w3.org/2011/prov/wiki/Main_Page

- Lack of template libraries for reusability: There are efforts like OpenZeppelin[10] Contracts offering reusable smart contract components. However, a gap exists in creating a standardized library of templates for common functionalities (e.g., escrow, voting) across different blockchain platforms. There is a scarcity of comprehensive, standardized template libraries that cater to various industry needs, which impedes widespread adoption and reuse.
- Version control: Template versioning is not widely practiced, making it hard to track changes or revert to previous versions when needed.
- Customizability and maintainability: While some templates offer high customizability, there is often a trade-off with complexity, making them less accessible to non-experts. Standardization efforts often lack clear guidelines for modularity and upgradability within smart contract templates, creating a gap in ensuring long-term maintainability.
- Formal verification: It can assist in ensuring the correctness and reliability of smart contracts by mathematically proving their properties. While there are some tools and methodologies available for formal verification, they are not widely adopted due to their complexity and lack of standardisation. Efforts to standardize formal verification techniques for smart contracts could significantly enhance their reliability and trustworthiness.

Reference Architecture for Integration with Distributed Ledger Technologies

Current state: While some efforts like the Enterprise Ethereum Alliance provide reference architectures for specific blockchain environments, comprehensive architecture that integrates smart contracts across different blockchain platforms is missing. Current architectures focus on single-platform solutions and do not address cross-platform integration well.

Gaps Identified:

- Cross-platform communication: There is a need for reference architectures that support communication between smart contracts on different blockchain platforms. This includes not only the direct interaction between contracts but also the integration of off-chain resources and services.
- Modularity and scalability: Current architectures often do not adequately address issues of scalability and modularity. A modular, scalable architecture would allow for parts of the system to be upgraded or replaced without affecting the entire system.
- Compliance and governance: Integrating smart contracts into existing ICT ecosystems and ensuring they comply with local regulations and governance models is challenging. A reference architecture should include compliance modules and governance tools that can be adapted to various regulatory environments.

The integration of smart contracts with blockchain technologies like Ethereum and Hyperledger Fabric involves following as an example -

- Ethereum:
  - Ethereum Virtual Machine (EVM): Standardize how smart contracts are deployed on EVM-compatible chains to ensure they perform uniformly.
  - Data Handling: Standardize data inputs and outputs, especially for complex operations like calls to other contracts and interactions with decentralized storage.
- Hyperledger Fabric:

[10] https://docs.openzeppelin.com/

- Chaincode: Fabric uses a different approach (chaincode) than EVM. Standardizing the development environment and execution model of chaincode can help in better integration.
- Private Data Collections: Define standards for handling confidential data within contracts, crucial for enterprises.

The above-mentioned gaps represent enormous opportunities for standard development and adoption for smart contract. Addressing them through international standards will enhance the security, interoperability, and efficacy of smart contracts across different platforms and (European) industries. Collaborative efforts in bodies such as ISO/TC 307 and IEEE Standards Association will be pivotal in driving these standardisation efforts forward.

DISCLAIMER

The **European Commission** support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.