Towards a Software Toolkit for Specifying and Monitoring Smart Contracts in the Application Integration Domain*

Mailson Teles-Borges[1], Eldair F. Dornelles[1], Fabricia Roos-Frantz[1], Antonia M. Reina-Quintero[2], Sandro Sawicki[1], José Bocanegra[3], Rafael Z. Frantz[1]

[1]Unijuí University – Ijuí/RS – Brazil

mailson.borges@sou.unijui.edu.br

{eldair.dornelles, frfrantz, sawicki, rzfrantz}@unijui.edu.br

[2]University of Seville – Seville – Spain

reinaqu@us.es

[3]Universidad Distrital Francisco José de Caldas – Bogotá – Colombia

jjbocanegrag@udistrital.edu.co

1. Introduction

This paper contributes a toolkit that we have developed to ease the implementation of smart contracts. It is an ongoing project, yet, we have already implemented four of its core components: Jabuti DSL, Jabuti CE, the Transformation Engine, and the Monitoring System. Jabuti DSL is a domain-specific language for writing smart contracts for Enterprise Application Integrations [Dornelles et al. 2022]. It includes constructs for encoding operations frequently found in smart contracts for the application integrations. Our toolkit is aimed at developers of Jabuti DSL smart contracts. It provides a comprehensive programming environment that should encourage the adoption of the language. In the following sections, we will discuss only the last three components.

2. Software Toolkit

Jabuti CE[1] is an editor for writing contracts in Jabuti DSL. It includes a VSCode Plug-in and a Language Server. The VSCode Plug-in connects to the VSCode Editor and integrates with both the Language Server and the Transformation Engine. The Language Server provides editor features such as code formatting, syntax and semantic validation, auto-completion, colour highlighting, and code transformation.

The Transformation Engine transforms contracts written in Jabuti DSL into executable smart contracts. It consists of five components: Grammar Parser, Validators, Canonical Parser, Code Generator, and Code Formatter. The Jabuti DSL grammar is based on ANTLR, so the Grammar Parser also uses ANTLR to generate the Abstract

*Research funded by the Co-ordination for the Brazilian Improvement of Higher Education Personnel (CAPES) and the Brazilian National Council for Scientific and Technological Development (CNPq) under grants 311011/2022-5, 309425/2023-9, 402915/2023-2. Antonia M. Reina has been funded by projects PID2020-112540RB-C44 and TED2021-130355B-C32. Thanks to Carlos Molina-Jimenez from Cambridge University for his valuable feedback in early versions of this work.

[1]The source code of Jabuti DSL, Jabuti CE, the Transformation Engine, and the Monitoring System is available on GitHub: https://github.com/gca-research-group/smart-contract-execution-monitoring-system?tab=readme-ov-file#project-repositories

Syntax Tree (AST) of the smart contract. The AST is a structured data format that links each token value in a hierarchical sequence of relationships. Validators, also implemented using ANTLR, perform syntactic and semantic checks. The Canonical Parser converts the generated AST into the format required by the Code Generator. The Code Generator uses a *template rendering library*, currently *ejs*[2], to transform the Jabuti DSL smart contract into the format of target platform. Finally, the Code Formatter corrects formatting errors.

The Monitoring System mediates between integrated applications and blockchain platforms. It manages blockchain connections and smart contract execution. It captures execution events and detects clause violations. The system forwards contractual events to integrated applications. Monitoring System is event driven and is activated on demand. It consists of three main components: Event Handler, Contract Invoker, and Event Updater. The Event Handler polls and prepares events awaiting processing. The Contract Invoker connects to the target blockchain, executes the smart contract, and captures execution events. Lastly, the Event Updater evaluates the Contract Invoker's response and makes the results available to the integrated application.

3. Usage of the toolkit in a case study

Imagine a contract with a single rule between a shop-owner and a supplier that dictates that *delivery trucks are expected only from 6:00 pm to 11:59 pm*. Next, imagine that Alice (a developer) proceeds to implement it using our toolkit. (1) Alice uses Jabuti CE to encode the rule in Jabuti DSL. The result is a Jabuti DSL smart contract stored in a .jabuti file. (2) In this example, she wishes to deploy the smart contract in Hyperledger Fabric, thus she uses the Transformation Engine to transform it into Golang (Solidity would be another alternative). (3) She deploys the Golang smart contract into the Monitoring System to finish her task. (4) When a delivery is completed, the shop-owner generates an event reporting the truck's arrival and departure times and sends it to the Monitoring System. (5) The Monitoring System launches the smart contract into execution. (6) If the smart contract detects a rule breach, it sends a violation event to the shop-owner and the supplier to enable compensatory actions.

The main advantages of this approach are: firstly, Jabuti DSL can encode rules directly and intuitively, Jabuti CE automates code specification and transformation; secondly, the Monitoring System simplifies the connection and management of the interaction between the integration and the blockchain; it lowers the adoption barrier; finally, the blockchain serves as an immutable storage of transaction records.

4. Future Work

The adoption of AI to automate the transformation process that runs into the Transformation Engine is in our plans. Managing and deploying a Hyperledger Fabric network can be cumbersome, even for testing purposes. Therefore, we would like to implement a tool to automate the deployment of local Hyperledger Fabric networks. We plan to evaluate the performance of the Monitoring System under stressful conditions.

References

Dornelles, E. F., Parahyba, F., Frantz, R. Z., Roos-Frantz, F., Reina-Quintero, A. M., Molina-Jiménez, C., Bocanegra, J., and Sawicki, S. (2022). Advances in a DSL to specify smart contracts for application integration processes. In *Ibero-American Conference on Software Engineering (CIbSE)*, pages 46–60.

[2]https://ejs.co/