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

Domain-Specific Languages (DSLs) abstract problems through high-level constructs closely aligned with their domain. Tools such as a runtime for code execution or editors that support the DSL syntax play a crucial role in facilitating or enabling DSL adoption for domain users [Bünder and Kuchen 2020]. However, the DSL ecosystem often lacks the necessary attention. Conversely, Jabuti DSL [Dornelles et al. 2022], a DSL for the Enterprise Application Integration (EAI) domain, has been proposed to specify integration constraints, such as the maximum number of requests allowed, and has been enhanced by tools that enable the Jabuti DSL implementation in real scenarios.

The Jabuti DSL ecosystem uses a VSCode-based implementation instead of adopting Model-Driven Tools like the Eclipse Modelling Framework. On the one hand, these tools facilitate the implementation process; on the other hand, they often cater to IT professionals rather than domain experts. Our goal is to deliver a comprehensive environment that enables the transformation of Jabuti DSL smart contracts into executable smart contracts, automating the deployment, monitoring, and execution process within the blockchain platform, and finally triggering events to compensate for potential clause violations. In the next section, we outline the Software Toolkit, its components, and their primary functionalities.

2. Software Toolkit

The software toolkit includes Jabuti CE, a Transformation Engine, and the Monitoring System.

– Jabuti CE[1] is an editor for writing contracts in Jabuti DSL. It includes the VSCode Plug-in and the Language Server. The VSCode Plug-in connects to the VSCode Editor and integrates with both the Language Server and the Transformation Engine. The

*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. Thanks to Carlos Molina-Jimenez from Cambridge University for his valuable feedback in early versions of this work.

[1]https://github.com/gca-research-group/jabuti-ce-vscode-plugin

Language Server provides editor features such as code formatting, syntax and semantic validation, auto-completion, colour highlighting, and code transformation.

– **Transformation Engine**[2] 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[3] is based on ANTLR, so the Grammar Parser also uses ANTLR to generate the Abstract 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* [EJS 2025], to transform the Jabuti DSL smart contract into the format of target platform. Finally, the Code Formatter corrects formatting errors.

– **Monitoring System**[4] is as an intermediary layer 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 events to integrated applications, which can use them for internal business processes. Designed as an event-driven software, the Monitoring System triggers its services 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. Future Work

We plan to evaluate the adoption of artificial intelligence to automate the transformation process that runs within the Transformation Engine. Since we are working within an enterprise context, we have chosen Hyperledger Fabric as the blockchain platform to validate our tools. Managing and deploying a Hyperledger Fabric network can be cumbersome, even for testing purposes. Therefore, we propose developing a tool to automate the deployment of local networks. As a middleware component, the Monitoring System must endure high workloads. Thus, we intend to evaluate its performance under stressful conditions.

References

Bünder, H. and Kuchen, H. (2020). Towards multi-editor support for domain-specific languages utilizing the language server protocol. In *Model-Driven Engineering and Software Development (MODELSWARD)*, pages 225–245.

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.

EJS (2025). EJS. https://ejs.co/. Accessed: January 20, 2025.

[2]https://github.com/gca-research-group/jabuti-ce-transformation-engine

[3]https://github.com/gca-research-group/jabuti-ce-jabuti-dsl-grammar

[4]https://github.com/gca-research-group/smart-contract-execution-monitoring-system