Politecnico di Torino
Master's Thesis in Computer Engineering
Summary for the evaluation committee

Smart Contract Analysis and Visualization Software

Candidate: Lorenzo Gangemi
Supervisors: Prof. Valentina GATTESCHI & Eng. Emanuele Antonio NAPOLI

Introduction and State of the art

Decentralized application are widely available thanks to the rapid evolution of Blockchain technology. Smart contracts, self-executing programs that automate transactions without intermediaries, are central in those applications. However the complexity of Solidity development represent a challenge, particularly for non-technical users, as errors can lead to financial losses without a proper way of patching.

The state of the art provides frameworks like Foundry and Truffle to streamline the development process of smart contracts. Foundry excels in speed, fuzz testing, and local blockchain simulations, while Truffle offers a complete suite for testing, deployment and network management.

Additionally, many low-code or no-code platforms such as DappBuilder or AuditWizard simplify the blockchain application creation with templates, AI assistance and security tools. However, limitations in graphical interfaces and integrated security analysis highlight are a gap in the current solutions.

This thesis describe a system built and designed to simplify smart contract development by enhancing code comprehension, improving security and describing the functionalities in a human readable way.

The proposed tool employs three key features:

- **Graphical representation**: abstracts code into a visual format, allowing the users to interact with the contract, without having to directly modify the code
- **Security analysis**: detects vulnerabilities, reduces the risk of errors and ensures higher reliability
- **Large Language Model (LLM) integration**: generates human readable explanations of the contract elements, specifies the functional relationships, and highlights improvable functions

System Architecture and Design

The system employs a microservices architecture that focus on modularity, scalability and fault tolerance. It is composed of three Golang microservices: Codec, Auditor and AI Assistant . The Backend-For-Frontend act as the middle man, facilitating communication between these microservices and the Flutter based frontend.

Each microservice handles distinct tasks, all working towards the same single analysis.

- **Codec**: Parses Solidity code into JSON for structured manipulation and back.
- **Auditor**: Makes use of Slither for static analysis.
- **AI Assistant**: Uses GPT-3.5 APIs for element descriptions, functional relationships, and identifying improvement opportunities.

Each of these element is interacting with the others through the use of gRPC communication. The BFF integrates the clients of each of the service, and it orchestrate the analysis requests and the microservices responses. Each incoming request from the client is handled as a single Task, which is assigned an unique ID and is then stored in the BFF. The client can then refer to the analysis by using the Task ID.

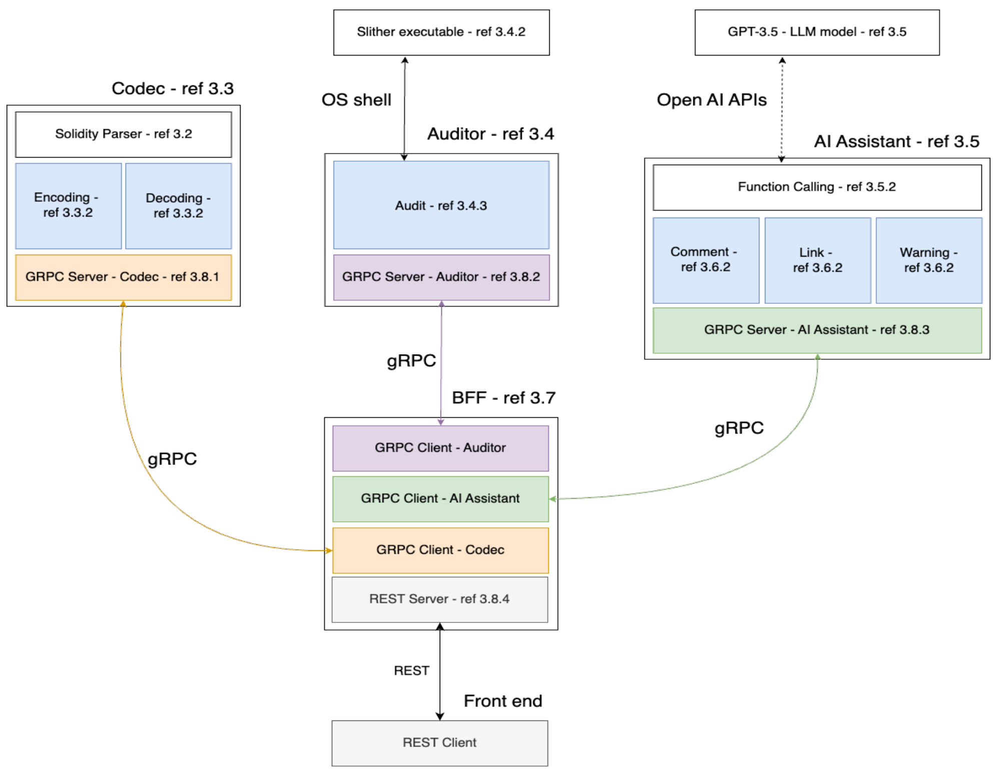

Figure 1: Backend System Architecture

Custom Solidity Parse

By using ANTLR, a custom solidity parser have been developed with the scope of transforming Solidity code into a structured JSON format that would suit the needs of the system. ANTLR operates by generating an abstract syntax tree and running though it, all of that while calling specific listeners that can be integrated with custom code. This allows to specifically build the structure needed and obtain the UML present at Figure 3.3 in the thesis. The obtained structure is developed with the ability to perform the inverse operation, as it can be converted back into functional solidity code, even after modifications are applied.

Function Calling and Setup Prompts

GPT-3.5 comes with the Function Calling features. This allows the model to return structured output in a predefined format, facilitating the integration of responses without the need for complex regex to extract data. In the project, function calling is employed within the AI Assistant service for each of its functionalities. These prompts define the task's scope by providing example input and outputs, and detailed step-by-step instructions for the model to follow. Function calling require also some specific integrations directly on the code, so that it can be ready to receive and read the results. Later, a specific input prompt provides the actual smart contract code and structure to the model, triggering the function call. This ensures consistent, structured responses that are directly incorporated int the JSON structure.

Frontend

The frontend, built in Flutter, provides a user friendly interface for interacting with the smart contract analysis result. Its core feature is the Editor Grid which allows users to visually explore the components, drag them around, view the functional relationship through connection lines. Users can interact with elements to view or edit their properties.

In the **Code Page** user can take care of contract uploads and interact with analysis results in the Editor Grid. The **Contract Page** shows the smart contract plain code alongside its description for easy understanding. The **Settings Page** allows to configures OpenAI API key and theme preferences.

Software/Remix.com - MultisignWallet.sol				
	Identify the types of the three parameters required by the `submitTransaction` function, listing them in order	Determine the five types returned by the `getTransaction` function, listing them in order	Identify all functions within the contract that emit the `SubmitTransaction` event	Describe the outcome when `revokeConfirmation` is called with a `_txIndex` that does not correspond to any existing transaction
AVG	**57.97%**/72.98%	**56.30**/70.28%	**63.39%**/70.51%	**64.78%**/78.02%
STD	0.30/0.26	0.32/0.29	0.28/0.25	0.26/0.24
p-value	0.06	0.09	0.21	0.06

Table 1: Survey results: MultisignWallet.sol - Thesis Software / Remix.com

Software/Remix.com - FundMe.sol				
	Identify which function invokes `getConversionRate` from the `PriceConverter` library	Explain the purpose and functionality of the `latestRoundData` function	Describe the implementation differences between the `withdraw` and `cheaperWithdraw` functions that contribute to the reduced cost of the latter	Identify the vulnerability affecting the `setOwner` function and provide a corrected version of the function with the necessary modifications
AVG	**69.53%**/74.19%	**69.56%**/76.38	**66.55%**/75.52%	76.54%/**71.39%**
STD	0.28/0.24	0.25/0.23	0.35/0.23	0.31/0.27
p-value	0.30	0.20	0.29	0.30

Table 2: Survey results: FundMe.sol - Thesis Software / Remix.com

Results

A survey was conducted to evaluate the tool functionalities and effectiveness compared to the Remix.com IDE. The targets where individual with varying levels of expertise in programming and blockchain. It consisted in two main sections: a task based evaluation followed by a NASA-TLX. Participants complete the tasks using two smart contracts `MultisignWallet.sol` (Table 1) and `FundMe.sol` (Table 2), alternating between the thesis software and Remix. The questions included identifying parameters, functions, events, vulnerability and providing descriptions and optimizations. The NASA-TLX aimed to capture the workload over different dimensions.

The first values in the tables (Table 1 and 2) represent the average score obtained by the users. The value in percentage is the time taken by the user to provide the answer, relative to a time limit of five minutes. Given a wrong answer, the score was considered failed even within the time limit, with a resulting value of 100%.

The survey results highlight the effectiveness of the developed tools in simplifying smart contract analysis. The tool demonstrate an advantages in tasks that makes use of the graphical representation and relational insights, particularly from the `MultisignWallet.sol` contract, where average task completion times were significantly lower compared to Remix. For `FundMe.sol`, while the tool excelled in most tasks, one question about resolving a vulnerability showed Remix performing slightly better, indicating that user's skills were more important and the tool was not able to provide an advantage.

On the contrary, the analysis revealed limitations in proving the software's superiority. The variability in user responses and the lack of statistically significant p-values suggest that the observed results may not be robust. The users participating in the thesis survey presented a wide pool of programming skills and blockchain knowledge. Deciding to apply a value of 100% (failed) when tasks' responses were technically wrong, even if given within the time limit, could have resulted in an inflated standard deviation, potentially impacting the p-values calculated. Future evaluations could benefit from a more precise scoring system that is able to distinguish between levels of correctness.

The NASA-TLX results (Figure 2) further validate the user-friendly design of the tool. Participants reported lower mental and temporal demand, with a significant reduction also in frustration. The overall NASA score highlights the tool's accessibility and efficiency, even for user with limited experience in programming or blockchain.

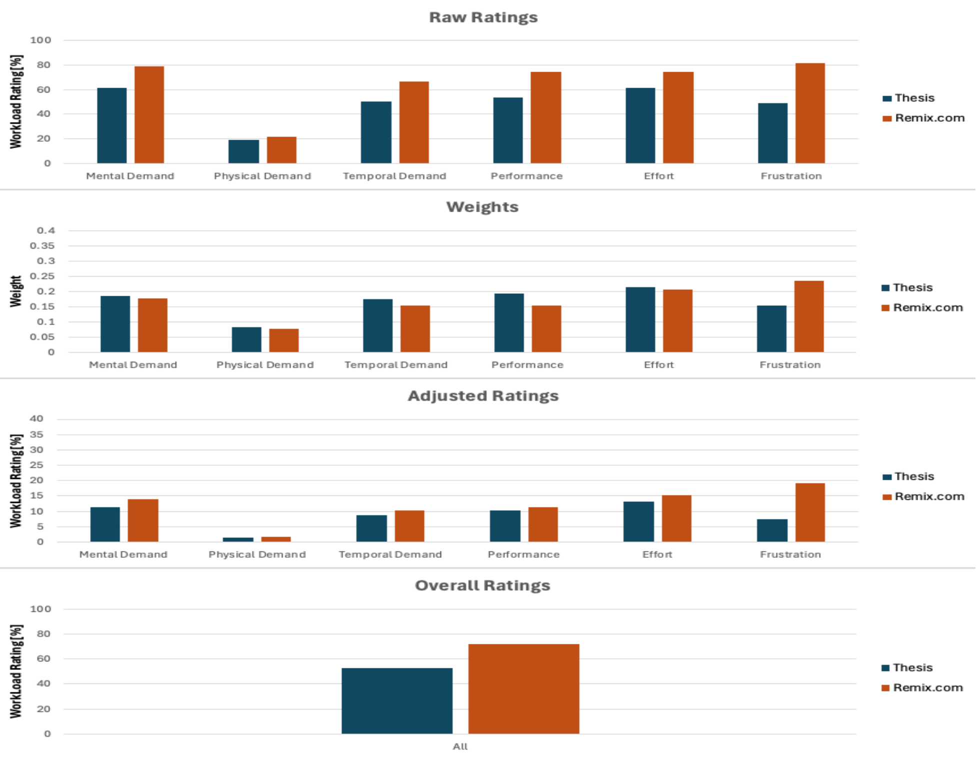

Figure 2: NASA-TLX Results