University of Camerino

School of Sciences and Technologies

Course in MSc in Computer Science (Class LM-18)

An Approach for Loosely Specified Smart Contracts

Student
Alessio Prosperi

Matricola 127834

Supervisor
Flavio Corradini

External supervisor
Andrea Morichetta
Alessandro Marcelletti

A.A. 2023/2024

Contents

Listings

List of Figures

1. Abstract

In numerous application domains, it's impossible to fully define the entire process model beforehand. Although some aspects of the process model are clear during the design phase, other parts remain uncertain and can only be determined during execution. To effectively manage this uncertainty, decisions about the precise details of certain parts of the process must be postponed until run-time. [MR12]

This project aims to develop more types of innovative frameworks designed to address the problem Challenges posed by loosely specified process (LSP) Smart Contracts, allowing high flexibility in scenarios represented. Smart contracts allow users to customize dynamically and select specific behaviors, adapting them to the evolving needs of the context.

The goal is to create an innovative smart contract framework that leverages LSP principles to improve the adaptability and resilience of organizations across various industries, ultimately providing a robust set of tools for managing complex and fluid scenarios. To achieve this outcome, I have utilized Remix and Solidity, as well as Solidity Assembly, to ensure the development of a versatile and efficient framework.

2. Introduction

In the realm of blockchain technology, smart contracts have revolutionized the way transactions and processes are executed, providing transparency, security, and immutability. These self-executing contracts, with the terms of the agreement directly written into code, eliminate the need for intermediaries, thereby reducing costs and enhancing efficiency. However, traditional smart contracts often encounter limitations when dealing with Loosely Specified Processes (LSP). These processes, characterized by their inherent flexibility and evolving nature, require a more dynamic and adaptable approach than what conventional smart contracts typically offer.

Traditional smart contracts are typically rigid and follow a predefined sequence of actions, which makes them unsuitable for environments where conditions and requirements can change rapidly. In such dynamic environments, the need for real-time adaptation and flexibility is paramount. This rigidity can be a significant hindrance in scenarios where processes must adapt to unforeseen changes and continuously evolving requirements. For example, in sectors such as supply chain management, healthcare, and decentralized finance (DeFi), the ability to adjust to real-time changes is crucial for maintaining efficiency and effectiveness.

This project addresses the challenges posed by LSPs by developing an innovative smart contract framework. The proposed solutions are designed to provide users with the ability to dynamically customize and select specific behaviors within the smart contract. This flexibility ensures that the smart contract can adapt to the changing needs and contexts in which it operates, making it a robust tool for managing complex and fluid scenarios. By allowing users to modify the behavior of the smart contract as conditions change, this framework enhances the utility and applicability of smart contracts in a wide range of industries.

Using this adaptable framework, users can enjoy the benefits of blockchain technology while overcoming the rigidity traditionally associated with smart contracts.

The ability to adjust contract behaviors in real-time not only enhances user experience but also opens up new possibilities for applications in various fields. For instance, in supply chain management, the dynamic customization of smart contracts can help track goods more efficiently, respond to disruptions, and ensure compliance with regulatory requirements. In healthcare, adaptable smart contracts can be used to manage patient data, handle insurance claims, and coordinate care among multiple providers more effectively.

Moreover, in the realm of decentralized finance (DeFi), the ability to adjust contract terms in response to market conditions can provide more flexible and responsive financial instruments. This adaptability can lead to the creation of innovative financial products that better meet the needs of users and respond to the volatility and dynamism of the financial markets.

Beyond these sectors, the flexible smart contract framework can be applied to areas such as legal agreements, intellectual property management, and automated compliance systems, further expanding the potential applications of blockchain technology.

3. Background

For the background of this project, I utilized Solidity, RemixIDE, and conducted an in-depth analysis of the challenges associated with Loosely Specified Processes (LSP) in smart contracts.

Solidity is a high-level programming language, similar to JavaScript, specifically designed to create smart contracts on the Ethereum platform. Its syntax and advanced features allow the definition of complex smart contracts capable of handling a wide range of operations and conditions.

For the development of this project, RemixIDE, a powerful and widely used integrated development environment within the Ethereum community, was employed. RemixIDE offers advanced tools for writing, debugging, and deploying smart contracts, significantly facilitating the development process. Its user-friendly interface and extensive functionalities make it possible to efficiently test and verify code behavior.

The analysis of LSP issues in smart contracts was a fundamental part of this project. Loosely Specified Processes represent a set of processes with less detailed or non-rigid specifications, requiring a high degree of flexibility and adaptability. This type of process presents several challenges, including dynamic management of conditions and actions, the ability to respond to unforeseen events, and the need to customize behaviors in real time.

3.1 Loosely Specified Processes

Loosely Specified Processes (LSP) represent a class of processes characterized by their flexible and adaptable nature, which do not follow a linear and predefined sequence of actions. These processes are designed to operate in environments where conditions and requirements can change rapidly, and where real-time adaptation is required. Unlike rigidly structured processes that adhere to a determined and immutable workflow, LSPs allow for greater freedom of action and decision-making, enabling operators to respond promptly to external variables. [MR12]

The inherent adaptability of LSPs is crucial in contexts where unpredictability and volatility necessitate a dynamic approach. For instance, in sectors such as healthcare, emergency management, or software development, LSPs enable the handling of complex and evolving situations without being constrained by a fixed plan. This type of process is particularly useful in crisis scenarios, where decisions must be made quickly based on the most recent and accurate information available.

Another distinguishing aspect of Loosely Specified Processes is their capacity to integrate continuous feedback, adapting to new information and changing contexts. This feature allows teams to be more responsive and to constantly improve their operations, optimizing outcomes and reducing resource waste. Additionally, the flexible approach

of LSPs fosters innovation, as it encourages experimentation and iteration, which can lead to more effective and creative solutions.

Implementing LSPs requires an organizational culture that values collaboration, open communication, and knowledge sharing. Technological tools play a crucial role in supporting these processes, providing platforms for information management, real-time collaboration, and data analysis. However, it is equally important that the people involved possess the necessary skills and mindset to work in a dynamic and uncertain environment.

Finally, Loosely Specified Processes promote greater organizational resilience. Being less susceptible to failures resulting from overly rigid plans, organizations that adopt LSPs can better adapt to disruptions and maintain operational continuity even under adverse conditions. This makes LSPs a key element for organizations operating in highly competitive and rapidly evolving markets, where the ability to adapt can make the difference between success and failure.

3.1.1 Example (Patient Treatment Processes)

Patient treatment processes in a hospital typically comprise activities related to patient intake, admission, diagnosis, treatment, and discharge . Typically, such processes comprise dozens up to hundreds of activities and are long-running (i.e., from a few days to several months). Furthermore, the treatments of two different patients are rarely identical; instead the course of action greatly depends on the specific situation; e.g., health status of the patient, allergies and chemical intolerances, decisions made by the physician, examination results, and clinical indications. This situation can change during the treatment process, i.e., the course of action is unpredictable.

3.2 Ethereum

The Ethereum (ETH) blockchain, inaugurated in 2015 by Vitalik Buterin together with a team of visionary developers, [Etha] is a decentralized platform that has redefined a new vision of how digital transactions and data can be recorded and managed.

Ethereum operates as a peer-to-peer system that allows you to record transactions and implement smart contracts. This platform was designed to be significantly more adaptable than Bitcoin's infrastructure. ETH makes use of a Turing complete programming language, meaning that developers have the ability to create a wide range of applications on the platform.

3.2.1 Structure of Ethereum

The ETH Blockchain is decentralized, meaning it operates on a network of nodes distributed across the world. There is no control by a central authority, but rather it is managed by the community of users who participate in the network.

The main features are the use of smart contracts and the creation of digital assets, which several of the standards of this network allow to implement. [cri]

Ethereum is currently the blockchain with the largest number of applications and uses in existence. It is the blockchain on which the largest number of nfts are traded and on which the largest number of smart contracts are executed every day globally. [EG]

3.3 Smart Contract

Smart contracts are self-executing programs with the terms of the agreement directly written into code. These contracts operate on blockchain platforms, ensuring transparency, security, and immutability. [di)19]

However, one of the main challenges in designing smart contracts is managing "looseness," or the flexibility and adaptability in processes with loosely specified requirements (Loosely Specified Processes, LSP).

Below are the primary issues associated with this looseness, particularly when writing smart contracts in Solidity. One of the significant difficulties is managing conditions that can change over time. LSPs often require that smart contracts adapt to new conditions or changes in the operational context without having to rewrite or redeploy the contract. [MR12] This can be complex since smart contracts, once deployed on the blockchain, are immutable by design.

Solidity, the programming language used for writing these contracts, pro- vides tools and constructs to handle such dynamic conditions, but implementing them requires careful planning and expertise.

3.3.1 Solidity

The language used to develop smart contracts on the ETH platform is Solidity.

Solidity is a high-level object-oriented programming language, and was specifically designed to write code that defines the rules and conditions of smart contracts, allowing them to operate autonomously and securely. [Aut]

We can define it as similar to C++, Python and JavaScript in syntax, which makes it relatively easy for programmers to learn and allows you to establish conditions for executing transactions, authorization criteria and security aspects.

Once the code is written, it is compiled into bytecode to run on the blockchain.

Bytecode

Bytecode is a low-level representation of the source code written in Linux. high-level programming skills. It is designed to be run by one specific virtual machine, often used in scripting language applications, interpreted or in web development contexts. In this case it uses the Ethereum virtual machine (EVM).

3.3.2 RemixIDE

Remix IDE is an integrated development environment (IDE) that allows you to write, test and deploy smart contracts on the Ethereum blockchain. In figure 3.1 is show remixIDE. It is a powerful and versatile tool that supports Solidity, the programming language used to create smart contracts.

Remix IDE is particularly appreciated for its ease of use and the numerous features it offers smart contract developers. It is accessible directly from the web browser, without the need for installation. This makes it extremely convenient for developers who want to start working with smart contracts quickly.

The interface is intuitive and easy to navigate, even for new users, with a clear layout that includes panels for writing code, compiling, and deploying.

Remix is extendable via plugins, allowing developers to add features as needed. There are plugins for gas management, static code analysis, documentation, and more. Developers can easily share their code and projects with others, making collaboration easier. A tool that allows you to connect your local filesystem with Remix IDE, making it easy to edit local files directly from the IDE.

Thanks to the integrated test environment, developers can test their contracts without incurring costs associated with mainnet transactions. [Com]

Figure 3.1: RemixIDE

Gas

"Gas" is a unit of measurement that represents the cost of carrying out operations or transactions on the network. Each operation performed on the Ethereum blockchain requires a certain amount of gas, which is paid for in Ether (ETH), the native cryptocurrency of Ethereum. It refers to the unit that measures the amount of computational effort necessary to perform specific operations on the Ethereum network, gas is therefore the fuel that allows the network to operate.

Gas prices are denoted in gwei. Each gwei is equivalent to 0.000000001 ETH. [Ethb]

Each operation requires a certain number of gas units based on its complexity and the necessary computational load. More complex operations require more gas. Users must specify the amount of gas they wish to allocate for an operation and the price they are willing to pay for each unit of gas (in terms of Ether).

The total cost of an operation is, therefore, calculated as a unit of gas used (energy cost) multiplied by the sum of the basic commission (set by protocol) and the priority commission (how much you are willing to pay more).

4. Methodology

In this section, we outline the methods and procedures utilized to develop and test the function capable of deploying smart contracts at runtime. This research is primarily applied and experimental. The focus is on developing a practical solution to a problem identified in the domain of blockchain and smart contracts. The research adopts a hands-on approach, involving coding, deployment, and testing of smart contracts on a blockchain platform.

4.1 Possible solutions

The first case study to address these challenges was to develop a function that can dynamically handle various types of input without predefined constraints on the length or number of parameters. This solution allows for greater flexibility and adaptability in managing dynamic conditions and customizing behaviors.

4.1.1 Function Overview

The primary solution involves creating a function that accepts a byte array as input and decodes it into the specified data type. This function can take inputs of any type without requiring a fixed number of parameters, and returns two arrays: one containing the converted strings from bytes to their actual types and the other containing the corresponding types of the strings.

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3
4  contract Decoder {
5     function decodeStrings(bytes[] memory byteArrays, string[] memory typesArray)
6     public pure returns (string[] memory) {
7        require(byteArrays.length == typesArray.length, "Arrays length mismatch");
8
9        string[] memory decodedValues = new string[](byteArrays.length);
10
11       for (uint i = 0; i < typesArray.length; i++) {
12          if (keccak256(bytes(typesArray[i])) == keccak256(bytes("string"))) {
13             decodedValues[i] = string(byteArrays[i]);
14
15          } else if (keccak256(bytes(typesArray[i])) == keccak256(bytes("int"))) {
16
17             int decodedInt = abi.decode(byteArrays[i], (int));
18             // Convert the integer into a string
19             decodedValues[i] = intToString(decodedInt);
20          }
21              else if(keccak256(bytes(typesArray[i])) == keccak256(bytes("bool"))) {
22
23                decodedValues[i] = byteToBoolAndString(byteArrays[i]);
```

```
24          }
25       }
26
27       return decodedValues;
28    }
29
30    function intToString(int _i) internal pure returns (string memory) {
31       if (_i == 0) {
32          return "0";
33       }
34       uint256 absoluteValue = uint256(_i < 0 ? -_i : _i);
35       uint256 maxLength = 100; //Maximum length of the integer as a string
36       bytes memory reversed = new bytes(maxLength);
37       uint256 i = 0;
38       while (absoluteValue > 0) {
39          uint256 remainder = absoluteValue % 10;
40          absoluteValue /= 10;
41          // Convert the number to an ASCII character
42          reversed[i++] = bytes1(uint8(48 + remainder));
43       }
44       bytes memory s = new bytes(i); // Actual size of the string
45       for (uint256 j = 0; j < i; j++) {
46          s[j] = reversed[i - j - 1]; // Inverti la stringa
47       }
48       return string(s);
49    }
50
51
52     function byteToBoolAndString(bytes memory data) internal pure returns (string memory) {
53       require(data.length == 1, "Invalid input length");
54
55       if (data[0] == 0x00) {
56          return "false";
57       } else if (data[0] == 0x01) {
58          return "true";
59       } else {
60          revert("Invalid boolean value");
61       }
62    }
63 }
```

Listing 4.1: First Function

Now we will analyze the code.

The function decodeStrings at lines 5, takes two arrays as input,byteArrays containing byte-encoded data and typesArray specifying the data types. It ensures the lengths of both arrays match and decodes each byte array based on the specified type (string, int, or bool) and stores the decoded value in the decodedValues array. After it, he will return the decodedValues array.

The IntToString and byteToBoolAndString functions at lines 30 and 52, are used respectively to convert the various values into strings.

To run remix IDE, you need to go to the DEPLOY RUN TRANSACTIONS screen and click on the deploy command. In figure 4.1 is show how execute the deploy of smart contracts.

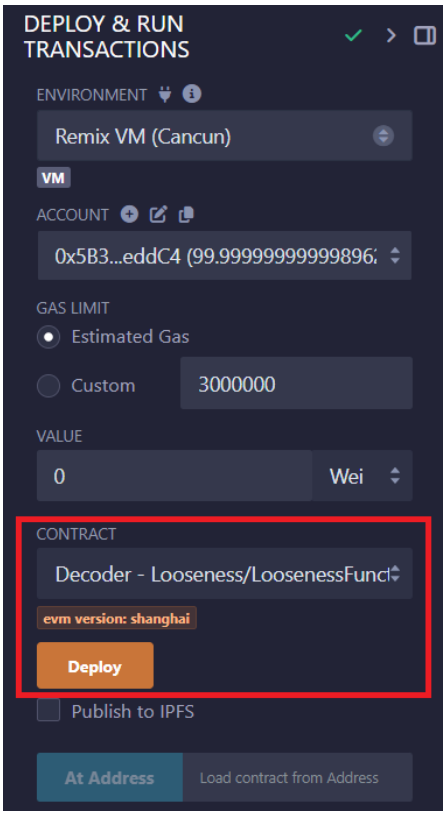

Figure 4.1: Deploy of smart contract

In the figure 4.2 we can see the result of the execution.

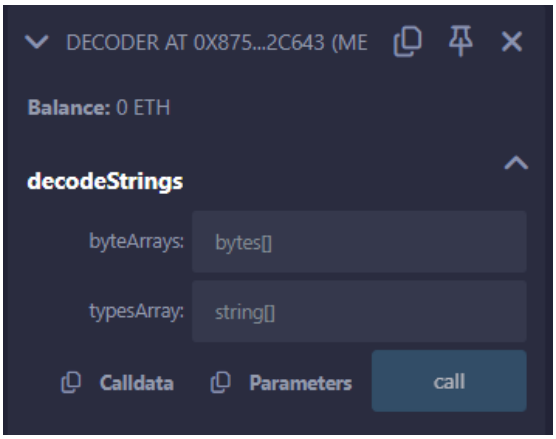

Figure 4.2: Example of a first function

Let's try to insert this input: (8, ciao, true); so we have to put them in their encoded version, we can see it in figure 4.3:

["0x0000000000000000000000000000000000000000000000000000000000000008", "0x6369616f", "0x01"]

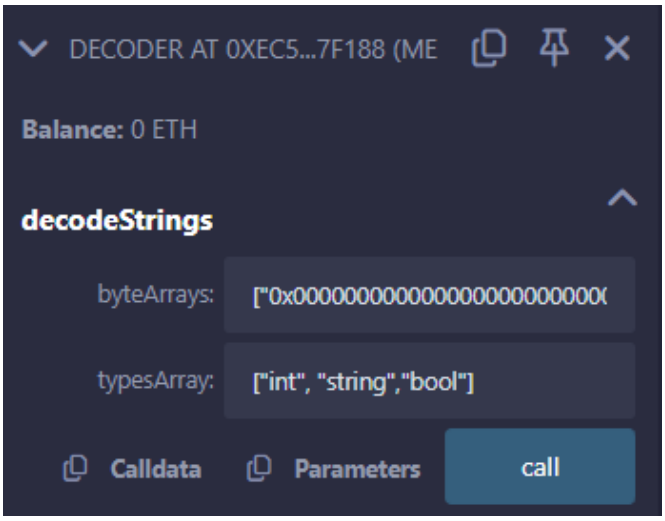

Figure 4.3: Input of the Function

and obviously we have to enter the type of each inserted string: [”int”, ”string”,”bool”] once executed we will obtain this result (figure 4.4).

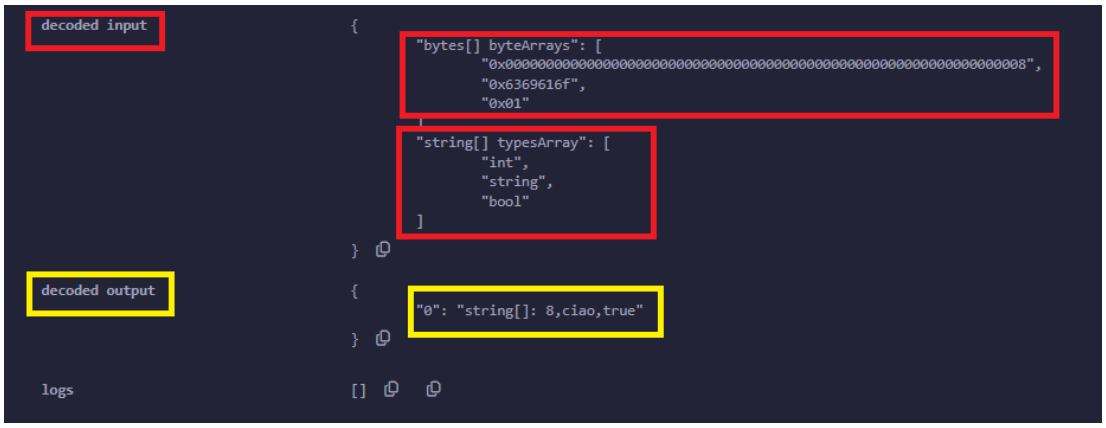

Figure 4.4: Result of the Function

As you can see, the inputs inserted are highlighted in red, while the relative outputs due to the conversion are highlighted in yellow.

4.1.2 Selector

The second question we asked ourselves was to find a way to perform different functions in runtime. So I developed an advanced function in Solidity that allows executing other functions at runtime using their selector. This capability is particularly useful for implementing flexible and modular contracts, where functions can be dynamically called based on certain conditions or inputs.

A function selector is a four-byte sequence derived from the first four bytes of the hash of a function signature. The function signature consists of the function name and the parameter types. For example, for a function defined as:

```
1
2  function transfer(address _to, uint256 _value),
3
4  the selector would be calculated as :
5
6  bytes4(keccak256("transfer(address,uint256)")).
```

Listing 4.2: Example how selector works

In this contract, the function CallFunction at line 38, allows you to dynamically call one of these functions (A,B,C) based on a given selector.

```
1
2  // SPDX-License-Identifier: MIT
3  pragma solidity ^0.8.0;
4
5  contract FunctionContract {
6     //Function A that adds two integers
7     function A(uint256 x, uint256 y) public pure returns (uint256) {
8        return x + y;
9     }
10
11    // Function B that returns true if the number is greater than 50 and returns 1,
12 otherwise it returns false and returns 0
13    function B(uint256 x) public pure returns (bool, uint256) {
14       if (x > 50) {
15          return (true, 1);
16       } else {
17          return (false, 0);
18       }
19    }
20
21    //Function C that returns the smallest of three numbers
22    function C(uint256 x, uint256 y, uint256 z) public pure returns (uint256) {
23       return min(x, y, z);
24    }
25
26    // Auxiliary function to calculate the minimum of three numbers
27    function min(uint256 a, uint256 b, uint256 c) private pure returns (uint256) {
28       if (a < b && a < c) {
29          return a;
30       } else if (b < c) {
31          return b;
32       } else {
33          return c;
34       }
35    }
36
37    // Function that calls one of three functions based on the provided selector
38    function CallFunctionWithSelector(bytes4 selector, uint256 x, uint256 y, uint256 z)
39    public pure returns (uint256) {
40       if (selector == this.A.selector) {
41          return A(x, y);
42       } else if (selector == this.B.selector) {
43          (bool result, uint256 intValue) = B(x);
44          // Converti il booleano in un intero: true diventa 1 e false diventa 0
```

```
45          return result ? intValue : 0;
46       } else if (selector == this.C.selector) {
47          return C(x, y, z);
48       } else {
49          revert("Invalid Selector");
50       }
51    }
52 function getSelectors() public pure returns (bytes4, bytes4, bytes4) {
53       return (this.A.selector, this.B.selector, this.C.selector);
54    }
55    }
```

Listing 4.3: Selector

In figure 4.5 the contract is shown.

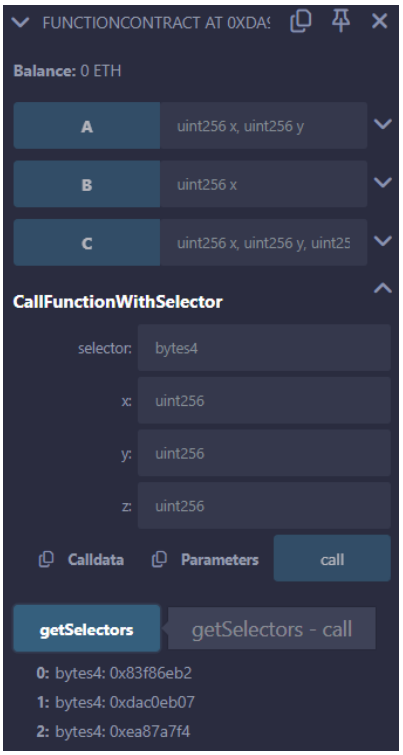

Figure 4.5: Selector but not optimal

However, this contract is not optimal because it requires specifying the CallFunction parameters. A better way is that using Solidity assembly. In this way, the CallFunction takes only a string of bytes as input.

```
1
2  // SPDX-License-Identifier: MIT
3  pragma solidity ^0.8.0;
4
5  contract FunctionContract {
6     // Function A that adds two integers
7     function A(uint256 x, uint256 y) public pure returns (uint256) {
8        return x + y;
9     }
10
11    // Function B that returns true if the number is greater than 50 and returns 1,
12  otherwise it returns false and returns 0
13    function B(uint256 x) public pure returns (bool, uint256) {
14       if (x > 50) {
15          return (true, 1);
16       } else {
```

```
17          return (false, 0);
18       }
19    }
20
21    // Function C that returns the smallest of three numbers
22    function C(uint256 x, uint256 y, uint256 z) public pure returns (uint256) {
23       return min(x, y, z);
24    }
25
26    // Auxiliary function to calculate the minimum of three numbers
27    function min(uint256 a, uint256 b, uint256 c) private pure returns (uint256) {
28       if (a < b && a < c) {
29          return a;
30       } else if (b < c) {
31          return b;
32       } else {
33          return c;
34       }
35    }
36
37    // Function that calls one of three functions based on the supplied data
38    function CallFunction(bytes memory data) public pure returns (uint256) {
39       // Extract the selector from the first 4 bytes of the data
40 bytes4 selector;
41       assembly {
42          selector := mload(add(data, 0x20))
43       }
44
45       // Extract the remaining topics from the data
46       uint256 x;
47       uint256 y;
48       uint256 z;
49
50       assembly {
51          x := mload(add(data, 0x24))
52          y := mload(add(data, 0x44))
53          z := mload(add(data, 0x64))
54       }
55
56       if (selector == this.A.selector) {
57          return A(x, y);
58       } else if (selector == this.B.selector) {
59          (bool result, uint256 intValue) = B(x);
60          // Convert the boolean to an integer: true becomes 1 and false becomes 0
61          return result ? intValue : 0;
62       } else if (selector == this.C.selector) {
63          return C(x, y, z);
64       } else {
65          revert("Invalid␣Selector");
66       }
67    }
68
69    function getSelectors() public pure returns (bytes4, bytes4, bytes4) {
70       return (this.A.selector, this.B.selector, this.C.selector);
71    }
72 }
```

Listing 4.4: Selector With Assembly

In figure 4.6 we can see the optimal selector, the CallFunction can extract the function selector and arguments from the provided byte array, making the contract more versatile.

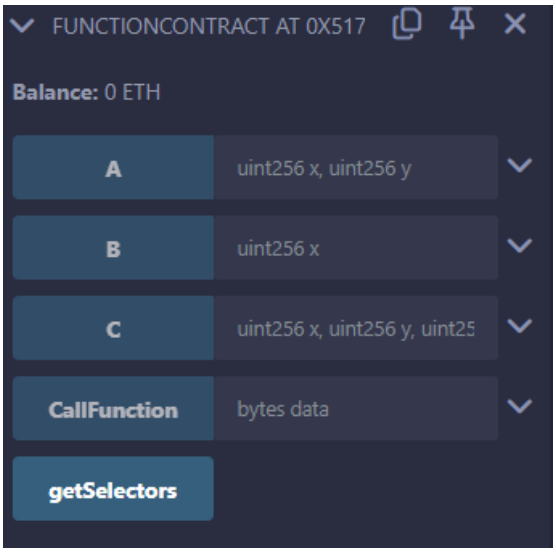

Figure 4.6: Optimal Selector

Now I will explain what the task of the part written in assembly is.

First of all:

```
1
2  bytes4 selector;
3  assembly {
4     selector := mload(add(data, 0x20))
5  }
```

Listing 4.5: First Step Of Assembly

The operation (mload) loads 32 bytes (256 bits) of data from a specified memory location. The data parameter is a dynamic byte array and add(data, 0x20) computes the memory address offset to skip the first 32 bytes, which contain the array's length, to get the actual data.

So This line (selector := mload(add(data, 0x20)):) loads the next 32 bytes (starting from byte 32 to byte 63) from the computed memory address. Since we are interested in the first 4 bytes for the selector, selector is assigned these 4 bytes.

```
1
2  uint256 x;
3  uint256 y;
4  uint256 z;
5
6  assembly {
7     x := mload(add(data, 0x24))
8     y := mload(add(data, 0x44))
9     z := mload(add(data, 0x64))
10 }
```

Listing 4.6: Second Step Of Assembly

Now we extract X,Y and Z from the corrispective data bytes. Extracts x from bytes 36 to 67 (offset by 36 bytes from the start of the data) Extracts y from bytes 68 to 99 (offset by 68 bytes from the start of the data). Extracts z from bytes 100 to 131 (offset by 100 bytes from the start of the data).

In figure 4.7 we can see the result of the execution of the smart contract. The 3 selectors relating to the functions are highlighted in red. Now let's take the first one which is related to a simple sum function, and also insert two coded values 130 and 13, which

would be respectively: 0000000000000000000000000000 0000000000000000000000000000000082 and 0000000000000000000000000000000 0000000000000000000000000d.

In green we can see the result of the operation.

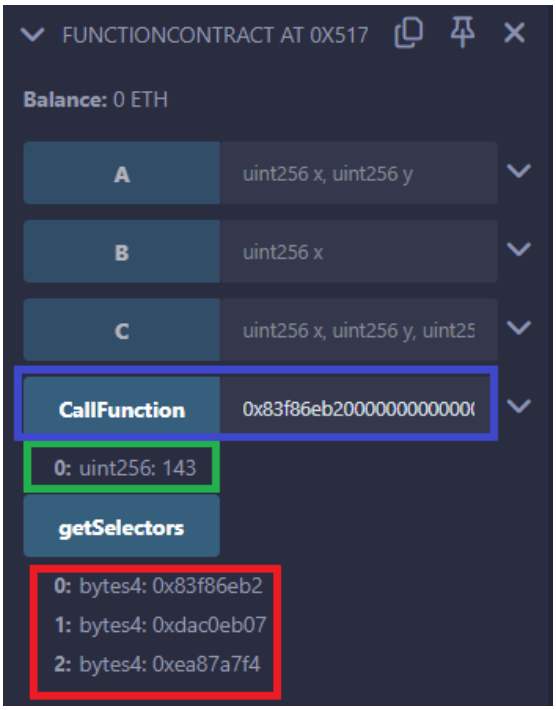

Figure 4.7: Optimal Selector Execution

There is one more way to make this function more efficient, and that is by eliminating all if-else statements, using a static call.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract FunctionContract {
    // Function A that adds two integers
    function A(uint256 x, uint256 y) public pure returns (uint256) {
        return x + y;
    }

    // Function B that returns true if the number is greater than 50 and returns 1,
    otherwise returns false and returns 0
    function B(uint256 x) public pure returns (bool, uint256) {
        if (x > 50) {
            return (true, 1);
        } else {
            return (false, 0);
        }
    }

    // Function C that returns the smallest of three numbers
    function C(uint256 x, uint256 y, uint256 z) public pure returns (uint256) {
        return min(x, y, z);
    }

    // Auxiliary function to calculate the minimum of three numbers
    function min(uint256 a, uint256 b, uint256 c) private pure returns (uint256) {
        if (a < b && a < c) {
            return a;
        } else if (b < c) {
            return b;
        } else {
```

```
32          return c;
33       }
34    }
35
36    // Function that calls one of three functions based on the supplied data
37    function CallFunction(bytes memory data) public view returns (uint256) {
38       (bool success, bytes memory result) = address(this).staticcall(data);
39       require(success, "Call failed");
40
41       return abi.decode(result, (uint256));
42    }
43
44    function getSelectors() public pure returns (bytes4, bytes4, bytes4) {
45       return (this.A.selector, this.B.selector, this.C.selector);
46    }
47 }
```

Listing 4.7: Static call

Using address(this).staticcall(data) on line 38, we can dynamically call a function of the contract itself based on the function selector included in data. staticcall is used for calls that do not change the state.

The staticcall returns two values, lines 39: success, which indicates whether the call was executed successfully, and result, which contains the data returned from the call. Let's verify that success is true. If it isn't, we reverse the transaction with require(success, "Call failed"). We decode the result of the call using abi.decode(result, (uint256)).

It allows you to dynamically call different functions without having to explicitly write code for each call. It reduces the number of if-else conditions in the code, making it more readable and maintainable. It uses less gas than a series of if-else statements to select the function to call.

Obviously the selector can also be used in the Decoder contract like this, line 69.

```
1
2  // SPDX-License-Identifier: MIT
3  pragma solidity ^0.8.0;
4
5  contract Decoder {
6     function decodeStrings(bytes[] memory byteArrays, string[] memory typesArray)
7     public pure returns (string[] memory) {
8        require(byteArrays.length == typesArray.length, "Arrays length mismatch");
9
10       string[] memory decodedValues = new string[](byteArrays.length);
11
12       for (uint i = 0; i < typesArray.length; i++) {
13          if (keccak256(bytes(typesArray[i])) == keccak256(bytes("string"))) {
14             decodedValues[i] = string(byteArrays[i]);
15
16          } else if (keccak256(bytes(typesArray[i])) == keccak256(bytes("int"))) {
17             // Converti l'intero in una stringa
18             int decodedInt = abi.decode(byteArrays[i], (int));
19             decodedValues[i] = intToString(decodedInt);
20          }
21              else if(keccak256(bytes(typesArray[i])) == keccak256(bytes("bool"))) {
22
23                decodedValues[i] = byteToBoolAndString(byteArrays[i]);
24
25          }
26       }
27
28       return decodedValues;
29    }
30
31    function intToString(int _i) internal pure returns (string memory) {
32       if (_i == 0) {
```

```
33          return "0";
34       }
35       uint256 absoluteValue = uint256(_i < 0 ? -_i : _i);
36       uint256 maxLength = 100; //Maximum length of the integer as a string
37       bytes memory reversed = new bytes(maxLength);
38       uint256 i = 0;
39       while (absoluteValue > 0) {
40          uint256 remainder = absoluteValue % 10;
41          absoluteValue /= 10;
42          reversed[i++] = bytes1(uint8(48 + remainder));
43          // Convert the number to an ASCII character
44       }
45       bytes memory s = new bytes(i); // Actual size of the string
46       for (uint256 j = 0; j < i; j++) {
47          s[j] = reversed[i - j - 1]; // Reverse the string
48       }
49       return string(s);
50    }
51
52
53     function byteToBoolAndString(bytes memory data) internal pure returns (string memory) {
54       require(data.length == 1, "Invalid input length");
55
56       if (data[0] == 0x00) {
57          return "false";
58       } else if (data[0] == 0x01) {
59          return "true";
60       } else {
61          revert("Invalid boolean value");
62       }
63    }
64
65    function getSelectors() public pure returns (bytes4) {
66       return (this.decodeStrings.selector);
67    }
68
69    function callFunction(bytes4 selector, bytes[] memory byteArrays, string[] memory
70    typesArray) public pure returns (string[] memory) {
71
72       if (selector == this.decodeStrings.selector) {
73          return decodeStrings(byteArrays, typesArray);
74
75       }
76        else {
77
78          revert("Invalid selector");
79       }
80    }
81 }
```

Listing 4.8: Contract Decoder with selector

4.1.3 Contract Deployer

As a final solution to LSP issues on smart contracts, I created a function capable of deploying another contract at runtime using its bytecode This method enhances modularity and flexibility in contract design, ensuring that derived contracts can replace base contracts without altering the desirable properties of the program.

```
1
2    // SPDX-License-Identifier: GPL-3.0
3 pragma solidity >=0.4.16 <0.9.0;
4
5 contract ContractDeployer {
6    event ContractDeployed(address indexed deployedAddress);
7
8    function deployContract(bytes memory bytecode) public returns (address deployedAddress) {
```

```
9         assembly {
10           deployedAddress := create(0, add(bytecode, 0x20), mload(bytecode))
11           if iszero(extcodesize(deployedAddress)) {
12              revert(0, 0)
13           }
14        }
15        emit ContractDeployed(deployedAddress);
16     }
17  }
```

Listing 4.9: Contract Deployer

Here too to create this function, I had to use the solidity assembly.

```
1
2  deployedAddress := create(0, add(bytecode, 0x20), mload(bytecode))
```

Listing 4.10: Contract Deployer Assembly

This line of code deploys a new contract with the given bytecode. The first '0' means the amount of Ether (in wei) to send to the new contract. In this case, it's set to 0, meaning no Ether is sent. add(bytecode, 0x20) is the memory address where the actual bytecode starts. Bytecode is a dynamic array where the first 32 bytes store the length of the array and i need to skips the first 32 bytes to get the start address of the bytecode. In final (mload) Loads the length of the bytecode from the first 32 bytes of the bytecode array. (deployedAddress) Is the address of the newly deployed contract.

This address is returned by the create operation.

```
1
2  if iszero(extcodesize(deployedAddress)) {
3     revert(0, 0)
4  }
```

Listing 4.11: Contract Deployer Assembly

This part of code returns the size of the code at the address deployedAddress and checks if the size of the code at the deployed address is zero.

If the deployed contract's code size is zero, the deployment is considered unsuccessful, and the transaction is reverted. This ensures that the function only returns successfully if a valid contract is deployed.

4.1.4 How to use Contract Deployer

To thoroughly test the deployContract function in the ContractDeployer, I created three different contracts. These contracts demonstrate various use cases, including a contract with a constructor, a simple contract without a constructor, and a contract that provides a utility function for ABI encoding.

SimpleAdder Contract

The SimpleAdder contract provides a simple function to add two uint256 numbers and return the result. In figure 4.8 is show the SimpleAdder contract.

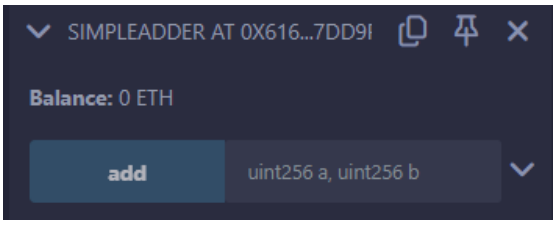

Figure 4.8: SimpleAdder Contract

This contract has no constructor.

```
pragma solidity ^0.8.0;

contract SimpleAdder {
    function add(uint256 a, uint256 b) public pure returns (uint256) {
        return a + b;
    }
}
```

Listing 4.12: SimpleAdder Contract

To get the bytecode of the SimpleAdder contract, using Remix IDE, you need to go to the solidity compiler and click on the bytecode item. In figure 4.9 is show the solidity compiler and in red where to get the bytecode of the contract.

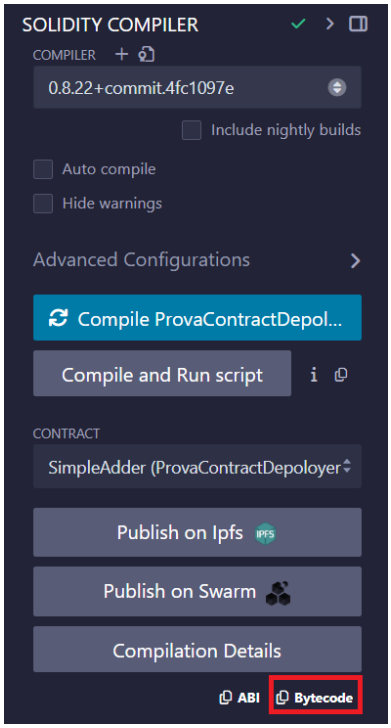

Figure 4.9: Soldity Commpiler

Now just insert the bytecode inside the ContractDeployer contract and once execution has started, it will deploy a new SimpleAdder contract.

```
608060405234801561000f575f80fd5b506101a58061001d5f395ff3fe608060405234801561000f
575f80fd5b506004361061029575f3560e01c8063771602f71461002d575b5f80fd5b6100476004
803603810190610042919061009a565b61005d565b604051610054919061010f6565b604051809103
90f35b5f818361006a919061013c565b905092915050565b5f80fd5b5f819050919050565b610088
81610076565b8114610092575f80fd5b50565b5f813590506100a38161007f565b92915050565b5f
```

```
6  806040838503121561006bf576100be610072565b5b5f6100cc85828601610095565b925050602061
7  00dd85828601610095565b91505092509290505565b6100f081610076565b82525050565b5f602082
8  0190506101095f8301846100e7565b92915050565b7f4e487b7100000000000000000000000000000
9  0000000000000000000000000000005f526011600452602045ffd5b5f610146826100765656b91506101
10 5183610076565b92508282019050808211156101695761016861010f565b5b9291505056fea26469
11 70667358221220b07b107011bd4b6ac7bfec821146ca401d556590f5a3c1d3779f23f10f011cb964
12 736f6c63430008160033
```

Listing 4.13: SimpleAdder Contract

This is the bytecode of the SimpleAdder contract. Now insert it, into the Contract-Deployer (always remember to put 0x in front of the bytecode) and execute it. We can see it in figure 4.10.

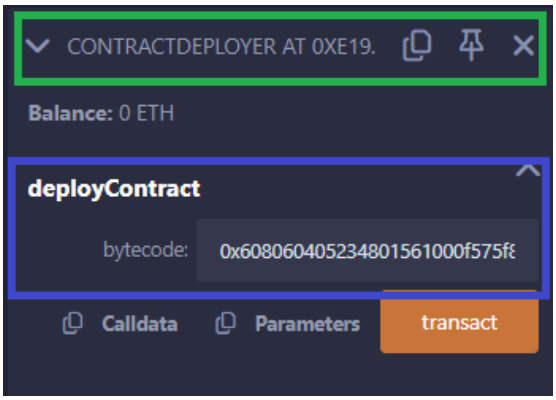

Figure 4.10: ContractDeployer

The result of the execution will be shown in figure 4.11.

Figure 4.11: Result of the Execution

As you can see, the blue arrow indicates the inserted bytecode of the SimpleAdder contract, the green line indicates the address of the ContractDeployer contract, and the two red lines indicate the address of the new SimpleAdder contract.

Now all we have to do is load the new address. In figure 4.12 we can see the load of new SimpleAdder contract.

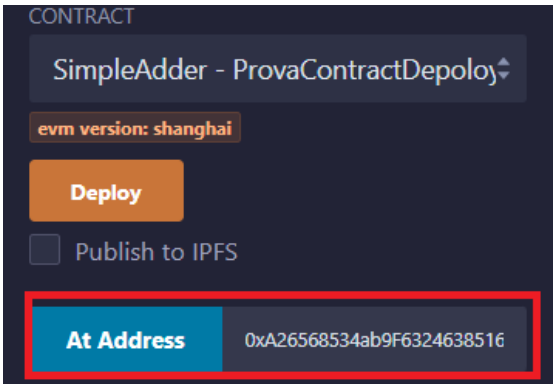

Figure 4.12: Load of new SimpleAdder contract

Then we will have the new contract in the Deployed/Unpinned Contracts section of Remix IDE. In figure 4.13 is shown the new SimpleAdder contract.

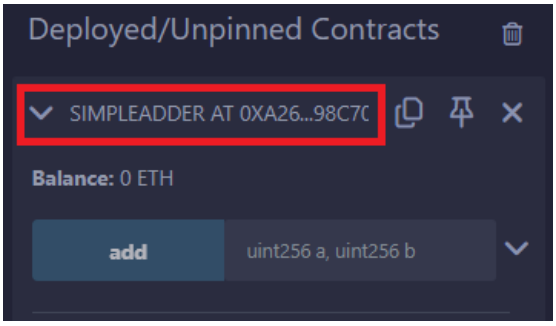

Figure 4.13: New SimpleAdder Contract

As you can see, the address corresponds to the contract created before.

SimpleStorage Contract

The SimpleStorage contract includes a constructor that initializes a uint256 variable, along with functions to set and get the value of this variable.

```
1
2  // SPDX-License-Identifier: GPL-3.0
3  pragma solidity >=0.4.16 <0.9.0;
4
5  contract SimpleStorage {
6
7
8     uint256 public data;
9
10    constructor(uint256 _data) {
11       data = _data;
12    }
13
14    function setData(uint256 _data) public {
15       data = _data;
16    }
17
18    function getData() public view returns (uint256) {
19       return data;
20    }
21 }
```

Listing 4.14: SimpleStorage Contract

The process for using the ContractDeployer contract is the same. The only difference is that since there is a constructor, in addition to the bytecode of the contract, it is necessary to insert the value to be inserted in the constructor in hexadecimal coded in the ABI version.

To simplify the conversion, I created a new ABIEncoder contract, which given a decimal number, returns its format in hexadecimal with ABI encoding.

In figure 4.14 is shown the ABI Encoder contract.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;

contract ABICoder {
   //This function takes a uint256 and returns its ABI encoding
   function encodeUint256(uint256 value) public pure returns (bytes memory) {
      // We use the abi encode to get the ABI representation of the value
      return abi.encode(value);
   }
}
```

Listing 4.15: ABIEncoder Contract

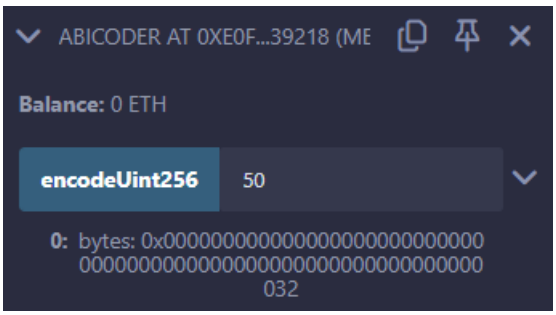

Figure 4.14: ABIEncoder Contract

As you can see, I entered the number 50 and it gave me the corresponding coded value.

This is the bytecode of SimpleStorage:

```
608060405234801561000f575f80fd5b5060405161021d38038061021d8339818101604052810190
6100319190610074565b805f81905550506100 9f565b5f80fd5b5f81905091905056 5b6100538161
0041565b811461005d575f80fd5b50565b5f8151905061006e8161004a565b92915050565b5f6020
8284031215610089576100886100 3d565b5b5f6100968482850161006056 5b91505092915050565b
610171806100ac5f395ff3fe608060405234801561000f575f80fd5b506004361061003f575f3560
e01c80633bc5de3014610043578063 5780635b4b73a91461006157806373d4a13a1461007d575b5f80fd5b
61004b61009b565b6040516100589190610 0c9565b60405180910390f35b61007b60048036038101
9061007691906100 10565b6100a3565b005b6100856100ac565b6040516100929190610 0c9565b60
405180910390f35b5f8054905090565b805f8190555050565b5f5481565b5f81905091905056 5b61
00c3816100b1565b82525050565b5f602082019050610 0dc5f8301846100ba565b92915050565b5f
80fd5b6100ef816100b1565b81146100f9575f80fd5b50565b5f8135905061010a816100e6565b92
915050565b5f602082840312156101255761012461 00e2565b5b5f6101328482850161 00fc565b91
50509291505056fea2646970667358221220 3a337eb614a65cdc6fc657e55b82705860bf19479464
00cb8c8866919b967bf464736f6c63430008160033
```

Listing 4.16: Bytecode SimpleStorage

So after that we need to add:

```
0000000000000000000000000000000000000000000000000000000000000032
```

Listing 4.17: Coded Value

In figure 4.15 is shown the result of the execution. As before, the blue arrow indicates the bytecode of the SimpleStorage contract, while the yellow line represents the value 50 encoded in ABI.

The red line confirms the creation of the contract by giving its address.

Figure 4.15: Result of ContractDeployer with constructor

After uploading the corresponding address, we will obtain the deployment of the new contract. In figure 4.16 it shows the new contract created. We can verify that the address of the new SimpleStorage contract corresponds to the one we just created a little while ago, and also if we use the GetData function, we will see that it will return the value of 50.

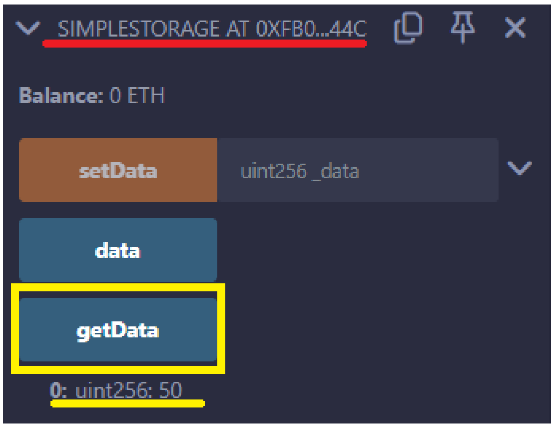

Figure 4.16: New SimpleStorageContract

5. Conclusion

In conclusion, this elaborate provides a practical solution for deploying smart contracts at runtime, addressing the issues associated with Loosely Specified Processes (LSP) in smart contract development. By leveraging the principles of LSP, the proposed framework enhances the adaptability and resilience of smart contracts, allowing them to dynamically respond to changing conditions and requirements. This work lays the foundation for more advanced and dynamic applications of blockchain technology, where contracts can be deployed and interacted with in a flexible and automated manner. Such adaptability is crucial in various domains where processes cannot be fully predefined and must be capable of real-time adjustment.

The framework developed in this elaborate not only addresses current limitations but also opens up new avenues for innovation in the blockchain space. By enabling dynamic customization and behavior selection within smart contracts, the framework supports a wide range of applications across different industries. From supply chain management and healthcare to decentralized finance (DeFi) and beyond, this flexible approach can significantly enhance operational efficiency and effectiveness.

Future work could focus on optimizing the deployment process further, making it even more efficient and user-friendly. This might involve refining the underlying algorithms and improving the integration with existing blockchain platforms. Additionally, exploring additional use cases and applications for this dynamic deployment method could provide valuable insights and drive further innovation. For instance, investigating how this approach can be applied to emerging fields such as IoT, AI, and smart cities could reveal new opportunities and benefits.

Moreover, security considerations should be continually addressed to ensure that the deployment mechanism remains robust against potential vulnerabilities. This involves conducting thorough security audits, implementing best practices for smart contract development, and staying updated with the latest advancements in blockchain security. Ensuring the integrity and reliability of the deployment process is paramount, as any weaknesses could be exploited by malicious actors.

Another important area for future research is the development of standardized protocols and frameworks that can facilitate the widespread adoption of dynamic smart contract deployment. By creating a unified approach, it would be easier for developers to implement and utilize these methods across different platforms and industries. Collaboration with industry stakeholders and academic institutions could help drive these standardization efforts, promoting a more cohesive and interoperable blockchain ecosystem.

Bibliography

[Aut] The Solidity Authors. *Solidity*. URL: https://docs.soliditylang.org/en/v0.8.21/.

[Com] Remix Community. *Remix*. URL: https://remix-ide.readthedocs.io/en/latest/.

[cri] criptovaluta.it. *Ethereum: Guida definitiva su ETH Coin*. URL: https://www.criptovaluta.it/ethereum#:~:text=Ethereum%3A%20smart%20contract%20e%20creazione,questo%20network%20permettono%20di%20implementare..

[di)19] Raffaele Battaglini Marco Tullio Giordano (a cura di). *Blockchain E Smart Contract*. Milano: Giuffrè, 2019.

[EG] T. Ricci E. Gambula. *Merge di Ethereum, cosa cambia per la blockchain: rischi e opportunità dopo la svolta*. URL: https://www.agendadigitale.eu/documenti/merge-di-ethereum-cosa-cambia-per-la-blockchain-rischi-e-opportunita-dopo-la-svolta/.

[Etha] Ethereum.org. *Ethereum*. URL: https://web.archive.org/web/20140208030136/http://www.ethereum.org/.

[Ethb] Ethereum.org. *Gas e commissioni*. URL: https://ethereum.org/it/developers/docs/gas/.

[MR12] Barbara Weber Manfred Reichert. *Enabling Flexibility in Process-Aware Information Systems*. London, 2012.