Jeong-Jin Kang
Edward J. Rothwell
Yang Hao
Jongwook Jang

Advanced and Applied Convergence Letters AACL 17

Advanced and Applied Convergence & Adavanced Culture Technology

9th International Symposium, ISAAC 2021 in Conjunction with ICACT 2021, ICKAI 2021

November 11 - 13, 2021, SETEC, Korea
Revised Selected Papers

IIBC The Institute of Internet, Broadcasting and Communication
IPACT The International Promotion Agency of Culture Technology
지식의 숲 FOREST OF KNOWLEDGE

Volume Editors:

Jeong-Jin Kang
Dong Seoul University, 423 Bokjeong-Dong, Sujeong-Gu, Seongnam, Gyunggi, 13117 Korea
E-mail: jjkang@du.ac.kr

Edward J. Rothwell
Michigan State University, 2120 Engineering Building East Lansing, MI 48824-1226, USA
E-mail: rothwell@egr.msu.edu

Yang Hao
Queen Mary University of London, Mile End Road, London E1 4NS, UK
y.hao@qmul.ac.uk

Jongwook Jang
Dongeui University, 176, Eomgwang-ro, Busanjin-gu, Busan, Korea
jwjang@deu.ac.kr

ISSN: 2288-6060

CONTENTS

Design Validation Practices on Design Extraction of Solidity's Smart Contract Code based on Reverse Engineering

[1]Chansol Park, [2]Jungeun Park, [3]Seohee Hong, [4]Jiyun Kim, [5]Janghwan Kim, [6]Bokyung Park, [7]Soyoung Moon, [8]Hyun Seung Son, [9]R. Young Chul Kim

[1,2,3,4,5,7,9]*Dept. of Software and Communication Engineering, Hongik University, South Korea*
[6]*Dept. of Computer Education, Chinju National University of Education, South Korea*
[8]*Dept. of Computer Engineering, Mokpo National University, South Korea*
[1]chansol53, [2]je2998, [3]tjgml708, [4]b789014}@mail.hongik.ac.kr,
{[5]janghwan, [7]msy, [9]bob}@selab@hongik.ac.kr,
[6]parkse@cue.ac.kr, [8]hson@mokpo.ac.kr

Abstract

As interest in cryptocurrencies increases, the blockchain technology market is rapidly expanding. In the meantime, as we enter the second-generation blockchain era, our interest and research on smart contracts and Decentralized applications are progressing together. In 2020, with the enactment of the Software Promotion Act in Korea, we must separate and order the requirements, design, and implementation for a project. At this moment, we need to make technical research on whether development has been carried out according to the original requirements-based design is required or not. To solve this problem, we propose an extraction method of requirement-based smart contract design restoration based on reverse engineering. Through this, it is expected to validate the reliability of the second-generation blockchain software will increase and help fast and efficient development.

Keywords: Solidity Visualization, Reverse Engineering, Design Restoration, Block Chain, Ethereum

1. Introduction

Recently, the blockchain market is rapidly expanding due to the crypto currency. With the advent of the Ethereum blockchain environment, the blockchain Paradigm has shifted to a second-generation blockchain that creates and uses Decentralized Applications(dApp), unlike other blockchains that focus on functions as a cryptocurrency [1]. The dApp is a digital application that automatically executes pre-written code when the conditions of a smart contract are satisfied. Solidity is used as a representative programming language to implement smart contract [2]. Solidity is a Turing-complete programming language that allows you to write smart contracts in the Ethereum environment [3].

In 2020, the Software Promotion Act was enacted for separate ordering of design and implementation [4]. Therefore, to reduce the difference between the design and implementation analyzed from the requirements, it is necessary to verify whether the implemented software is developed according to the requirements-based design. In academia, research has been conducted to apply various verification techniques that reduce the difference between design and implementation in existing software. As such research and attempts are continuing in the existing object-oriented software, but research is hardly conducted about the blockchain-based dApp.

We propose method about an extraction method of requirement-based smart contract design restoration based on reverse engineering in the Ethereum environment. Through this, it is expected that errors that may occur during separate orders for between software development and design can be reduced. In Chapter 2, as related studies, research on reverse engineering-based object-oriented software design restoration and smart contracts with dApp in the Ethereum environment are discussed. Chapter 3 introduces the proposed method that is how to extract the design from the source code through reverse engineering for the smart contract that runs in the Ethereum environment. And then, we will remark conclusion with future research in chapter 4.

2. Related works

2.1 Extracting Use Case Design Mechanisms on Reverse Engineering

Extracting software designs through reverse engineering has been continuously studied in the field of software engineering. This study proposes design extraction according to the relationship between classes in object diagram and sequence diagram through programming for target code analysis[5]. In addition, this study focused on design restoration from an object-oriented perspective and studied coupling graphs, class diagrams, and sequence diagrams together.

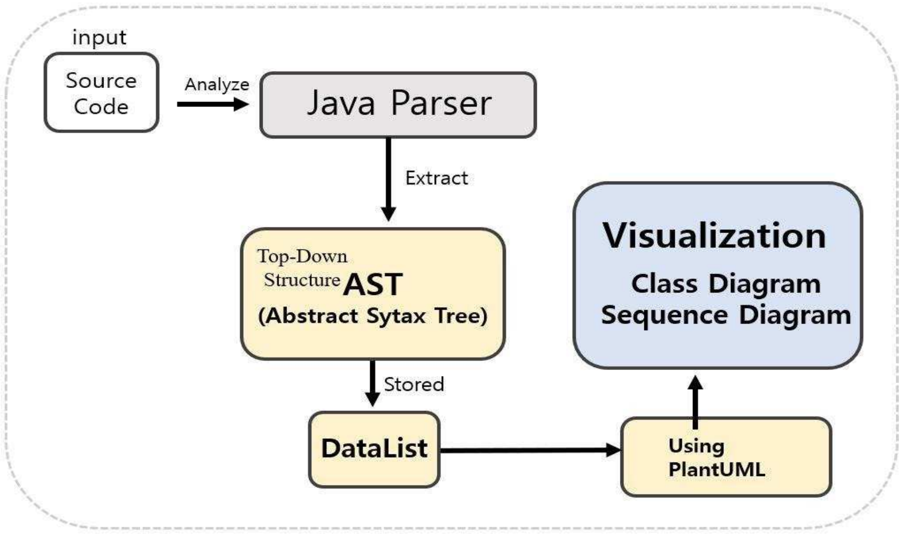

Figure 1. Schematic Diagram of Object-Oriented Software Visualization Tool Chain

Figure 1 is a schematic diagram of the software visualization toolchain. Existing research uses a tool called Source Navigator to analyze the source code and uses SQLite to store the data in the DB. This study focused on software that are written in Java programming language and studied design extraction. Unlike the analysis methods of existing studies, Java parser, a Java static analysis tool, is used instead of Source Navigator, an open-source tool that requires query statement optimization to extract essential information. By using the Java parser, the AST of the source code is extracted, and the extracted data is stored in an object, not in the DB, so that fast and efficient analysis is possible. In addition, instead of GraphViz as a visualization tool, PlantUML, an open-source extensible tool based on GraphViz, was used to make visualization more convenient.

2.2 Extracting Use Case Design Mechanisms on Reverse Engineering [6]

The smart contract is a technology that has not been actively used until the release of the Ethereum platform due to security vulnerabilities and technical limitations [7]. However, it became the starting point of implementing contract automation by overcoming the limitations through Ethereum. Typically, smart contracts are not only used for cryptocurrency transactions, but are also actively applied to fields such as trade, finance, and real estate [8].

A decentralized application (dApp) generally consists of a smart contract recorded on a distributed ledger and a program executed through agreement of the contract [9]. Figure 2 describes a simplified diagram of the dApp operation process. As shown in Figure 2, the user manipulates the dApp through the user interface.

This calls the necessary functions through the remote procedure call(RPC) method from the smart contract recorded in the blockchain network [10]. The value of state variable that is changed in this process is recorded in a new block through a transaction.

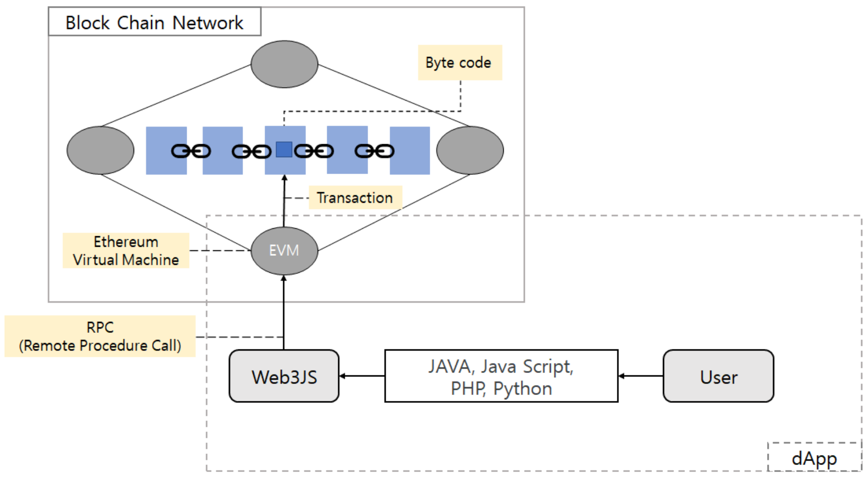

Figure 2. Schematic Diagram of dApp Operation Process

3. Design Extraction of Smart Contract Code based on Reverse Engineering

This describes a blockchain-based software visualization toolchain in the Ethereum environment that can analyze solidity codes into class diagrams, sequence diagrams, and use case diagrams. Figure 3 is a structural diagram regards the blockchain-based software visualization process that we mentioned above. First, the .sol file written in the Solidity language is entered into the tool-chain as an input. Then, the AST for the source code is extracted through the Solidity Static Binaries Compiler which is provided by official solidity web repository. The extracted AST is saved in .json format, and the AST of the source code is stored in an object in memory using the Solidity AST.JSON Parser which we developed for this tool-chain. Finally, the toolchain retrieves the necessary information for each diagram from the stored data and then, it draws each diagram by executing PlantUML, a visualization tool. In this paper, we draw class diagrams, sequence diagrams, and use case diagrams to restore the high-level design.

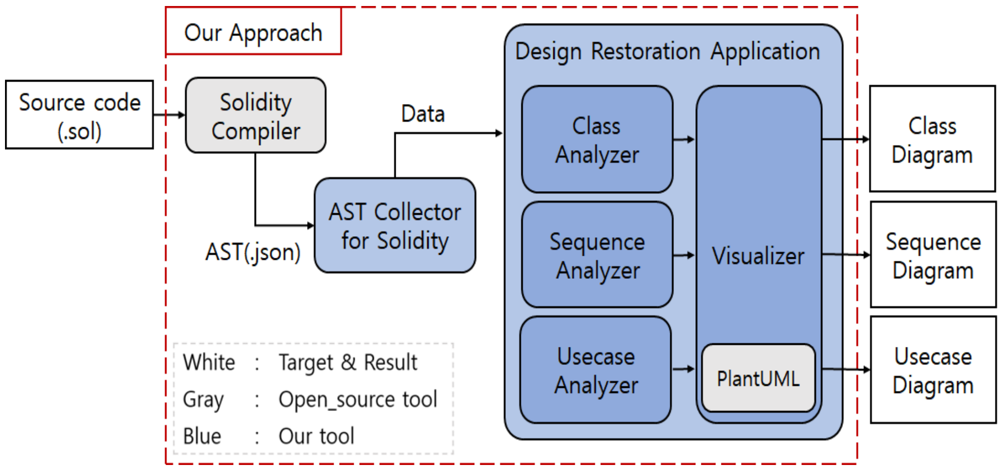

Figure 3. High-level Design Extraction Process for Blockchain based Application

3.1 AST Collector for Solidity

AST Collector for Solidity is a tool that stores AST information extracted with Solidity Complier as objects in memory. AST information consists of nodes such as SourceUnit, PragmaDirective, ContractDefinition, and VariableDeclaration. When AST Collector receives .sol files as an input, a SourceUnit node is created for each .sol file. The PragmaDirective node is a child node of the SourceUnit node and has compiler version that extract AST. The ContractDefinition node contains all information about the Contract and is created as Contracts are declared. Contract nodes include nodes corresponding to state variables, functions, structures, modifiers, events, and enum structures declared in the contract. For example, if a state variable A is declared in a contract, a VariableDeclaration node is created as a branch of the Contract node. Then, information such as name, data type, and declaration method are stored for the state variable A.

3.2 High-level Design Restoring Application

High-level Design Restoration Application processes data according to each diagram to draw each UML diagram from the AST information stored in the object. After that, it draws the UML diagram through the PlantUML which is a visualizer. The following is a summary of the extraction methods for each diagram.

3.2.1 Class Diagram for Solidity

In blockchain software written by Solidity language, the structure called 'Contract' is very similar to 'Class' in object-oriented design. However, we will use word 'Contract diagram' for the class diagram in this environment because it is used as the name of the contract instead of the class. The contract diagram includes the contract components (state variable, struct, enum, function, constructor, event, modifier, Visibility, State Mutability, etc.), which are data structures of software in the Ethereum environment. A state variable is a variable declared in a contract but is not declared in a function. A struct is a integrated data type that consists of variables that are different data types. An enum is a list of predefined constant values. A function is a set of program code designed to perform a special task. A constructor is a function that is executed when the contract is created. Only one constructor can be used per contract. An event is an action or event that can be detected and handled by a program. A modifier is a structure associated with a function that always executes before the target function to change the function's behavior. Visibility limits the access scope of functions like access modifiers in Java or C++. A State Mutability is a type of qualifier that affects the operation of a function and is used to clarify the role of a function. Therefore, it is possible to intuitively grasp the relationship and structure between contracts through the contract diagram. Figure 3 is an example contract

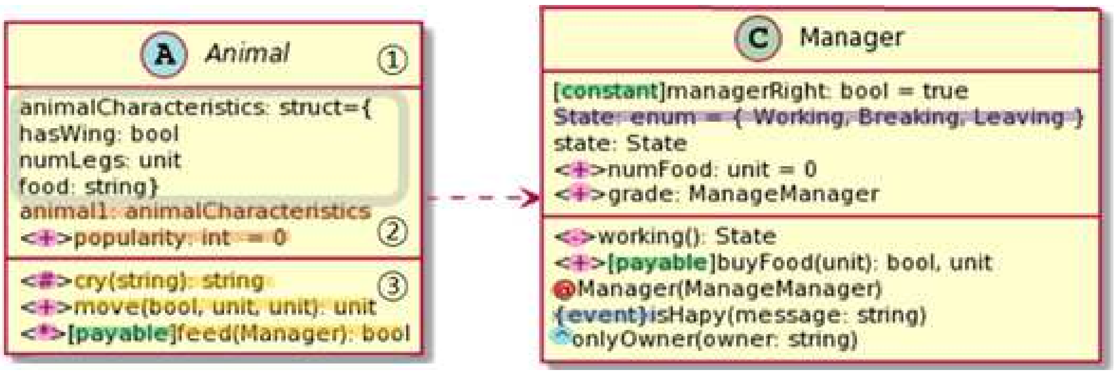
A Animal
①
animalCharacteristics: struct={
hasWing: bool
numLegs: unit
food: string}
animal1: animalCharacteristics
<+>popularity: int = 0
②
<#>cry(string): string
③
<+>move(bool, unit, unit): unit
<*>[payable]feed(Manager): bool
C Manager
[constant]managerRight: bool = true
State: enum = { Working, Breaking, Leaving }
state: State
<+>numFood: unit = 0
<+>grade: ManageManager
<->working(): State
<+>[payable]buyFood(unit): bool, unit
Manager(ManageManager)
{event}isHapy(message: string)
^onlyOwner(owner: string)

Figure 4. Example of Contract Diagram

diagram.

The contract diagram is divided into three main parts. Area ① is displayed as general, abstract, and interface according to the name of the contract and the type of the contract. In area ②, state variables, struct, and enumeration types are displayed. Area ③ displays functions, constructors, events, and modifiers. State variables are indicated in orange as ′variable name: data type′. Struct are colored in gray and indicated as ‘struct name: struct = {content}’. Enumeration types are expressed in purple as ‘enum type name: enum = {content}’. Functions are indicated in yellow and marked as ‘function name (parameter: data type): return type’. Constructors are indicated by '^' in light blue. Events are marked with ‘{event}’ in blue. Modifiers are indicated by using ‘@’ in red. Visibility is indicated in pink with + for public, # for internal, - for private, and * for external. State Mutability is indicated by ‘[]’ sign in green, and there are four types such as, constant, view, pure, and payable. The representation of the relationship between each contract is the same as in the existing class diagram for object-oriented structure. Table 1 shows a summary of notation that express unique elements to Solidity that do not correspond to the existing class diagram.

Table 1. New Expressions to Contract Diagram

Category	Notation		Category	Notation	
State Mutability	constant	[constant]	Visibility	public	+
	view	[view]		internal	#
	pure	[pure]		private	-
	payable	[payable]		external	*
Struct	Struct name: struct = {…}		Constructor	^	
Enum	Enum name: enum = {…}		Event	(event)	
Modifier	@			…	

3.2.2 Sequence Diagram for Solidity

Sequence diagrams show the interactions between contracts, allowing you to intuitively understand the flow of messages between contracts. After we find the connection between functions by analyzing the call relationship between functions, the list of connected functions is called case. Then, the flow of each case is expressed as a sequence diagram. Figure 4 is an example of a blockchain-based sequence diagram. The flow is understood by expressing the name of the function that calls the message between the contract lifelines.

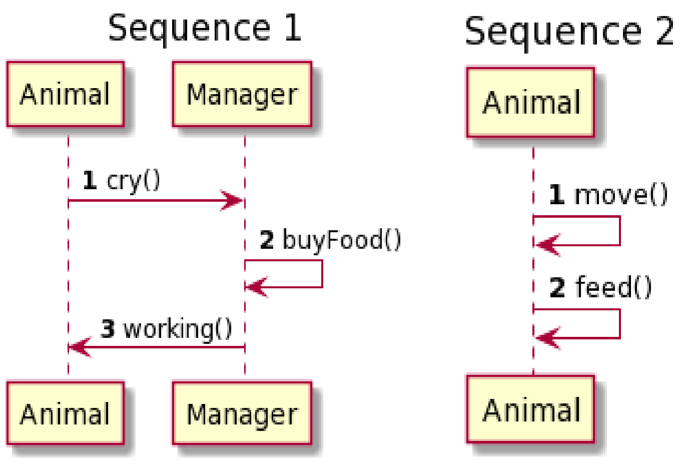

Figure 5. Example of Sequence Diagram for Solidity

3.2.3 Use Case Diagram for Solidity

Use case diagrams represent the interactions between users and systems, making software intuitively understandable. We try to help to compare the design and development results by analyzing the written code as a use case diagram. Figure 5 is an example of a blockchain-based use case diagram. Actors are represented as contractors for contract agreement, and cases are represented by functions analyzed through sequence diagrams.

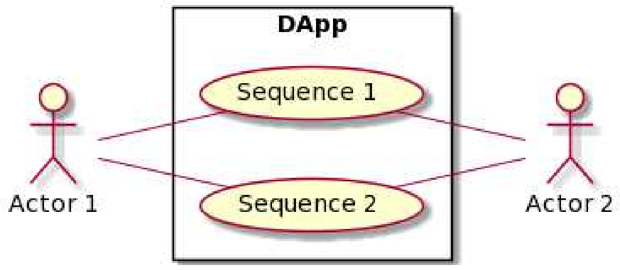

Figure 6. Example of Use case Diagram for Solidity

4. Conclusion

Our works is about a design extraction based on reverse engineering of smart contract code that is written in Solidity language. We restore the high-level design from the source code that is written in Solidity, a contract-oriented language, through an object-oriented approach using reverse engineering. This language is like an object-oriented language, but research on this topic is still insufficient. In the future, we plan to restore the design and requirements by tracing from the source code to the requirements based on the design and study the Solidity Static Analyzer.

5. Acknowledgement

The research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2021R1I1A305040711) and the BK21 FOUR (Fostering Outstanding Universities for Research) funded by the Ministry of Education (MOE, Korea) (No.2021R1I1A305040711).

6. References

[1] K. Kim, "The Impact of Blockchain Technology on the Music Industry." *International journal of advanced smart convergence* Aug. 1st (2019) pp: 196-203.B. Sklar, Digital Communications, Prentice Hall, pp. 187, 1998.
[2] D.Lee, W. Ji, H. Kim. "Survey on trends in smart contract-compatible programming languages." *Korean Information Science Association Academic Presentation Paper Collection* (2018) pp: 1490-1492.
[3] Jansen, Marc, et al. "Do smart contract languages need to be Turing complete?." *International Congress on Blockchain and Applications*. Springer, Cham, 2019.
[4] ZDNET Korea, Enforcement of the revised SW Promotion Act ··· "Open a new SW culture", https://zdnet.co.kr/view/?no=20201210061826
[5] H.Kwon, and R. Kim. "Extracting Use Case Design Mechanisms via Programming based on Reverse Engineering." *International Journal of Applied Engineering Research* 10.90: 2015.
[6] S.Jung, et al. "Automatic UML Design Extraction with Software Visualization based on Reverse Engineering." *International Journal of Advanced Smart Convergence* 10.3 (2021) pp: 89-96.
[7] C. Roh, I. Lee. "Smart Contract-based Electronic Voting System in Blockchain." *Proceedings of the Korea Information Processing Society Conference. Korea Information Processing Society* 26.1 (2019) pp: 190-191.
[8] H.T. Choi. "The legal issues and tasks of smart contracts." *Law and policy research* 21.1 (2021): 381-408.
[9] WANG, Shuai, et al. "An overview of smart contract: architecture, applications, and future trends." *2018 IEEE Intelligent Vehicles Symposium (IV). IEEE*, 2018. p. 108-113.
[10] Svenson, C. "Blockchain: Using cryptocurrency with java." *Java Magazine*, January/February (2017): 36-46.