Journal of the Korea Academia-Industrial cooperation Society
Vol. 19, No. 6 pp. 110-119, 2018

https://doi.org/10.5762/KAIS.2018.19.6.110
ISSN 1975-4701 / eISSN 2288-4688

블록체인 기반의 스마트 컨트랙트 정적/동적 설계 기법

김철진
인하공업전문대학 컴퓨터시스템과

A Static and Dynamic Design Technique of Smart Contract based on Block Chain

Chul-Jin Kim
Dept. of Computer Systems and Engineering, Inha Technical College

요 약 블록체인 기술은 무결성과 부인 거부 등의 뛰어난 보안성으로 계약 분야(매매 계약, 부동산 계약)에 활용도가 높게 평가되고 있다. 블록체인에서 이러한 계약 서비스는 스마트 컨트랙트라는 기술을 활용하여 개발 가능하며 여러 블록체인 플랫폼들이 스마트 컨트랙트를 개발하기 위한 프로그래밍 언어를 제공한다. 대표적인 블록체인 플랫폼인 비트코인과 이더리움은 비트코인 스크립트와 솔리디티 언어를 제공한다. 이러한 프로그래밍 언어를 이용하여 동적으로 처리될 수 있는 디지털 계약인 스마트 컨트랙트를 개발할 수 있다. 그러나, 다양한 계약 분야에서 스마트 컨트랙트의 개발이 진행되고 있으나 블록체인 기반의 설계를 위한 연구가 미흡한 상황이다. 이에 본 연구에서는 이더리움 기반으로 스마트 컨트랙트를 설계하기 위한 메타모델 및 UML 기반의 정적/동적 설계 기법을 제안한다. 정적설계에서는 스마트 컨트랙트의 속성과 기능을 설계하기 위한 기법을 제안하며, 그리고 컨트랙트들 간의 구조를 설계하기 위한 기법을 제안한다. 동적설계에서는 블록체인 내의 스마트 컨트랙트, 계정, 블록들 간에 배포, 기능 호출, 동기화를 설계하기 위한 기법을 제안한다. 실험은 부동산 계약 사례를 통해 정적/동적 설계 기법을 적용하여 설계 기법의 타당성을 검증한다.

Abstract Blockchain technology has been highly evaluated for its contracts (contracts for sale, real estate contracts) because of its excellent security, including integrity and non-repudiation. In a blockchain, these contract services can be developed using a technology called a smart contract, and several blockchain platforms provide a programming language for developing smart contracts. Bitcoin and Ethereum, typical blockchain platforms, provide the Bitcoin Scripts and Solidity languages. Using these programming languages, we can develop the smart contract, a digital contract that can be processed dynamically. Smart contracts are being developed in a variety of areas, but studies of designs based on a blockchain are insufficient. In this paper, we propose a meta-model and a static/dynamic design method based on Unified Modeling Language (UML) for smart contracts based on Ethereum. We propose a method for static design attributes and functions of smart contracts, and propose a technique for designing structures among contracts. Dynamic design proposes a technique for designing deployment, function calls, and synchronization among smart contracts, accounts, and blocks within a blockchain. Experiments verify the validity of the design method by applying the static/dynamic design method through real estate contracts.

Keywords : Block Chain, Smart Contrct, Solidity, Static Design, Dynamic Design

1. 서론

블록체인(Block Chain)은 금융 거래뿐만 아니라 유통, 계약, SNS 등 다양한 비금융 분야에 활용 가능하다. 특히 블록체인 플랫폼은 계약 관련해서 제3자의 개입 없이 자동으로 처리될 수 있는 스마트 컨트랙트(Smart

*Corresponding Author : Chul-Jin Kim(Inha Technical College)
Tel : +82-32-870-2338 Email : cjkim@inhatc.ac.kr
Received March 7, 2018 Revised (1st April 3, 2018, 2nd April 26, 2018, 3rd May 25, 2018)
Accepted June 1, 2018 Published June 30, 2018

Contract)[1] 기술을 제공한다. 블록체인 기반의 스마트 컨트랙트는 개발 가능하며, 비트코인(Bitcoin)이나 이더리움(Ethereum) 등과 같은 블록체인 플랫폼들은 이러한 스마트 컨트랙트를 개발하기 위한 스크립트 언어를 제공한다. 그러나, 이러한 스마트 컨트랙트 기술을 이용하여 다양한 분야에서 개발하는 사례가 증가하고 있으나 체계적인 설계 기법에 대한 연구는 미흡한 상황이다. 이에 본 연구에서는 UML의 확장 메커니즘을 이용하여 블록체인 플랫폼의 스마트 컨트랙트를 설계하기 위한 메타모델과 정적/동적 설계 기법을 제안한다.

1장에서는 스마트 컨트랙트의 상황을 설명하고, 2장에서는 관련 연구로서 스마트 컨트랙트 기술과 스마트 컨트랙트를 개발하기 위한 언어 중에 이더리움의 솔리디티 언어에 대해 분석한다. 또한 블록체인 분야의 설계 사례를 연구한다. 3장에서는 스마트 컨트랙트의 정적/동적 설계 기법을 제안하며 이더리움 솔리디티 언어와 맵핑하여 구현 관계를 확인한다. 4장에서는 부동산 계약 사례를 통해 스마트 컨트랙트를 설계 및 구현 검증 한다. 5장에서 결론 및 향후 연구과제를 제시한다.

2. 관련연구

2.1 스마트 컨트랙트

스마트 컨트랙트는 계약의 협상을 디지털 방식으로 시행하기 위한 프로토콜이다[1]. 제3자의 개입 없이 신뢰할 수 있는 거래를 할 수 있으며 이러한 거래는 추적이 가능하고 변경될 수 없도록 한다. 스마트 컨트랙트에 의한 계약은 기존의 오프라인 계약 보다 우수한 보안을 제공하고 거래 비용을 줄일 수 있다. 암호화폐에서 스마트 컨트랙트를 구현하기 위한 방법으로 비트코인은 다중서명 계정(Multisignature accounts), 지불 채널(Payment channels), 조건부 날인(Escrow), 시간 잠금(Time locks), 등과 같은 스마트 컨트랙트를 생성하기 위해 튜닝 완전[2]스크립트(Tuning-incomplete Script) 언어를 제공한다[3,4]. 이더리움(Ethereum) 또한 스마트 컨트랙트 프레임워크인 블록체인을 제공하기 위해 튜닝 언어를 제공한다[5]. 본 연구에서는 이더리움의 스마트 컨트랙트를 구현하기 위한 솔리디티(Solidity) 언어의 사례를 기반으로 스마트 컨트랙트 설계 기법을 제안한다.

2.3 솔리디티

솔리디티는 스마트 컨트랙트를 개발하기 위한 계약 지향 프로그래밍 언어이다[6]. 다양한 블록체인 플랫폼에서 스마트 컨트랙트를 개발하기 위해 사용된다[7]. 솔리디티는 EVM(Ethereum Virtual Machine)에서 실행되는 스마트 계약을 개발하기 위한 정적 형식의 프로그래밍 언어로서[8,9], 바이트 코드(Bytecode)로 컴파일 되고 EVM 상에서 실행된다. ECMAScript[10]구문을 따르며 웹 개발에 적합하도록 설계되었다. 솔리디티는 계층적 맵핑과 구조체를 멤버 변수로 정의할 수 있으며 다중 상속을 지원한다. 개발된 컨트랙트의 함수에 대해 여러 유형의 안전한 접근을 지원하기 위해 ABI(Application Binary Interface)[11]기능을 제공한다. 개발자는 솔리디티를 이용하여 자체적인 비즈니스 로직을 포함하는 스마트 컨트랙트를 개발할 수 있으며 계약 완료 후에 당사자 간에 거부할 수 없으며 신뢰할 수 있는 트랜잭션을 보관한다.

2.4 식품 이력 인증 서비스를 위한 블록체인 시스템 설계

연구 [12]는 식품 유통과정의 안정성을 보장하기 위해 공개키 기반의 블록체인 이력 인증 시스템 설계를 제안한다. 연구 [12]는 식품의 생산, 유통, 소비에 관련된 데이터를 분산 저장하는 EPCIS(Electronic Product Code Information Service)를 활용하고 공개키 서명을 이용하여 블록체인을 형성할 수 있는 이력 인증 체계를 설계하였다. EPCIS 표준을 기반으로 분산 저장하기 위한 아키텍처 와 식품 이력을 위한 문서 구조, 그리고 문서 간의 인증 절차 등을 제안하였다.

연구 [12]에서 제안하는 식품 인증 도메인 측면에서 블록체인 구조 및 문서 구조를 제안하는데 반해, 본 연구에서는 다양한 블록체인 서비스 개발 시 설계할 수 있는 일반적인 설계 방안을 제안한다.

2.5 UML을 이용한 아두이노 어플리케이션 설계

연구 [13]은 아두이노(Arduino) 어플리케이션을 개발하기 위한 효과적인 설계 방식이 미흡하므로 표준 객체지향 설계 언어인 UML의 확장 메커니즘(스테레오 타입(Stereotype), 제약사항(Constraint), 태그정의(Tag Definition), 태그값(Tagged Value))을 이용하여 설계 방식을 제안한다.

Stereotype	Description
<<Arduino>>	Arduino Board
<<Digital Input>>	Digital Input Device
<<Digital Outout>>	Digital Output Device
<<Analog Input>>	Analog Input Device
<<Analog Outout>>	Analog Output Device
<<Value>>	Input/Output Value of Digital and Analog

Fig. 1. Class Stereotype for Arduino

Fig. 1에서와 같이 아두이노의 디바이스 관련 표기를 위해 클래스의 스테레오 타입 설계 방안을 제시한다. 또한 디바이스 클래스들 간의 관계(Association)를 위한 스테레오 타입 설계 방안도 제안한다. 이와 같이 UML 표기법을 이용하여 아두이노 어플리케이션을 설계하기 위한 방안을 제시한다. 본 논문에서도 확장 메커니즘의 스테레오 타입을 이용하여 블록체인 기반의 스마트 컨트랙트를 설계하기 위한 방안을 제안한다.

2.6 iOS 어플리케이션을 위한 UML프로파일

연구 [14]는 UML의 확장 메커니즘을 이용하여 iOS 어플리케이션의 설계 방안을 제안한다. iOS 어플리케이션 설계를 위한 프로파일로서 Object-C 프로파일, Foundation 프로파일, iOS 프로파일을 정의하였으며, 스테레오 타입을 이용한다. Fig. 2는 Object-C 프로파일의 스테레오 타입으로서 Object-C 의 특성을 표현할 수 있는 설계 방안이다.

Stereotype	Description
Class	
<<Protocol>>	Classes that only provide interfaces
<<Category>>	Used when you want to add a method without modifying an existing class
Operation	
<<Required>>	Protocol methods must be implemented
<<Optional>>	Method that is not required to be implemented in Protocol method

Fig. 2. Stereotype of Object-C profile

연구 [14]와 같이 본 논문에서도 UML의 확장 메커니즘인 스테레오 타입을 이용하여 블록체인 기반의 설계 방안을 제안하고자 한다.

3. 블록체인 기반 스마트 컨트랙트의 정적/동적 설계 기법

본 연구에서는 블록체인의 스마트 컨트랙트 정적/동적 설계를 위한 기법을 제안 한다. 스마트 컨트랙트 설계 기법은 이더리움의 솔리디티를 개발할 수 있도록 객체지향 설계 기법인 UML의 확장 메커니즘을 이용하여 제안한다.

3.1 스마트 컨트랙트의 정적 설계 기법

스마트 컨트랙트를 구성하기 위한 메타모델은 Fig. 3과 같다.

스마트 컨트랙트(이후 컨트랙트)의 기능을 제공하기 위한 Contract와 구성 요소인 Attribute와 Function으로 구성되며, Contract는 거래 내역을 제공하는 Transaction과 함께 Block 내에 포함된다. Block들 간에는 작업증명(Proof of Work) 관계가 형성되며 블록체인 내의 계정인 Account에 의해 참조된다. Account들 간에는 네트워크를 구성하여 블록을 형성한다. 일반적으로 블록을 형성하기 위해 계정 간에 암호화폐를 송금하여 블록을 형성한다. Account, Block, Transaction은 블록체인 내에 포함되어 있으므로 본 연구에서는 Contract에 집중하여 설계 기법을 제안한다. 그러나 동적 설계 시 Account와 Block은 배포 및 동기화를 설계하기 위해 포함하여 설계한다.

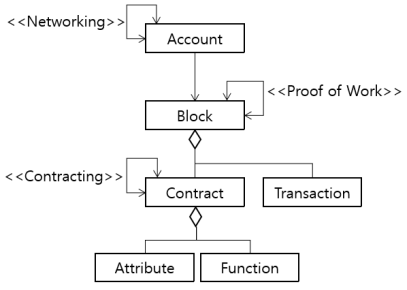


Fig. 3. Meta-Model for Designing Smart Contract based on Block Chain

컨트랙트는 Fig. 4와 같이 객체지향 설계표기의 클래스로 설계한다. 컨트랙트명 위에 스테레오 타입(Stereotype)을 <<Contract>>로 정의한다. 컨트랙트는 데이터에 해당하는 Attribute와 행위에 해당하는 Function으로 구성된다.

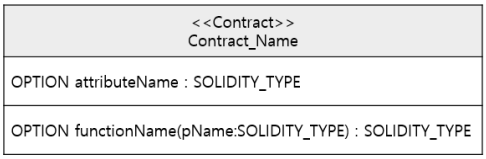


Fig. 4. Design Notation of Contract

컨트랙트에서 속성의 설계는 Fig. 5와 같다. UML의 문법을 그대로 사용하며 가시성(Visibility) 영역에 option을 정의하며 데이터 타입은 솔리디티에서 제공하는 타입을 정의한다.

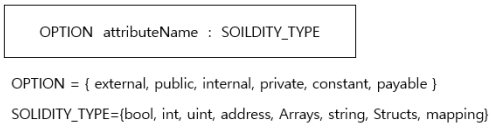


Fig. 5. Design Notation of Contract's Attribute

솔리디티에서 제공하는 데이터 타입은 값과 참조 형식으로 구분하며 값을 저장하기 위한 변수는 bool, int, uint, address 가 있다. address는 계정의 주소를 저장하기 위한 변수로서 20 바이트 크기의 변수이다. 참조 형식의 변수는 Arrays, string, Structs, mapping 이 있다. mapping 변수는 키와 값의 매핑을 지정할 때 사용하는 변수이다. 가시성에 해당하는 OPTION은 Fig. 6과 같이 public과 private은 UML의 '+'와 '-'를 그대로 적용하며, external, internal, constant, payable은 스테레오 타입을 이용하여 적용한다.

OPTION	Design Notation
external	<<external>>
public	+
internal	<<internal>>
private	-
constant	<<constant>>
payable	<<payable>>

Fig. 6. Design Notation of Attribute's Option

컨트랙트의 함수 설계는 Fig. 7과 같다. 함수 앞에 OPTION을 명시하며, 가시성을 포함하여 솔리디티 언어에서 제공하는 추가적인 접근자를 포함한다. 예를 들면 payable은 블록체인 내의 다른 전자지갑으로 부터 송금을 받을 수 있음을 나타내는 함수를 정의하고자 할 때 선언한다. 이와 같이 블록체인 특성에 맞는 접근자를 포함하여 설계한다. 반환 타입은 기본형 데이터 타입인 bool, int, uint 외에 계정 주소값에 해당하는 address 등의 추가적인 데이터 타입을 설계한다.

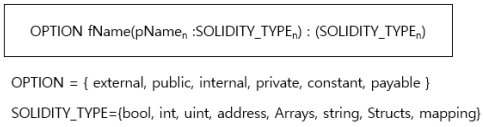


Fig. 7. Design Notation of Contract's Function

컨트랙트 함수 설계와 솔리디티 코드 간의 관계는 Fig. 8과 같다. OPTION와 함수명이 바뀌어서 구현되어야 하며 OPTION 앞에는 returns 예약어를 포함하여 구현한다.

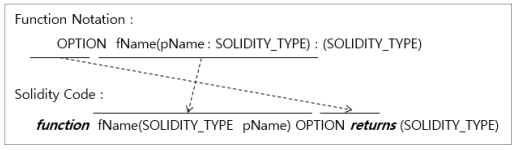


Fig. 8. Mapping of Contract Function Notation and Solidity Code

컨트랙트들 간의 관계 설계는 크게 연관(Association), 상속(Inheritance), 구현(Implements) 으로 크게 구분하여 설계할 수 있다.

컨트랙트 연관설계는 Fig. 9와 같다. 컨트랙트들 간의 연관관계에 <<contract_association>> 스테레오 타입을 이용한다. 연관관계 설계에 대한 솔리디티 코드는 Fig. 10과 같다. Fig. 10 사례는 A 컨트랙트 함수 내에서 B 컨트랙트를 생성하여 연관관계를 갖는 사례이다.

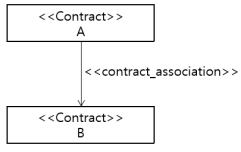


Fig. 9. Design Notation of Contract's Association

```
contract A {
    function fun( ) {
        B b = new B( );
        b.call( );
    }
```

Fig. 10. Solidity Code of Contract's Association

컨트랙트 상속 관계는 Fig. 11과 같다. 컨트랙트들 간의 상속관계는 <<contract_inheritance>> 스테레오 타입 표기법을 사용하며, 솔리디티 코드는 Fig. 12와 같이 is 예약어를 사용하여 상속 관계를 구현한다.

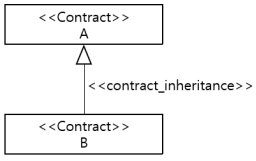


Fig. 11. Static Notation of Contract's Inheritance

```
contract A { ... }
contract B is A { ... }
```

Fig. 12. Solidity Code of Contract Inheritace

컨트랙트 인터페이스 구현 관계는 Fig. 13과 같이 <<contract_implements>> 스테레오 타입을 이용하여 설계한다. 구현은 Fig. 14와 같이 interface 예약어를 이용하여 인터페이스를 선언하며 상속과 동일하게 is 예약어를 이용하여 구현한다.

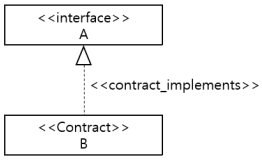


Fig. 13. Contract Interface Design

```
interface A { ... }
contract B is A { ... }
```

Fig. 14. Solidity Code of Contract Interface

3.2 스마트 컨트랙트의 동적 설계 기법

컨트랙트의 동적설계는 객체지향 설계 기법의 순차도(Sequence Diagram)을 활용한다.

동적설계는 정적설계에서 정의한 컨트랙트를 기반으로 설계하며 블록체인 내의 계정과 블록 인스턴스(instance)를 포함하여 설계한다.

블록체인 내의 인스턴스들을 스테레오 타입을 이용하여 정의하며 스테레오 타입은 다음과 같다.

- <<contract>> : 컨트랙트 인스턴스
- <<account>> : 블록을 포함하고 있는 계정 인스턴스
- <<block>> : 컨트랙트를 포함하고 있는 블록 인스턴스

서비스 흐름에 대한 명시를 위해 다음과 같은 스테레오 타입을 정의한다.

- <<deploy>> : 현재 계정에서 블록체인 내의 다른 계정으로 컨트랙트 배포
- <<create>> : 작업증명(Proof of Work) 후에 컨트랙트 생성
- <<sync>> : 블록체인 내의 컨트랙트들 간의 동기화

컨트랙트가 초기화 되면 실행할 수 있도록 다른 계정으로 배포해야 한다. Fig. 15와 같이 현재 계정에서 다른 계정의 블록으로 배포가 이루어 질 수 있도록 설계한다.

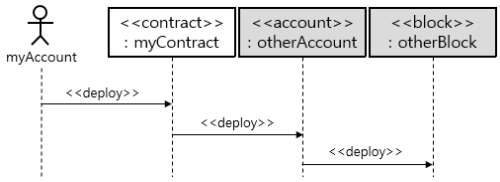


Fig. 15. Dynamic Design of Contract Deployment (Concept)

Fig. 15의 설계에 대해 Go Ethereum[15]환경에서 컨트랙트를 배포하기 위한 코드는 Fig. 16과 같다. eth.account[0]가 현재 계정을 의미하며 Contract 를 다른 계정으로 배포됨을 의미한다.

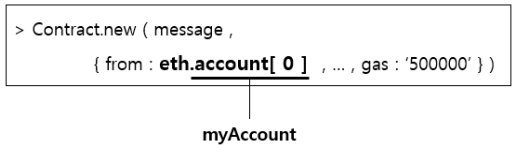


Fig. 16. Contract Deployment in Go Ethereum

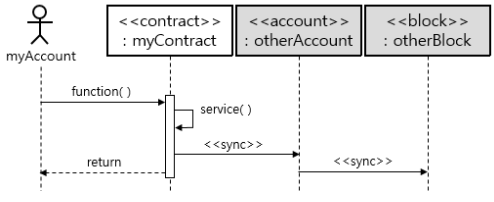


Fig. 17. Dynamic Design of Contract Service Call and Synchronization(Concept)

컨트랙트가 배포되면 컨트랙트의 기능을 호출하여 실행할 수 있다. 그런데 블록체인 내에서는 실행된 처리 결과가 다른 계정의 블록들과 동기화를 해야 한다. Fig. 17에서와 같이 service() 기능을 호출하며 다른 계정의 블록으로 동기화를 수행할 수 있도록 설계한다.

Fig. 17의 설계에 대해 Go Ethereum환경에서 컨트랙트 기능을 호출한 후 동기화하기 위한 코드는 Fig. 18과 같다. Contract.function 이 컨트랙트의 기능을 호출함을 의미하며 다른 계정에 배포된 컨트랙트와 동기화가 된다.

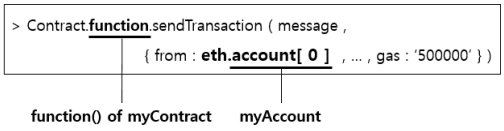


Fig. 18. Contract Function Call in Go Ethereum

배포와 동기화는 블록체인 플랫폼 내에서 수행되므로 설계가 유사하지만 컨트랙트 내의 기능에 대한 동적 설계는 매번 다르게 설계되어야 한다.

Fig. 19는 Fig. 15의 컨트랙트 배포에 대해 상세한 동적 설계이다. 배포는 블록체인 내에 연결된 블록들 간에 배포가 이루어져야 한다. 반복문을 통해 배포가 이루어지며 다른 계정의 블록에 배포가 되어 생성된다. 이때 생성(<<create>>)은 작업증명이 이루어져 블록체인 내에 확정됨을 의미한다.

배포는 주로 블록체인 플랫폼에 의해 이루어지며 어떤 계정으로 배포해야할지를 동적설계를 통해 설계한 후 블록체인 플랫폼에 적용해야 한다.

컨트랙트 기능 호출 및 동기화에 대한 상세설계는 Fig. 20과 같다. 배포와 유사하게 블록체인 내의 계정(블록)들 간에 <<sync>> 표기를 이용하여 동기화됨을 설계한다. 기능 호출 시 반드시 블록들 간에 동기화가 발생한다.

4. 실험 및 평가

본 사례연구에서는 제안한 스마트 컨트랙트 정적/동적 설계 기법을 검증하기 위해 Fig. 21, Fig. 22와 같은 부동산 계약을 위한 기능을 정의하며, 설계의 적합성을 확인하기 위해 솔리디티 언어를 이용하여 구현한다.

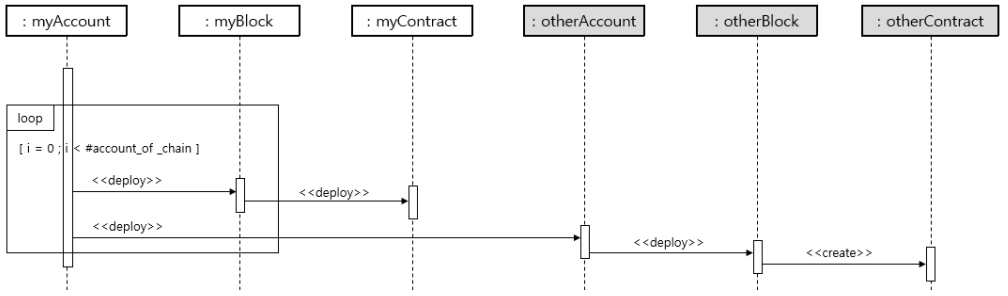


Fig. 19. Dynamic Design of Contract Deployment(Detail)

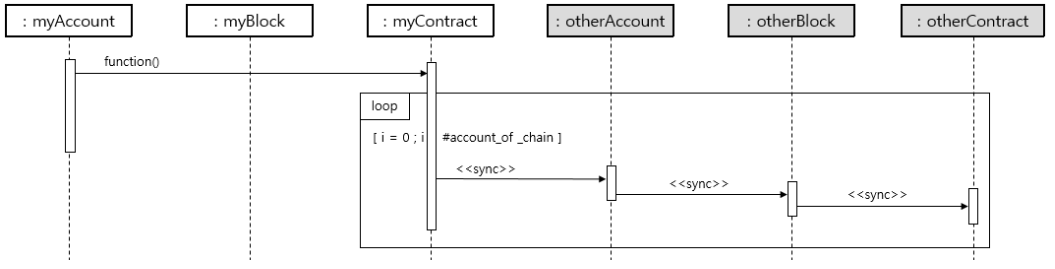


Fig. 20. Dynamic Design of Contract Service Call and Synchronization(Detail)

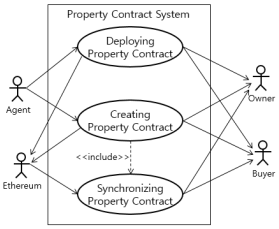


Fig. 21. Use Case Model of Property Contract

Fig. 22. UI Design of Property Contract

부동산 계약의 스마트 컨트랙트를 위한 정적설계는 Fig. 23과 같이 3개의 컨트랙트로 구성한다. 블록체인 내의 시스템 요소인 블록와 계정에 대한 요소는 정적설계에서 제외하였다.

부동산 계약 컨트랙트인 Property_Contract는 상위 컨트랙트의 Contract를 상속 받으며 소유자와 구매자 정보를 포함하고 있는 Contractor와 연관관계를 가진다. 계약 매매자의 id는 계정 address를 가지게 되며 계약서가 배포되고 동기화되는 주소가 된다.

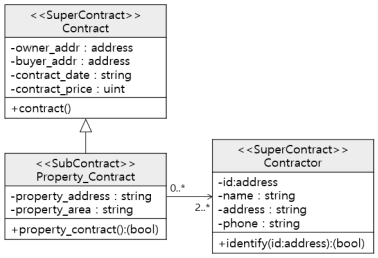


Fig. 23. Static Design of Property Contract

부동산 계약 컨트랙트의 정적 설계를 기반으로 Fig. 24와 같이 계약 기능에 대한 동적 흐름을 설계한다. agent는 부동산 중계자 계정을 의미하며 중계자에 의해 Property_Contract가 생성된다. 생성된 컨트랙트는 블록체인으로 연계된 계정(소유자, 구매자)에게 배포되며 각각의 계정에서 인스턴스(객체)로 생성된다. 블록체인으로 연결하기 이더리움에서는 계정 간에 ether를 송금하여 블록체인을 형성하는데 본 사례연구에서는 ether를 송금하여 블록체인을 형성하는 동적설계를 생략한다.

부동산 계약을 하기 위한 컨트랙트의 property_contract() 기능 흐름 설계는 Fig. 25과 같다. 소유자와 구매자의 계정 주소와 기타 계약 관련 정보를 입력하면 정적설계에서 설계한 컨트랙트인 Property_Contract, Contractor 의 인스턴스 컨트랙트를 기반으로 소유자와 구매자를 확인한다. 확인이 처리되면 계약 관련 정보를 Property_Contract에 저장하여 관리한다.

부동산 계약 관련 정보가 저장되면 블록체인 내에 소유자와 구매자 블록 내의 컨트랙트와 동기화를 수행한다. Fig. 26과 같이 블록체인 내에 연결된 블록들에게 동기화하는 작업을 수행할 수 있도록 설계한다.

부동산 계약 컨트랙트에 대한 솔리디티 코드는 Fig. 27과 같다. Property_Contract는 상위 컨트랙트의 Contract 를 상속 받으며 사용자 주소 확인을 위해 Contractor 와 연관관계를 갖는다. property_contract() 기능에 대한 동적 설계는 Fig. 25와 동일하다.

정적설계와 동적설계를 기반으로 이더리움으로 구현된 화면은 Fig. 28과 같다. 결과 화면은 전자지갑인 Mist 블라우저[16]를 통해 컴파일되어 실행하였다. Fig. 28에서와 같이 Owner와 Buyer가 동기화를 통해 동일한 부동산 계약 컨트랙트를 소유하고 있음을 확인할 수 있다.

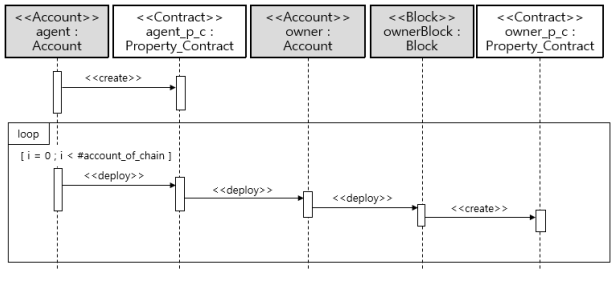


Fig. 24. Dynamic Design of Property Contract Deployment

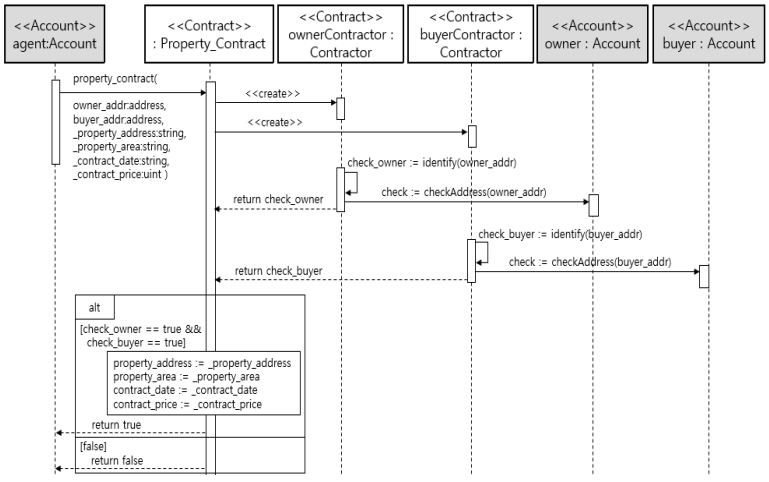


Fig. 25. Dynamic Design of Property Contract Creation

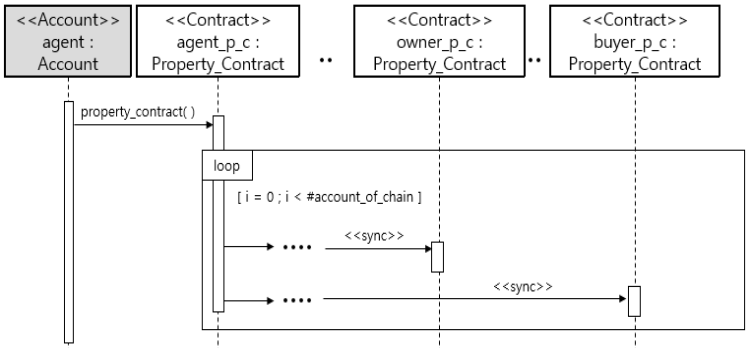


Fig. 26. Dynamic Design of Property Contract Synchronization

```
contract Property_Contract is Contract {
    string property_address;
    string property_area;
    string contract_date;
    uint contract_price;

function property_contract(address owner_addr,
            address buyer_addr,
            string _property_address,
            string _property_area,
            string _contract_date,
            uint _contract_price) public{
      Contractor owner = new Contractor(owner_addr);
      Contractor buyer = new Contractor(buyer_addr);
      if (owner.identify(owner_addr) && buyer.identify(buyer_addr)) {
        property_address = _property_address;
        property_area = _property_area;
        contract_date = _contract_date;
        contract_price = _contract_price;
      }
    }
}
```

Fig. 27. Property Contract Solidity Code

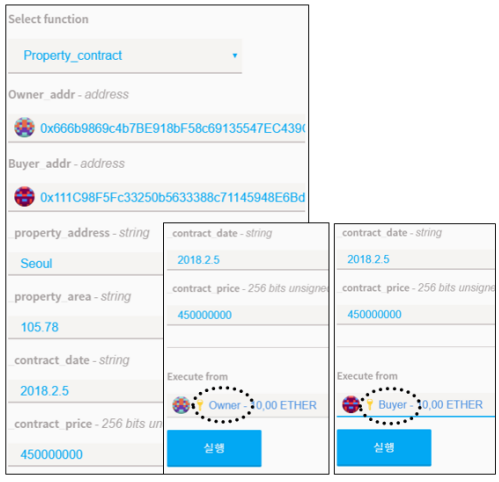


Fig. 28. Property Contract UI in Mist Browser

부동산 계약 컨트랙트가 배포된 트랜잭션 정보는 Fig. 29와 같다. Fig. 30은 동기화 후에 블록의 작업증명(Proof of Work)[17] 과정을 나타내고 있다. 12개의 블록 중에 9개의 블록에 대한 작업증명이 완료된 상태를 나타내고 있다.

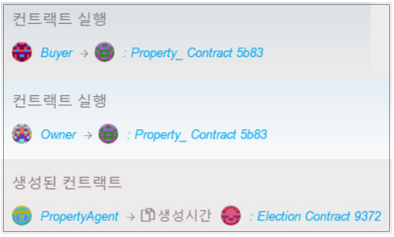


Fig. 29. Transaction List of Property Contract Deployment

Fig. 30. Proof of Work Status

지금까지 본 논문에서 제안한 블록체인 스마트 컨트랙트의 정적/동적설계 기법을 검증하기 위해 부동산 계약의 스마트 컨트랙트에 대해 정적/동적설계를 적용하였다. 또한 설계의 타당성을 검증하기 위해 솔리디티 언어를 통해 구현하였다.

5. 결론

본 연구에서는 블록체인 기반의 스마트 컨트랙트를 설계하기 위한 메타모델과 정적/동적설계 기법을 제안하였다. 스마트 컨트랙트를 구현하기 위한 이더리움의 솔리디티 언어가 객체지향 언어와 유사하므로 객체지향 설계 기법의 확장 메커니즘을 적용하는 것이 적합하였으며 동적설계 시 블록체인 기반으로 서비스 되어야 하므로 배포 및 동기화에 대한 설계를 고려하였다. 부동산 계약 컨트랙트 설계 사례를 통해 블록체인의 스마트 컨트랙트 설계가 적합함을 증명하였으며 정적/동적설계와 솔리디티 코드가 일관성 있음을 증명하였다. 본 연구를 통해 기존 UML의 확장 프로파일에 블록체인 설계 기법을 추가할 수 있도록 하여 블록체인 어플리케이션 설계의 기반을 제공할 수 있을 것이다.

향후에는 본 연구에서 제안한 정적/동적설계 기법을 프로그래밍이 가능한 다른 블록체인 플랫폼에 적용하여 일반적인 스마트 컨트랙트 설계 기법으로 고도화할 수 있도록 한다.

References

[1] Nick Szabo, "Smart Contracts: Building Blocks for Digital Markets", http://www.fon.hum.uva.nl, 1996, retrived 19 December 2017.

[2] ArthurB, "Smart contracts: Turing completeness & reality", https://hackernoon.com/smart-contracts-turing-completeness-reality-3eb897996621, Oct 3, 2016, retrived 19 December 2017.

[3] Davide De Rosa, "The Bitcoin Script language in basic blockchain programming", http://davidederosa.com/basic-blockchain-programming/bitcoin-script-language-part-one/, May 25, 2015, retrived 19 February 2018.

[4] Atzei Nicola, Bartoletti Massimo, Cimoli Tiziana, Lande Stefano, and Zunino Roberto, "SoK: unraveling Bitcoin smart contracts", 7th International Conference on Principles of Security and Trust , European Joint Conferences on Theory and Practice of Software, 2018.

[5] Atzei Nicola, Bartoletti Massimo, and Cimoli Tiziana, "A survey of attacks on Ethereum smart contracts", 6th International Conference on Principles of Security and Trust, European Joint Conferences on Theory and Practice of Software, 2017.

[6] Allison Ian, "PwC blockchain expert pinpoints sources of ambiguity in smart contracts", IBTimes, 12 August 2016.

[7] Alyssa Hertig, "Blockchain Veterans Unveil Secure Smart Contracts Framework", CoinDesk, 15 September 2016.

[8] Mougayar, William, *The Business Blockchain: Promise, Practice, and Application of the Next Internet Technology*, Wiley Publishing, 2016.

[9] Allison Ian, "Microsoft adds Ethereum language Solidity to Visual Studio", International Business Times, March 2016.

[10] Stefanov, Stoyan, *JavaScript Patterns*, O'Reilly Media, 2010.

[11] ABI(Application Binary Interface), https://en.wikipedia.org/wiki/Application_binary_interface, retrived 19 February 2018.

[12] Sangtae Kim, Seunghyeon Moon, Seungyong Jung, Sooji Jeon, and Sungkwan Jung, "A Design of EPCIS Block-chain System for Food Safty Service", Proceedings of the Korean Institute of Communication Sciences Conference, 2017.

[13] Ki Chang Park, Hyun Cheol Lee, and Eun Seok Kim,

"A Software Design Method for Arduino Applications using UML", Journal Of The Korea Contents Association, 15(8), August 2015.
DOI: http://dx.doi.org/10.5392/JKCA.2015.15.08.001

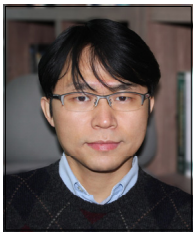
[14] Yong jin Seo, Dea geon Mun, Seung hak Kuk, and Hyeon Soo Kim, "UML Profile for iOS Application", Preceeding of KOREA INFORMATION SCIENCE SOCIETY, 38(1B), June 2011.

[15] Go Ethereum(Geth), https://ethereum.github.io/go-ethereum/downloads, retrived 10 December 2017.

[16] Mist and Ethereum Wallet, https://github.com/ethereum/mist/releases, retrived 10 October 2017.

[17] PoW(Proof of Work), https://etherworld.co/2017/04/16/proof-of-work-pow/, retrived 23 February 2018.

김 철 진(Chul-Jin Kim) **[종신회원]**



- 2004년 2월 : 숭실대학교 대학원 컴퓨터학과 (공학박사)
- 2004년 3월 ~ 2009년 2월 : 삼성전자 책임연구원
- 2009년 3월 ~ 현재 : 인하공전 컴퓨터시스템과 부교수

<관심분야>
객체/컴포넌트/서비스 커스터마이제이션, 모바일 서비스, 아키텍처 리팩토링, 블록체인, 이더리움