Abstract

Setup on the paper 'Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach' by Anastasia Mavridou(Vanderbilt University) and Aron Laszka(University of Houston) to evaluate possibilities of visualization tools for the development of smart contracts. Especially the approach of linked contracts will be considered in this paper.

Keywords: Smart contract, security, finite state machine, Ethereum, Solidity, automatic code generation, design patterns, linked contracts

Visualisation and code generation of linked smart contracts as an extension of FSolidM

Thies Weinand
Flensburg University of Applied Sciences, Germany
thies.weinand@stud.hs-flensburg.de

March 1, 2021

1 Introduction

At the beginning of 2021 is obviously that crypto currencies become more and more importance in the global financial and economy world. Companies like Tesla invest 1,5 billion dollars in Blockchain based crypto currencies and the amount of money in the marked exploding. One of the major topics is the security of the underlying technology. Ethereum in one of the big currencies in the market and will be handled in so called smart contracts. These small contracts should be free of deadlocks or other vulnerabilities to guarantee the owner of Ether that the invested money cannot be stolen. To support the smart contract developer could visualisation tools like FSolidM or YAKINDU Solidity Tools help to avoid the deployment of unsecure smart contracts. This paper focuses the possibilities which FSolidM offering.

2 Why using visualization

A possible hazard during the design of smart contracts is that interactions between the contracts cannot be surveyed by the contract developer because complexity and encapsulation increasing. The solution for this problem is the visualisation of the dependencies of so called "Linked Contracts". You

can also easily create a contract by drag and drop items and create a state machine.
A disadvantage of the actual Etherium implementation is that once time deployed contracts cannot changed anymore. If you would like to change anything on a deployed contract, you only have the chance to embed the deployed contract as a subcontract and link the new fixed contract with the subcontract. To enabling this functionality is the main goal of this extension.

3 Introducing the FSolidM Tool

The FSolidM tool is a state chart tool for finite state machines (FSM) it is based on WebGme, a web based, collaborative, versioned, model editing framework from the Vanderbilt University. The FSolidM tool was published in 2018 during the International Conference on Financial Cryptography and Data Security by Anastasia Mavridou(Vanderbilt University) and Aron Laszka(University of Houston) [1]

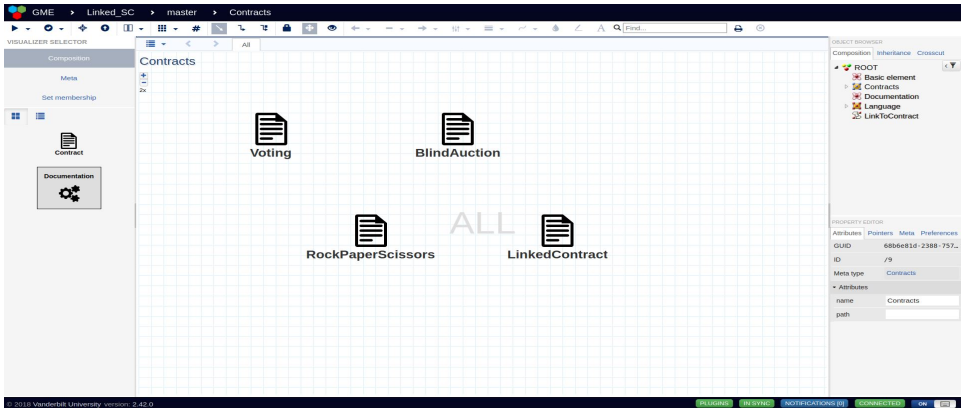

Figure 1: FSolidM contract view

The FSolidM developers focus on the known vulnerabilities of smart contracts and the solidity code generation. They had implemented security patterns which are checking the finite state machine(FSM) for deadlocks or other vulnerabilites. They also introduce a graphical FSM Editor for designing and linking the smart contract.

Additionally a code editor is implemented where the generated code is visible and code snippets can be added to complete the contract. Solidity files will be also generated and stored in the local storage.
The FSM editor consists of several items which can be placed by drag and drop. Normal states, initital states, transitions and timed transitions are basic elements of the FSM editor.
The Solidity Code Generator provides checks whether the FSM has formal the correct pattern for example the generator checks that an initial state for the FSM is defined. The possibility of linking two contracts is not implemented in the tool but described in the root paper[1] as an possible extension.

4 FSolidM extension

Picking up the described idea in the root paper was the root cause for the creation of this extension. For the extension was the metamodel modified a new connection type was introduced that allows two contracts to interact.

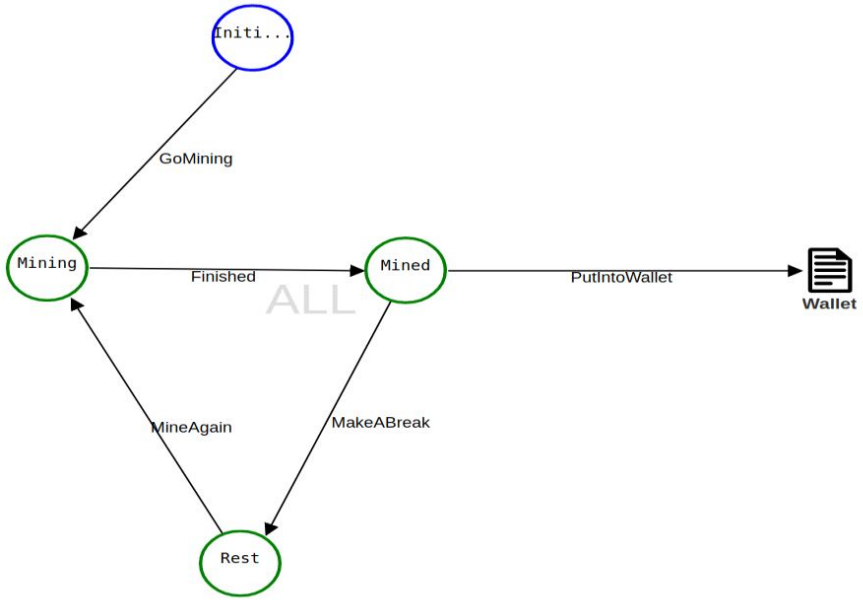

Figure 2: State Machine with Linked Subcontract

Code snippets were added in the Solidity Code Editor and in the Solidity Code Generator. The introduced code snippets will establish a connetction automatically in case of linking a state with a subcontract.

```
//Links to Contracts
  function PutIntoWallet (address Address) public

  {
      require(state == States.Mined);

      bytes memory function2call = abi.encodePacked(keccak256("Receiver()"));
      bool feedback = false;
      bytes memory MemFeedback;
      (feedback,MemFeedback) = Address.call(function2call);

  }
```

Figure 3: Function which enables linking

Additionally were some useful standard functions introduced for example a function which provides you the actual state of a state machine or a value of a predefined variable.

```
//Standard function - Get actual state
function showState() public view returns (string memory){
  if(state == States.InitialState){
    return "InitialState";
  }
  else if (state == States.Mined){
    return "Mined";
  }
  else if (state == States.Mining){
    return "Mining";
  }
  else if (state == States.Rest){
    return "Rest";
  }
return "Stateless";
}
```

Figure 4: Standard function which provides the actual state of the FSM

A new seed which consists the new standard types was also created. This seed can be chosen after creation of a new project at program start.

5 Conclusion and future work

FSolidM makes a step forward to become a performant tool for the development of smart contracts. New useful features were implemented and tested. Now is it possible to reengineer a flawed smart contract and fix the contract by embed him into another contract.
A getting started was added to the repository which should help new users to become familiar with FSolidM and encourages to develop the tool.

A possible extension for the future is, that more than one state machines can be drawn at one page and interacting visual and on code base together. Therefore, is a reconstruction of the Solditiy Code Generator and the Solidity Code Editor necessary.

References

[1] Anastasia Mavridou and Aron Laszka. Designing secure ethereum smart contracts: A finite state machine based approach. 2017.