IoT Blockchain

Smart Contract and Smart Meters

Functional Specification

Version 0.1

Yunli Wang

(yunli.wang@sjsu.edu)

[Version History](#h.gjdgxs)

[Introduction](#h.30j0zll)

[References](#h.1fob9te)

[Requirements](#h.3znysh7)

[Functional Overview](#h.2et92p0)

[Configuration/ External Interfaces](#h.tyjcwt)

[Debug](#h.3dy6vkm)

[Logging](#h.1t3h5sf)

[Counters](#h.4d34og8)

Implementation

[Testing](#h.17dp8vu)

[General Approach](#h.3rdcrjn)

[Unit Tests](#h.26in1rg)

[Appendix](#h.lnxbz9)

# Version History

|  |  |
| --- | --- |
| **Version** | **Changes** |
| 0.1 | Initial version |

# Introduction

Solidity is a development language for developing distributed decentralized applications (Dapps) on the Ethereum Virtual Machine (EVM). Its syntax is very similar to JavaScript but is statically typed.

As we are developing our IoT devices on the blockchain provided by Ethereum, it will be in the form of a Dapp that is running on the EVM. Therefore, it is natural to use solidify as our main development language.

The application will be centered around a smart contract that handles transactions between households who sell excessive solar power or buy them when necessary.

# References

<https://solidity.readthedocs.io/en/develop/>

<http://ethdocs.org/en/latest/introduction/what-is-ethereum.html>

# Requirements

Solidity programming language

Web 3 compiler

Ethereum Virtual Machine (homestead images)

# Functional Overview

We will first build a smart contract that can accept sell and buy requests from smart meters (clients). The smart contract will look at the asking price of each request and try to match them as much as possible, much like a stock market. A client will be able to withdraw a request and resubmit with a updated price at any time as long as the transaction has not completed.

## Configuration/ External Interfaces

* No external configuration required
* Dependency  
  - successfully set up Ethereum Virtual Machine on every participating node (computers)  
  - Successful implementation of a data structure like priority queue that can easily retrieve the highest bid and lowest offer  
  - Functioning smart contracts and smart meters program that transmit and receive data on the network

## Debug

Debugging will be done as we develop the app. Mostly by logging every transaction to the console.

### Logging

In the logs, we will record the originating party, amount offered and price, as well as the receiving party, amount transacted, date and time.

### Counters

No counters will be used as of now.

# Implementation

1. It seems that Solidity does not have many built-in data structures so we need to implement our own version of priority queue to track the bids and offers. This may not be an easy task. Once this is done, we can move on to create the smart contract that deals with client requests.
2. The interface for the contract will be very simple:
   1. buy(price, amount)
   2. sell(price, amount)
3. The smart contract will only return when a transaction is made, which may be challenging as we know blockchain usually does not support concurrency.

# Testing

## General Approach

Testing the functionalities of our smart contract and smart meters is straightforward. We send both buy and sell requests to the smart contract from different clients. There are several scenarios:

* + - 1. A sends a buy request and B sends a sell request with the same price and amount
      2. A sends a buy request and B sends a sell request with the same price but different amount
      3. A sends a buy request and B sends a sell request with a lower price and the same amount (success)
      4. A sends a buy request and B sends a sell request with a higher price and the same amount (fail)
      5. Some other partial combinations of above examples.

## Unit Tests

As of now, there does not seem to be a unit test facility available in Ethereum and Solidity. We will keep exploring all possibilities.

# Appendix

https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial