Hukkinen, Taneli; Mattila, Juri; Seppälä, Timo

Research Report

Distributed Workflow Management with Smart Contracts

ETLA Report, No. 78

Provided in Cooperation with:
The Research Institute of the Finnish Economy (ETLA), Helsinki

Suggested Citation: Hukkinen, Taneli; Mattila, Juri; Seppälä, Timo (2017) : Distributed Workflow Management with Smart Contracts, ETLA Report, No. 78, The Research Institute of the Finnish Economy (ETLA), Helsinki

This Version is available at:
https://hdl.handle.net/10419/201360

ETLA ELINKEINOELÄMÄN TUTKIMUSLAITOS
THE RESEARCH INSTITUTE OF THE FINNISH ECONOMY

ETLA Raportit
ETLA Reports

23 October 2017

No 78

Distributed Workflow Management with Smart Contracts

Taneli Hukkinen* – Juri Mattila** – Timo Seppälä*, **

* Aalto University, School of Science, Department of Industrial Engineering and Management; firstname.lastname@aalto.fi
** ETLA – The Research Institute of the Finnish Economy, firstname.lastname@etla.fii

Suggested citation: Hukkinen, Taneli, Mattila, Juri, & Seppälä, Timo (23.10.2017). "Distributed Workflow Management with Smart Contracts". ETLA Reports No 78. https://pub.etla.fi/ETLA-Raportit-Reports-78.pdf

This collaboration towards this research and development process was initiated in March 2016 under the BRIE-ETLA research project and continued under the BOND research project. The authors would like to thank Anu Saarinen and Jyrki Korhonen from Euroclear Finland, and Pirkka Frosti from Digital Living for their insightful comments and co-operation whilst drafting this document.

ISSN-L 2323-2447
ISSN 2323-2447 (print)
ISSN 2323-2455 (pdf)

Table of Contents

Distributed Workflow Management with Smart Contracts

Abstract

This report documents a blockchain application developed for the real estate sector. The application enables distributed workflow management in a complicated transaction process: the selling of a share of stocks in a housing corporation. As its core element, the application utilizes Ethereum-based smart contracts to facilitate the interaction of various parties involved, as well as the Interplanetary File System (IPFS) to combine data from a number of separate information pools. The motive for this application has been to understand the process of developing blockchain applications with industrial partners. Moreover, the purpose of this exercise has been to examine whether Ethereum-based smart contracts could be effectively utilized for applications in industry and finance. The application and the discussions during its development indicate that similar, market-driven workflow structures may appear in value chains where the number of parties is high and where the sources of information are numerous yet disconnected.

Key words: Blockchain, application, distributed workflow, real-estate, Ethereum, smart contract

JEL: L1, L8, L17, L73, L85

Hajautettu työnkulun hallinta älykkäillä sopimuksilla

Tiivistelmä

Tässä raportissa esitellään kiinteistöalalle kehitetty lohkoketjusovellus, joka mahdollistaa hajautetun työnkulun hallinnan monivaiheisessa kaupankäyntiprosessissa asunto-osakkeiden vaihdantaan liittyen. Sen keskiössä ovat Ethereum-lohkoketjuun perustuvat älykkäät sopimukset, joiden avulla voidaan koordinoida asuntokaupan prosessiin kytkeytyvien osapuolten vuorovaikutus, sekä hajautettu IPFS-tietokanta, jonka avulla voidaan suorittaa datan yhdistely osaksi kauppaprosessia useista eri tietolähteistä. Sovelluksen kehittämisen tavoitteena on ollut ymmärtää lohkoketjusovellusten kehitysprosessia yhteistyössä teollisuustoimijoiden kanssa. Lisäksi tavoitteena on ollut selvittää, voidaanko älykkäitä sopimuksia hyödyntää kuvatun kaltaisissa sovelluksissa teollisuudessa ja finanssialalla laajemmin. Sovellus ja sen kehityksen aikana käydyt keskustelut osoittavat, että vastaavia hajautettuja prosessirakenteita voi markkinavetoisesti syntyä sellaisiin työprosesseihin, joissa osallistujien lukumäärä on korkea ja joissa prosessin kannalta relevantti informaatio on hajallaan useissa eri tietolähteissä.

Asiasanat: Lohkoketju, sovellus, hajautettu työprosessi, asunto-osake, Ethereum, älykäs sopimus

JEL: L1, L8, L17, L73, L85

1 Introduction

The transaction of a share of stock in a housing company can be a complex process. Many different parties must be involved, several documents need to be collected from various places, approvals must be given in the right order by the right people, and the whole process needs to be accurately documented. Many of these required data reside in various legacy IT systems — or in no IT systems at all but simply on a piece of paper. Therefore, keeping track of all the various steps of the transaction process can be laborious and difficult.

The objective of this study is to determine whether a blockchain-based architecture could be employed to make the transaction process of a property more efficient and more transparent. The idea is that by storing all the relevant data regarding the transaction into a distributed blockchain database, the parties involved can access the transaction data and keep track of the progress almost in real time. This, in turn, would enable the parties autonomously to take action in the correct order of succession and at the right time, thus eliminating needless delays and allowing increased efficiency.

This study approaches the problem with a *product-centric data management* mentality. The idea behind this train of thought is that product and market data should not be asymmetrically fragmented in the market but rather shared in its complete form between all the market participants. Each product individual is represented by one matching information agent in a multi-agent workflow information environment. The agents can be distributed between organizations and do not reside in a single system.[1]

2 From a Use Case to an Application

The motivation for the development of this use case and the following application has been to investigate whether blockchain technology[2] could be used to create a distributed coordination and data management architecture for decentralized workflows in the real estate market, in accordance with the product-centric information management approach.[3] As its core element, the application utilizes Ethereum-based smart contracts to facilitate the first stages of the workflow required by the transaction of a share of stock in a housing company.[4] This facilitation is carried out without any of the participants involved acting as a trusted third party.

This application demonstrates that economic actors could organize their workflows in new ways which go beyond the structures of the current industry. Furthermore, the main purpose of this report is to document and to present the distributed workflow management application along with its source code, including the Ethereum smart contracts written in Solidity programming language.

The co-operation with Euroclear Finland Oy leading to the development of this demo application was initiated in March 2016. Multiple working months of resources were contributed to-

[1] For product-centric information management, see Kärkkäinen *et al.* (2003).

[2] For blockchain technology, see Mattila (2016); and Mattila & Seppälä (2015)

[3] For the conceptualization of the use case, see Mattila *et al.* (2016a).

[4] For smart contracts, see Lauslahti *et al.* (2017).

wards the collaboration from both sides. The use case on which the demo application is based was developed according to the same methods and principles as an earlier use case from September 2016 which examined the applicability of blockchain technology as an architecture solution in distributed energy systems.[5] Respectively, the application developed for the energy use case provided a foundation for the development of the application presented in this report.[6]

This report consists of the following parts. The user guide for running the demo application is found in Appendix 1. The source code of the smart contracts for the workflow architecture is situated in Appendix 2, expressed in Solidity smart contract programming language format. The software code for the status viewer which serves as a graphical user interface for the demo application is found in Appendix 3, expressed in HTML and JavaScript format.

This documentation has been intended for readers with a basic understanding on the Solidity smart contract programming language and on basic web developing tools. In order to run, the demo requires the installation of the software specified in the appendices. Respectively, it is verified that the demo runs properly with these specified versions.

3 Discussion

Traditionally, the workflow structure in digital interactions has been facilitated, and therefore controlled, by the service provider providing the technical architecture for the interaction. Our use case and demo application indicate that similar, more decentralized workflow frameworks can potentially surface in the future. These manifestations can take place at any point in the contemporary value chains and in various established industries and markets. As a result, the value creation and value capturing mechanisms of the contemporary players can become faced with new competition from unexpected directions.

If parties in the economy start transacting via more equilaterally facilitated workflow structures instead of the traditional facilitations by enterprises, this will introduce some regulatory issues. For example, while the facilitating enterprises have thus far acted as natural bottlenecks in the market—as choke points for regulating the entire workflow process—the same kinds of regulatory obligations and responsibilities may be difficult to enforce on decentralized workflow facilitations.

4 Further Research

With the demo application now released, we propose the following steps of research and development. First, we promote the construction of a lab experiment around the demo application, with external information pools and actual transactions of shares of stock in a housing company. Second, we advocate the examination of the technical and legal restrictions for such decentralized workflow structures in the current technological and regulatory environment, *e.g.* the scalability of distributed consensus architectures. Third, we encourage focus group studies

[5] For the methods of this use case development, see Mattila *et al.* (2016b).

[6] For the energy application, see Hukkinen *et al.* (2017).

to determine whether market demand for such distributed architectures could be anticipated to take form in the future.

We hope to see similar initiatives from other research institutions and companies where demo applications are published alongside with the source code and the relevant documentation, for two reasons. Firstly, we believe that sharing findings and results would serve to solidify the on-going discourse on blockchain technology. Secondly, we argue that publishing similar disruptive use cases and application demos is important to fully understand the extent of regulatory reconsiderations required for the effective fostering of innovations in the future.

References

Hukkinen, T., Mattila, J., Ilomäki, J. & Seppälä, T. (2017). A Blockchain Application in Energy. ETLA Reports No 71. <pub.etla.fi/ETLA-Raportit-Reports-71.pdf>

Kärkkäinen, M., Ala-Risku, T. & Främling, K. (2003). The Product Centric Approach. A Solution to a Supply Network Information Management Problems? Computers and Industry 52(3), p. 147–159.

Lauslahti, K., Mattila, J. & Seppälä, T. (2017). Smart Contracts. How will Blockchain Technology Affect Contractual Practices? ETLA Reports No 68. <pub.etla.fi/ETLA-Raportit-Reports-68.pdf>

Mattila, J. (2015). The Blockchain Phenomenon. The Disruptive Potential of Distributed Consensus Architectures. BRIE Working Paper 2016-1. University of California at Berkeley. <brie.berkeley.edu/wp-content/uploads/2015/02/Juri-Mattila-.pdf>

Mattila, J. & Seppälä, T. (2015). Blockchains as a Path to a Network of Systems. ETLA Reports No 45. <http://pub.etla.fi/ETLA-Raportit-Reports-45.pdf>

Mattila, J., Seppälä, T. & Holmström, J. (2016). Product-centric Information Management. A Case Study of a Shared Platform with Blockchain Technology. In Proceedings of the Industry Studies Association Conference 2016. (Mattila *et al.* 2016a). <https://industrystudiesconference.org/conference/papers/download/49>

Mattila, J., Seppälä, T., Naucler, C., Stahl, R., Tikkanen, M., Bådenlid, A. & Seppälä, J. (2016). Industrial Blockchain Platforms. An Exercise in Use Case Development in the Energy Industry. ETLA Working Papers No. 43. (Mattila *et al.* 2016b). <pub.etla.fi/ETLA-Working-Papers-43.pdf>

Appendix 1 – Running the Demo

Prerequisites

This documentation has been intended for readers with a basic understanding on the Solidity smart contract programming language and on basic web developing tools. In order to run, the demo requires the following software to be installed. For verified functionality, the following versions are recommended:

Ubuntu 16.04.2 LTS

TestRPC, version 3.0.4

Truffle, version 3.2.1

Node.js, version 7.9.0

The project repository is released under the MIT License[1] and it can be accessed at *https://github.com/hukkinj1/demo-workflow-management-in-real-estate.*

Running an Ethereum client

At first, an Ethereum client needs to be run:

```
testrpc -d
```

For demoing purposes, TestRPC is a good choice for a client, for a number of reasons. Firstly, TestRPC creates a new blockchain instance and transactions can be paid with tokens of the said blockchain. The creator of the TestRPC session gains access to the tokens for free and therefore transactions can be made without a cost. Secondly, by default, TestRPC is configured in such a way that there is no block time—instead, blocks are created on demand, whenever transactions occur. This type of a configuration is well suited for quick testing and demoing. Finally, TestRPC can be run in deterministic mode. This means that a smart contract's address, for example, can be known already before deploying it in the blockchain. This makes it possible to reference the address in scripts made for testing or demoing purposes.

Deploying the smart contracts

The smart contracts written in Solidity need to be compiled and deployed to the blockchain. This can be achieved by using a development environment for Ethereum called Truffle. A simple migration script needs to be created for Truffle, after which the contracts can be deployed using the following command:

```
truffle migrate
```

[1] <https://opensource.org/licenses/MIT>

Opening the status viewer

Without a graphical user interface, none of the process steps can be visually observed in any way. Therefore, a simple web-browser-based status viewer has been added to the demo application. It shows the changes in the status of the different entities as a crude HTML table. The status viewer can be accessed by opening the web page *index.html* in any web browser.

Creating the assets and the agents, and establishing ownership

For the demo, agents are needed in order to facilitate a workflow between them. Furthermore, for the purposes of facilitating the sale of a share in a housing association, the ownership of assets and documents needs to be assigned to these parties. In our demo, we utilize an approach where a master key holder has the power to establish ownerships to the system participants.

We establish a master key holder that is allowed to create owners for shares of stock, property agents and shares of stock in housing companies in the application. By running the script *0-issuer.js*, we create a number of owners, agents and a number of shares of stock, and we assign the first owner to each created share of stock.

```
node 0-issuer.js
```

```
const Common = require("./common.js");
const web3 = Common.web3;

// Create real estate owners
for (let i = 0; i < Common.accounts.owners.length; i++) {
      Common.market.createOwner(Common.accounts.owners[i].id,
      Common.accounts.owners[i].pubKey, {from: Common.accounts.issuer});
}

// Create agents
for (let i = 0; i < Common.accounts.agents.length; i++) {
      Common.market.createAgent(Common.accounts.agents[i].id,
      Common.accounts.agents[i].pubKey, {from: Common.accounts.issuer});
}

// Create real estates and assign their first owner
for (let i = 0; i < Common.realEstates.length; i++) {
      // Creating multiple transactions in a short period of time seems to cause
      // issues in TestRPC. Let's add a short interval before creating the next
      // transaction for this reason.
      setTimeout(function(){
              Common.market.createRealEstate(Common.realEstates[i].id,
              Common.realEstates[i].owner.id, {from: Common.accounts.issuer});
      }, i*5000);
}
```

The script calls the subscript *common.js* which looks as follows:

```
const Web3 = require('web3');
const web3 = new Web3();
web3.setProvider(new web3.providers.HttpProvider("http://localhost:8545"));

exports.marketAddress = "0xcfeb869f69431e42cdb54a4f4f105c19c080a601";

exports.marketAbi = require('../build/contracts/RealEstateMarket.json').abi;
exports.market = web3.eth.contract(exports.marketAbi).at(exports.marketAddress);
exports.web3 = web3;
exports.accounts = {
      "issuer": web3.eth.accounts[0],

      "owners": [{"id": 2457, "pubKey": web3.eth.accounts[1]}, {"id": 7252,
      "pubKey": web3.eth.accounts[2]}, {"id": 8345, "pubKey": web3.eth.accounts[3]},
      {"id": 1361, "pubKey": web3.eth.accounts[4]}, {"id": 4366, "pubKey":
      web3.eth.accounts[5]}, {"id": 6134, "pubKey": web3.eth.accounts[6]}],

      "agents": [{"id": 6990, "pubKey": web3.eth.accounts[7]}, {"id": 2214,
      "pubKey": web3.eth.accounts[8]}, {"id": 4868, "pubKey": web3.eth.accounts[9]}]
};

exports.realEstates = [{"id": 2645, "owner": exports.accounts.owners[0]},
{"id": 7727, "owner": exports.accounts.owners[1]}, {"id": 3444, "owner":
exports.accounts.owners[2]}];

exports.selectedRealEstate = exports.realEstates[0];
exports.selectedAgent = exports.accounts.agents[0];
```

Initiating the sale of the real estate

The process of selling a share of stock in a housing company usually starts with the seller contacting a property agent or agents for a listing offer. In the case of our demo application, the seller can announce a solicitation for listing offers by initiating a transaction in the smart contract designed to facilitate the workflow.

For the purposes of the demo, the smart contract should therefore be populated with at least one request for listing offers. The script *1-initiateSale.js* is used for this purpose.

```
node 1-initiateSale
```

```
const Common = require("./common.js");
const market = Common.market;

market.initTransaction(Common.selectedRealEstate.id, {from:
Common.selectedRealEstate.owner.pubKey, gas: 300000});
```

Uploading documents to the IPFS

Selling a share of stocks in a housing corporation requires a housing manager's certificate to be drafted.[2] In order to draft the certificate, the building manager must check the validity of the required information by combining data from several public and private information pools, *e.g.* the title and the mortgage register of the National Land Survey, the trade register of the Finnish Patent and Registration Office, the housing company debt report of the creditor banks, the property management planning report of the housing company in question, and so on.

In our demo application's workflow, the data required for the housing manager's certificate are requested from the information pools. The pools directly store the requested files in the Interplanetary File System (IPFS) and enter the associated hash values into the smart contract facilitating the workflow.[3] To emulate this in our demo, the hashes of three random documents followed by the hash of the housing manager's certificate are recorded into the blockchain.

We first run the following command to set up a local IPFS node.

```
ipfs daemon
```

We then open the web user interface at *http://localhost:5001/webui* in browser, drag-and-drop the information pool documents to the web user interface to upload them, and make note of the hash values of the documents.

```
node 2-uploadDocument <document hash>
```

As the next step, the command above is executed three times, each time replacing <document hash> with the hash of a different document. We then run the following command once to upload the actual housing manager's certificate, compiled from the already uploaded documents:

```
node 3-uploadConfirmationLetter <document hash>
```

Create offers from real estate agents to sell the property

When the housing manager's certificate has been received, the real estate can be sold. In the workflow of our demo application, real estate agents compete for who gets to sell the real estate by making offers to the seller of the real estate, specifying a fee (e.g. a percentual cut) that they'll sell it for.[4]

```
node 4-makeAgentOffer <fee>
```

[2] Also sometimes referred to as the building manager's confirmation letter

[3] Storing the files in the IPFS and only the hash values in the smart contract is due to the very good reason that storing the entire documents in the blockchain would be prohibitively costly and inefficient, and therefore completely infeasible. The hash value stored into the blockchain does not give access to the actual document but it can be used to prove that the same exact document has been approved by the identity that is supposed to issue the document.

[4] In reality, other terms and conditions would obviously also apply, but for simplicity's sake, we only apply one variable here, i.e. the fee.

To emulate this market behavior, we run the command above any number of times, each time changing the fee variable to differentiate between offers.

Accepting a real estate agents offer

As the last step of the workflow modelled in our demo application, the seller of the share of stocks in a housing company chooses one of the listing offers made by one of the agents. This is emulated by executing the command below, along with the proper offer ID from the status viewer window.

```
node 5-chooseAgentOffer <offer id>
```

Appendix 2 – Smart Contracts

The logic of the smart contract facilitating the workflow is defined in the Solidity file *RealEstateMarket.sol*. The contract defines the public methods for initiating sales, creating housing manager certificates, as well as creating listing offers and accepting them[5]:

```
pragma solidity ^0.4.8;

contract RealEstateMarket {

    enum TransactionState { NotCreated, WaitingForDocuments,
        WaitingForConfirmationLetter, AcceptingAgentOffers, OnSale, Completed }

    struct Owner {
        address pubKey;
    }

    struct Agent {
        address pubKey;
    }

    struct AgentOffer {
        uint16 fee;
        uint agentId;
    }

    struct Transaction {
        string doc1;
        string doc2;
        string doc3;
        string confirmationLetter;
        TransactionState state;
        mapping (uint => AgentOffer) agentOffers;
        uint chosenAgentOffer;
    }

    struct RealEstate {
        Transaction transaction;
        uint owner;
    }

    modifier transactionInState(uint id, TransactionState state) {
        if (realEstates[id].transaction.state != state) {
            throw;
        }
        _;
    }

    modifier noOngoingTransaction(uint id) {
        if (!(realEstates[id].transaction.state == TransactionState.NotCreated
            || realEstates[id].transaction.state == TransactionState.Completed))
        {
            throw;
        }
        _;
    }
```

[5] The contract also includes a set of non-state-changing getter methods for testing purposes.

```
modifier isOwner(uint id) {
    if (owners[realEstates[id].owner].pubKey != msg.sender) {
        throw;
    }
    _;
}

modifier isAgent(uint id) {
    if (agents[id].pubKey != msg.sender) {
        throw;
    }
    _;
}

modifier issuerOnly() {
    if (issuer != msg.sender) {
        throw;
    }
    _;
}

modifier offerExists(uint offerId, uint realEstateId) {
    if (realEstates[realEstateId].transaction.agentOffers[offerId].agentId
          == 0) {
        throw;
    }
    _;
}

// Mapping from real estate id to real estate object
mapping (uint => RealEstate) realEstates;
// Mapping from owner ID to owner object
mapping (uint => Owner) owners;
// Mapping from agent ID to agent object
mapping (uint => Agent) agents;

address issuer;

event LogCreateRealEstate(uint realEstateId, uint ownerId);
event LogInitTransaction(uint realEstateId, uint ownerId);
event LogOneDocumentUploaded(uint realEstateId, string document1);
event LogTwoDocumentsUploaded(uint realEstateId, string document2);
event LogAllDocumentsUploaded(uint realEstateId, string document3);
event LogConfirmationLetterUploaded(uint realEstateId, string confirmationLetter);
event LogAgentOffer(uint realEstateId, uint offerId, uint agentId, uint16 fee);
event LogAgentOfferChosen(uint realEstateId, uint offerId);

function RealEstateMarket() {
    issuer = msg.sender;
}

function createOwner(uint id, address pubKey)
    issuerOnly()
{
    owners[id].pubKey = pubKey;
}

function createRealEstate(uint id, uint owner)
    issuerOnly()
{
    realEstates[id].owner = owner;
    LogCreateRealEstate(id, owner);
}
```

```
function createAgent(uint id, address pubKey)
    issuerOnly()
{
    agents[id].pubKey = pubKey;
}

function initTransaction(uint id)
    isOwner(id)
    noOngoingTransaction(id)
{
    realEstates[id].transaction.state = TransactionState.WaitingForDocuments;
    LogInitTransaction(id, realEstates[id].owner);
}

function uploadDocument(uint id, string docHash)
    transactionInState(id, TransactionState.WaitingForDocuments)
{
    if (bytes(realEstates[id].transaction.doc1).length == 0) {
        realEstates[id].transaction.doc1 = docHash;
        LogOneDocumentUploaded(id, docHash);
    }
    else if (bytes(realEstates[id].transaction.doc2).length == 0) {
        realEstates[id].transaction.doc2 = docHash;
        LogTwoDocumentsUploaded(id, docHash);
    }
    else if (bytes(realEstates[id].transaction.doc3).length == 0) {
        realEstates[id].transaction.doc3 = docHash;
        nextTransactionState(id);
        LogAllDocumentsUploaded(id, docHash);
    }
}

function uploadConfirmationLetter(uint id, string hash)
    transactionInState(id, TransactionState.WaitingForConfirmationLetter)
{
    realEstates[id].transaction.confirmationLetter = hash;
    nextTransactionState(id);
    LogConfirmationLetterUploaded(id, hash);
}

function makeAgentOffer(uint offerId, uint realEstateId, uint agentId, uint16 fee)
    transactionInState(realEstateId, TransactionState.AcceptingAgentOffers)
    isAgent(agentId)
{
    realEstates[realEstateId].transaction.agentOffers[offerId].fee = fee;
    realEstates[realEstateId].transaction.agentOffers[offerId].agentId = agentId;
    LogAgentOffer(realEstateId, offerId, agentId, fee);
}

function chooseAgentOffer(uint offerId, uint realEstateId)
    transactionInState(realEstateId, TransactionState.AcceptingAgentOffers)
    isOwner(realEstateId)
    offerExists(offerId, realEstateId)
{
    realEstates[realEstateId].transaction.chosenAgentOffer = offerId;
    nextTransactionState(realEstateId);
    LogAgentOfferChosen(realEstateId, offerId);
}
```

```
    function nextTransactionState(uint realEstateId) private {
        uint currentState = uint(realEstates[realEstateId].transaction.state);
        realEstates[realEstateId].transaction.state
              = TransactionState(currentState + 1);
    }

    function getTransactionState(uint realEstateId) constant returns
      (TransactionState) {
        return realEstates[realEstateId].transaction.state;
    }

    function getOfferFee(uint realEstateId, uint offerId) constant returns (uint16) {
        return realEstates[realEstateId].transaction.agentOffers[offerId].fee;
    }

    function getDocumentHash(uint realEstateId, uint8 index) constant returns (string)
{
        if (index == 1) {
            return realEstates[realEstateId].transaction.doc1;
        }
        else if (index == 2) {
            return realEstates[realEstateId].transaction.doc2;
        }
        else if (index == 3) {
            return realEstates[realEstateId].transaction.doc3;
        }
        throw;
    }
}
```

Appendix 3 – The Status Viewer (GUI)

The status viewer is a web page which is useful for observing changes in the blockchain while running the demo. It shows the status of the workflow process, with all the contributions to it by the various participants. The status viewer can be run by opening the file *index.html* in any web browser.

```
<!doctype html>

<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Real Estate Market</title>

  <link rel="stylesheet" href="bower_components/bootstrap/dist/css/bootstrap.min.css">
  <link rel="stylesheet" href="bower_components/dynatable/jquery.dynatable.css">
  <link rel="stylesheet" href="status.css">
</head>

<body>
  <div>
    <h3>Transactions</h3>
    <table id="transactions" class="table table-bordered">
      <thead>
        <th>Real Estate ID</th>
        <th>Owner ID</th>
        <th>State</th>
        <th>Document One</th>
        <th>Document Two</th>
        <th>Document Three</th>
        <th>Confirmation letter</th>
        <th>Agent offers</th>
        <th>Chosen offer</th>
      </thead>
      <tbody>
      </tbody>
    </table>
  </div>

  <!--
  <div>
    <h3>Balances</h3>
    <table id="balances" class="table table-bordered">
      <thead>
        <th>Identity</th>
        <th>Account</th>
        <th>Balance</th>
      </thead>
      <tbody>
      </tbody>
    </table>
  </div>
  -->

  <script src="bower_components/jquery/dist/jquery.min.js"></script>
  <script src="bower_components/bootstrap/dist/js/bootstrap.min.js"></script>
  <script src="bower_components/dynatable/jquery.dynatable.js"></script>
  <script src="bower_components/web3/dist/web3.min.js"></script>
  <script src="bower_components/moment/min/moment.min.js"></script>
  <script src="market-address-and-abi.js"></script>
  <script src="status.js"></script>
</body>
</html>
```

The HTML-webpage calls for the script *status.js* which looks as follows[6]:

```
if (typeof web3 !== 'undefined') {
    web3 = new Web3(web3.currentProvider);
} else {
    // set the provider you want from Web3.providers
    web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
}

let transactions = [];
var dynatable = $('#transactions').dynatable().data('dynatable');
market = web3.eth.contract(marketAbi).at(marketAddress);

market.LogCreateRealEstate().watch(function(error, result) {
    if (!error) {
        let newOffer = result.args;
        newOffer.state = "No ongoing transaction"
        console.log(newOffer);
        transactions.push(newOffer);
        updateDynatable(dynatable, transactions);
    } else {console.log("error");}
});

market.LogInitTransaction().watch(function(error, result) {
    if (!error){
        let newOffer = result.args;
        let i = transactions.findIndex((obj
              => obj.realEstateId.equals(newOffer.realEstateId)));
        if (i !== -1) {
            transactions[i].state = "1/4 Waiting for documents";
            updateDynatable(dynatable, transactions);
        }
    }
});

market.LogOneDocumentUploaded().watch(function(error, result) {
    if (!error){
        let newOffer = result.args;
        let i = transactions.findIndex((obj
              => obj.realEstateId.equals(newOffer.realEstateId)));
        if (i !== -1) {
            transactions[i].documentOne = makeIpfsLink(newOffer.document1);
            updateDynatable(dynatable, transactions);
        }
    }
});

market.LogTwoDocumentsUploaded().watch(function(error, result) {
    if (!error){
        let newOffer = result.args;
        let i = transactions.findIndex((obj
              => obj.realEstateId.equals(newOffer.realEstateId)));
        if (i !== -1) {
            transactions[i].documentTwo = makeIpfsLink(newOffer.document2);
            updateDynatable(dynatable, transactions);
        }
    }
});
```

[6] The HTML-page also utilizes some JavaScript-libraries which can be installed using the Bower package manager.

```
market.LogAllDocumentsUploaded().watch(function(error, result) {
    if (!error){
        let newOffer = result.args;
        let i = transactions.findIndex((obj
              => obj.realEstateId.equals(newOffer.realEstateId)));
        if (i !== -1) {
            transactions[i].documentThree = makeIpfsLink(newOffer.document3);
            transactions[i].state = "2/4 Waiting for confirmation letter";
            updateDynatable(dynatable, transactions);
        }
    }
});

market.LogConfirmationLetterUploaded().watch(function(error, result) {
    if (!error){
        let newOffer = result.args;
        let i = transactions.findIndex((obj
              => obj.realEstateId.equals(newOffer.realEstateId)));
        if (i !== -1) {
            transactions[i].confirmationLetter
              = makeIpfsLink(newOffer.confirmationLetter);
            transactions[i].state = "3/4 Waiting for agent offers";
            updateDynatable(dynatable, transactions);
        }
    }
});

market.LogAgentOffer().watch(function(error, result) {
    if (!error){
        let newOffer = result.args;
        let i = transactions.findIndex((obj
              => obj.realEstateId.equals(newOffer.realEstateId)));
        if (i !== -1) {
            if (typeof transactions[i].agentOffers === 'undefined') {
                transactions[i].agentOffers = "";
            }
            transactions[i].agentOffers = transactions[i].agentOffers
              + JSON.stringify({"Offer Id": newOffer.offerId, "Agent Id":
              newOffer.agentId, "Fee": newOffer.fee});
            updateDynatable(dynatable, transactions);
        }
    }
});

market.LogAgentOfferChosen().watch(function(error, result) {
    if (!error){
        let newOffer = result.args;
        let i = transactions.findIndex((obj
              => obj.realEstateId.equals(newOffer.realEstateId)));
        if (i !== -1) {
            transactions[i].chosenOffer = newOffer.offerId;
            transactions[i].state = "4/4 On sale";
            updateDynatable(dynatable, transactions);
        }
    }
});
```

```
// Create a table of account balances and update it on a set interval
// var balancesTable = $('#balances').dynatable().data('dynatable');
// setInterval(function() {
//     let balances = [{"identity": "Smart contract", "account": marketAddress,
//     "balance": web3.eth.getBalance(marketAddress)},
//         {"identity": "Seller", "account": web3.eth.accounts[2],
//     "balance": web3.eth.getBalance(web3.eth.accounts[2])},
//         {"identity": "Buyer", "account": web3.eth.accounts[4],
//     "balance": web3.eth.getBalance(web3.eth.accounts[4])}];
//     updateDynatable(balancesTable, balances);
// }, 5000);

function updateDynatable(table, content) {
    table.settings.dataset.originalRecords = content;
    table.process();
}

function makeIpfsLink(hash) {
    const visibleText = hash;
    return '<a href="https://ipfs.io/ipfs/' + hash + '">' + visibleText + '</a>';
}
```

Aikaisemmin ilmestynyt ETLA Raportit-sarjassa (ennen ETLA Keskusteluaiheita)

Previously published in the ETLA Reports series (formerly ETLA Discussion Papers)

No 62 *Jyrki Ali-Yrkkö – Petri Rouvinen – Pekka Sinko – Joonas Tuhkuri*, Suomi globaaleissa arvoketjuissa. 30.11.2016. 41 s.

No 63 *Joona Widgrén*, Google-haut Suomen asuntojen hintojen ennustajana. 14.12.2016. 37 s.

No 64 *Rita Asplund – Antti Kauhanen – Pekka Vanhala*, Työpankin kautta työllistyminen. 20.12.2016. 19 s.

No 65 *Annu Kotiranta – Mika Pajarinen – Petri Rouvinen*, Alkuvaiheen koko, osakeyhtiömuoto ja kasvuhakuisuus selittävät nuorten yritysten toteutunutta kasvua. 22.12.2016. 12 s.

No 66 *Annu Kotiranta – Mika Pajarinen – Petri Rouvinen*, Miltä startupit näyttävät tilastojen valossa? 22.12.2016. 17 s.

No 67 *Annu Kotiranta – Mika Pajarinen – Petri Rouvinen*, Onko uusyrittäjyyden luonne muuttunut? 22.12.2016. 47 s.

No 68 *Kristian Lauslahti – Juri Mattila – Timo Seppälä*, Smart Contracts – How will Blockchain Technology Affect Contractual Practices? 9.1.2017. 27 s.

No 69 *Jyrki Ali-Yrkkö – Juri Mattila – Timo Seppälä*, Estonia in Global Value Chains. 11.1.2017. 24 s.

No 70 *Jyrki Ali-Yrkkö – Tero Kuusi – Mika Maliranta*, Miksi yritysten investoinnit ovat vähentyneet?. 16.2.2017. 73 s.

No 71 *Taneli Hukkinen – Juri Mattila – Juuso Ilomäki – Timo Seppälä*, A Blockchain Application in Energy. 3.5.2017. 22 s.

No 72 *Mika Maliranta – Nelli Valmari*, Suomen teollisuustuotannon uudistuminen tuotantolinjatasolla. 15.6.2017. 18 s.

No 73 *Mika Maliranta – Roope Ohlsbom*, Suomen tehdasteollisuuden johtamiskäytäntöjen laatu. 27.9.2017. 30 s.

No 74 *Annu Kotiranta – Timo Seppälä – Antti-Jussi Tahvanainen – Markus Hemminki – Juri Mattila – Samuli Sadeoja – Tea Tähtinen*, Roadmap for Renewal: A Shared Platform in the Food Industry. 2.10.2017. 51 s.

No 76 *Jyrki Ali-Yrkkö – Tero Kuusi*, Shield the US from Imports! GDP Impacts on Finland and Other European Union Member States. 4.10.2017. 24 s.

No 77 *Mika Pajarinen – Petri Rouvinen – Ilkka Ylhäinen*, Tuottavuuskehityksen eriytyminen: Karkaavatko eturintaman yritykset muilta? 13.10.2017. 12 s.

Sarjan julkaisut ovat raportteja tutkimustuloksista ja väliraportteja tekeillä olevista tutkimuksista.

Julkaisut ovat ladattavissa pdf-muodossa osoitteessa: www.etla.fi » julkaisut » raportit

Papers in this series are reports on research results and on studies in progress.

Publications in pdf can be downloaded at www.etla.fi » publications » reports

ETLA
Elinkeinoelämän tutkimuslaitos
The Research Institute of the Finnish Economy
Arkadiankatu 23 B
00100 Helsinki

Puh. 09-609 900
www.etla.fi
etunimi.sukunimi@etla.fi

ISSN-L 2323-2447, ISSN 2323-2447, ISSN 2323-2455 (Pdf)