Hukkinen, Taneli; Mattila, Juri; Ilomäki, Juuso; Seppälä, Timo

Research Report

A Blockchain Application in Energy

ETLA Report, No. 71

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

Suggested Citation: Hukkinen, Taneli; Mattila, Juri; Ilomäki, Juuso; Seppälä, Timo (2017) : A Blockchain Application in Energy, ETLA Report, No. 71, The Research Institute of the Finnish Economy (ETLA), Helsinki

This Version is available at:
https://hdl.handle.net/10419/201353

ETLA ELINKEINOELÄMÄN TUTKIMUSLAITOS
THE RESEARCH INSTITUTE OF THE FINNISH ECONOMY

ETLA Raportit
ETLA Reports

3 May 2017

No 71

A Blockchain Application in Energy

Taneli Hukkinen* – Juri Mattila** – Juuso Ilomäki* – 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.fi

Suggested citation: Hukkinen, Taneli, Mattila, Juri, Ilomäki, Juuso & Seppälä, Timo (3.5.2017). "A Blockchain Application in Energy". ETLA Reports No 71. https://pub.etla.fi/ETLA-Raportit-Reports-71.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 Catarina Naucler, Tobias Goodden, Riitta Stahl, and Heli Antila from Fortum Oyj, and Niclas Unnervik from Netlight Consulting Ab.

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

Table of Contents

A Blockchain Application in Energy

Abstract

This report documents a blockchain application developed for the energy sector that enables distributed market coordination for decentralized energy systems. As its core element, it utilizes Ethereum-based smart contracts to facilitate market matching between individual producers and consumers of electricity. The motive for this application was to understand the process of developing blockchain applications with industrial partners. Moreover, the purpose of this exercise was to examine whether Ethereum-based smart contracts could be effectively utilized for similar applications in industry and society at large. The application and the discussions during its development indicate that similar horizontal market structures may spring up in value chains in which the dynamicity of the market is growing and in which the roles of the market actors are shifting from fixed roles towards switch-role markets.

Key words: Blockchain, application, distributed marketplace, energy industry, Ethereum, smart contract

JEL: L1, L17, L52, L73, L94

Hajautettu markkinapaikka lohkoketjusovelluksena

Tiivistelmä

Tässä raportissa esitellään energiasektorille kehitetty lohkoketjusovellus, joka mahdollistaa hajautetun markkinakoordinaation toteuttamisen vaihtoehtoisille energiajärjestelmille. Sen keskiössä ovat Ethereum-lohkoketjuun perustuvat älykkäät sopimukset, joiden avulla yksittäiset toimijat voivat ostaa ja myydä sähköä ilman nykymuotoista keskitettyä markkinamekanismia. 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 yhteiskunnassa laajemmin. Sovellus ja sen kehityksen aikana käydyt keskustelut osoittavat, että vastaavia horisontaalisia markkinarakenteita voi syntyä sellaisiin arvoketjuihin, joissa markkinoiden dynaamisuus kasvaa ja toimijoiden roolit muuttuvat monisuuntaisiksi.

Asiasanat: Lohkoketju, sovellus, markkinapaikka, energia, Ethereum, älykäs sopimus

JEL: L1, L17, L52, L73, L94

1 Introduction

With digitalization, the ability of consumers to actively participate in both sides of the market is increasing. In the energy sector especially, this trend has been widely acknowledged in earlier research and in industry at large.[1]

The current mass production energy infrastructure is largely built on the assumption that access to information regarding the origin, allocation and utilization of energy is expensive. Over the past years, however, the trend towards more decentralized energy solutions has grown stronger, and gathering sensor data has become more affordable. For these reasons, the underlying assumption of the current energy system design has become increasingly questionable. This calls for the development for a new energy system paradigm.[2]

One of the big questions in regards to decentralized energy systems is whether their coordination and data management should be handled in a centralized or a decentralized architecture. While more centralized solutions are certainly more easily feasible, they involve certain short-comings in a truly decentralized energy system.

For example, in centralized solutions, the market coordinators most often produce, store and offer data on the basis of what is relevant from their market perspective. In many cases, however, the data collected or the data that that *could* be collected would be highly valuable to another party from the one who collects it. Due to the information asymmetries involved and the lack of mutual incentives, however, market inefficiencies are likely to occur in market coordination and data utilization.

To address these kinds of problems, the *product-centric data management* approach was developed. The idea behind it is that product and market data is not asymmetrically fragmented in the market but rather shared in its complete form between the market participants. Each product individual is represented by one matching information agent over its entire lifecycle. The agents can be distributed between organizations and do not reside in a single system.[3]

2 From a Use Case to an Application

The motive for the development of this use case and this application has been to examine whether blockchain technology could be utilized to create a distributed coordination and data management architecture for decentralized energy systems, in accordance with the product-centric information management mentality. As its core element, it utilizes Ethereum-based smart contracts to facilitate market matching between individual producers and consumers looking to buy and sell electricity. This facilitation is carried out without the involvement of the existing market operators acting as trusted third parties.

This application demonstrates that producers and consumers could organize themselves in new ways which go beyond the current industrial and societal structures. Furthermore, the

---

1 In the energy industry, phenomenon is referred to as the convergence of the consumer and the producer, or the prosumer trend. (Karnouskos, 2011). In a wider context, economic sociology uses the terms fixed-role and switch-role markets (Aspers, 2007).

2 For further information, see Katz *et al.* (2011).

3 For further information, see Kärkkäinen *et al.* (2003).

main purpose of this report is to document and to present our distributed energy sector marketplace application and its source code, including the Ethereum smart contracts written in Solidity programming language.

The co-operation with Fortum Oyj leading to the development of this demo application was initiated in March 2016.[4] Multiple working months of resources were contributed towards the collaboration from both sides. The demo application is based on an earlier use case published in September 2016 in which we examined the applicability of blockchain technology as an architecture solution for distributed energy systems.[5]

The demo application presented in 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 distributed marketplace are 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.

3 Discussion

Drawing macro scale conclusions from any given phenomenon first requires understanding it on the micro level. Our use case and demo application indicate that similar horizontal market manifestations 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 unconventional directions.

If new agents, such as housing co-operatives and private individuals, enter the electricity market in roles currently mostly reserved for large-scale enterprises, this will also introduce some regulatory issues. For example, while the large-scale enterprises have thus far acted as natural bottlenecks in the market—as choke points for regulating the entire value chain of energy—the same kinds of regulatory obligations and responsibilities cannot possibly be targeted at housing co-operatives and private individuals.

In recent academic discussions, legislation has been recognized as a key factor in fostering innovations.[6] The legislator must be able to identify and to respond to new technological disruptions or otherwise society runs a risk of thwarting new innovations. For this reason, it is important for the legislator to engage in an active discourse with technology providers and innovators. Our use case and demo application also serve to demonstrate that the field of technological development is becoming more widespread; It is no longer enough to engage only large-scale enterprises but the role of spontaneous innovation ecosystems must also be recognized in legislative discussions.

[4] In addition to the use case and the demo application presented herein, we have also been engaged in use case and application development in other industrial areas, such as digital asset management. The release of a similar demo application for another use case is scheduled in August 2017.

[5] For more information on use case, see Mattila *et al.* (2016).

[6] Chander (2014).

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 physical devices and actual transactions of electricity. Second, we advocate the examination of the technical and legal restrictions for such distributed marketplaces in the current technological and regulatory environment, *e.g.* the scalability of distributed consensus architectures. Third, we encourage focus group studies to determine whether market demand for such distributed architectures could be seen to actualize 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 effective fostering of innovations in the future.

References

Aspers, P. (2007). Theory, Reality, and Performativity in Markets. American Journal of Economics and Sociology 66(2), p. 379–398.

Chander, A. (2014). How Law Made Silicon Valley. Emory Law Journal 63(3), p. 39–694.

Karnouskos, S. (2011). Demand Side Management via Prosumer Interactions in a Smart City Energy Marketplace. Proceeding from the 2nd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies.

Katz, R., Culler, D., Sanders, S., Alspaugh, S., Chen, Y., Dawson-Haggerty, S., Dutta, P., He, M., Jiang, X., Keys, L., Krioukov, A., Lutz, K., Ortiz, J., Mohan, P., Reutzel, E., Taneja, J., Hsu, J. & Shankar, S. (2011). An Information-centric Energy Infrastructure: The Berkeley View. Sustainable Computing: Informatics and Systems 1(2011), p. 7–22.

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.

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 Paper No. 43.

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 specified 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[7] and it can be accessed at *https://github.com/hukkinj1/demo-marketplace-in-energy.*

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
```

[7] <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.

Establishing ownership for the smart meters

For the demo, a seller and a buyer of electricity are needed. Furthermore, the ownership of a smart meter needs to be assigned to both of 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 of smart meters for the electricity marketplace and have the key holder assign ownership to the smart meters.

```
node issuer.js
```

```
const Constants = require("./constants.js");

const web3 = Constants.web3;

const electricityMarket = Constants.market;

const smartMetersAddress = electricityMarket.getSmartMetersContract();
const smartMeters = web3.eth.contract(Constants.smartMetersAbi).
at(smartMetersAddress);

smartMeters.setIssuer(Constants.accounts.issuer, {from:
Constants.accounts.issuer});

smartMeters.changeOwner(Constants.accounts.sellerSmartMeter, Constants.
accounts.seller, {from: Constants.accounts.issuer});

smartMeters.changeOwner(Constants.accounts.buyerSmartMeter, Constants.
accounts.buyer, {from: Constants.accounts.issuer});
```

The script calls the subscript *constants.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/ElectricityMarket.json').abi;
exports.smartMetersAbi =
require('../build/contracts/SmartMeters.json').abi;
exports.market =
web3.eth.contract(exports.marketAbi).at(exports.marketAddress);
exports.web3 = web3;
exports.accounts = {
  "issuer": web3.eth.accounts[0],
  "sellerSmartMeter": web3.eth.accounts[1],
  "seller": web3.eth.accounts[2],
  "buyerSmartMeter": web3.eth.accounts[3],
  "buyer": web3.eth.accounts[4]
};
```

Creating sell offers

For the purposes of the demo, the market should be populated with sell offers of energy. The script *offer.js* is used for this purpose. The script's optional command line parameter *index* can be used to create different pre-populated sell offers. The index must be within range 0 to 9. When omitting the *index* parameter, the script defaults to using index 0.

```
node offer [<index>]
```

```
const Constants = require("./constants.js");

const market = Constants.market;

let offers = require("./offer-objects.js");

let offerIndex = parseInt(process.argv[2]);
if (isNaN(offerIndex)) {
    offerIndex = 0;
}

const offer = offers[offerIndex];

market.makeOffer(offer.id, offer.price, offer.electricityAmount,
    offer.startTime, offer.endTime, offer.sellerSmartMeter,
    {from: Constants.accounts.seller, gas: 300000});
```

The pre-populated offers are defined and can be edited in *offer-objects.js* which looks as follows:

```
const Constants = require("./constants.js");

module.exports = [{
    "id": 1349,
    "startTime": 1800011110,
    "endTime": 1800011111,
    "price": 1200000000,
    "electricityAmount": 110,
    "sellerSmartMeter": Constants.accounts.sellerSmartMeter,
    "buyerSmartMeter": Constants.accounts.buyerSmartMeter
},{
    "id": 1350,
    "startTime": 1800011115,
    "endTime": 1800011120,
    "price": 1200000000,
    "electricityAmount": 110,
    "sellerSmartMeter": Constants.accounts.sellerSmartMeter,
    "buyerSmartMeter": Constants.accounts.buyerSmartMeter
},{
    "id": 1351,
    "startTime": 1800011125,
    "endTime": 1800011130,
    "price": 1200000000,
    "electricityAmount": 110,
    "sellerSmartMeter": Constants.accounts.sellerSmartMeter,
    "buyerSmartMeter": Constants.accounts.buyerSmartMeter
},{
    "id": 1352,
    "startTime": 1800011200,
    "endTime": 1800011300,
    "price": 1200000000,
    "electricityAmount": 110,
    "sellerSmartMeter": Constants.accounts.sellerSmartMeter,
    "buyerSmartMeter": Constants.accounts.buyerSmartMeter
},{
    "id": 1353,
    "startTime": 1800011301,
    "endTime": 1800011324,
    "price": 1200000000,
    "electricityAmount": 110,
    "sellerSmartMeter": Constants.accounts.sellerSmartMeter,
    "buyerSmartMeter": Constants.accounts.buyerSmartMeter
}];
```

Accepting offers as a buyer

Any sell offers created earlier can be accepted by the buyer. The script *acceptoffer.js* can be used to do so. The optional *index* parameter of the script works similarly as the parameter earlier specified for the *offer.js* script and can be used to accept offers. For example, if an offer was created using the command *node offer 4*, then running the command *node acceptoffer 4* will make the buyer accept it.[8]

```
node acceptoffer [<index>]
```

```
const Constants = require("./constants.js");

const market = Constants.market;

let offers = require("./offer-objects.js");

let offerIndex = parseInt(process.argv[2]);
if (isNaN(offerIndex)) {
    offerIndex = 0;
}

const offer = offers[offerIndex];

market.acceptOffer(offer.id, offer.buyerSmartMeter,
    {from: Constants.accounts.buyer, gas: 300000, value: offer.price});
```

Sending reports from smart meters

Once an offer has been created and accepted, and the scheduled transfer of electricity is due, both the seller's and the buyer's smart meters are expected to report on the successfulness of the transfer. The Ethereum transactions submitting these reports can be created using the scripts *sellerreport.js* and *buyerreport.js*. The optional *index* parameter can be used to refer to different sell offers exactly the same way as in the scripts described earlier. Both scripts, *sellerreport.js* and *buyerreport.js* need to be run in order to move an instance of electricity transfer to its next state in the smart contract.

The reports have a deadline before which they need to be submitted. According to the deadline, the reports must be submitted and written into the blockchain no later than 30 minutes after the transfer is completed. In the case a transacting party fails to report within the allocated time frame, the smart contract will assume the worst possible economic outcome for the abstinent party.

[8] When accepting an offer, the buyer must deposit payment for the energy into the smart contract. The buyer's address is thus required to hold enough Ether tokens for this step to be successful.

node sellerreport [<index>]

```
const Constants = require("./constants.js");

const market = Constants.market;

let offers = require("./offer-objects.js");

let offerIndex = parseInt(process.argv[2]);
if (isNaN(offerIndex)) {
    offerIndex = 0;
}

const offer = offers[offerIndex];

market.sellerReport(offer.id, true, {from:
Constants.accounts.sellerSmartMeter, gas: 300000});
```

node buyerreport [<index>]

```
const Constants = require("./constants.js");

const market = Constants.market;

let offers = require("./offer-objects.js");

let offerIndex = parseInt(process.argv[2]);
if (isNaN(offerIndex)) {
    offerIndex = 0;
}

const offer = offers[offerIndex];

market.buyerReport(offer.id, true, {from:
Constants.accounts.buyerSmartMeter, gas: 300000});
```

Making a withdrawal

Once both smart meters have submitted their reports (or the deadline has expired), the assets stored in the contract can be withdrawn. The withdrawal can be initiated by anyone, in which case the assets are sent to their rightful owner, as determined by the logic of the smart contract. The script *withdraw.js* can be used to execute the withdrawal. The optional *index* parameter can be used to refer to different instances of electricity transfer.

```
node withdraw [<index>]
```

```
const Constants = require("./constants.js");

const market = Constants.market;

let offers = require("./offer-objects.js");

let offerIndex = parseInt(process.argv[2]);
if (isNaN(offerIndex)) {
    offerIndex = 0;
}

const offer = offers[offerIndex];

market.withdraw(offer.id, {from: Constants.accounts.issuer, gas:
300000});
```

Appendix 2 – Smart Contracts

The logic of the electricity market smart contract is defined in the Solidity file *ElectricityMarket.sol*. The contract defines the public methods for creating sell offers, accepting them, sending smart meter reports and withdrawing assets[9]:

```
pragma solidity ^0.4.8;

import "./SmartMeters.sol";

contract ElectricityMarket {

    enum ContractState { NotCreated, Created, Accepted,
WaitingForBuyerReport, WaitingForSellerReport, ReadyForWithdrawal,
Resolved, TimedOut }

    struct Contract {
        address seller;
        address sellerSmartMeter;
        address buyer;
        address buyerSmartMeter;
        uint price;
        uint electricityAmount;
        uint startTime;
        uint endTime;
        bool sellerReport;  // true if everything went OK
        bool buyerReport;  // true if everything went OK
        ContractState state;
    }

    modifier contractNotCreated(uint id) {
        if (contracts[id].state != ContractState.NotCreated) {
            throw;
        }
        _;
    }

    modifier contractInState(uint id, ContractState state) {
        if (contracts[id].state != state) {
            throw;
        }
        _;
    }

    modifier waitingForSellerReport(uint id) {
        if ((contracts[id].state != ContractState.Accepted) &&
(contracts[id].state != ContractState.WaitingForSellerReport)) {
            throw;
        }
        _;
    }

    modifier waitingForBuyerReport(uint id) {
        if ((contracts[id].state != ContractState.Accepted) &&
(contracts[id].state != ContractState.WaitingForBuyerReport)) {
            throw;
        }
        _;
    }
```

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

```
    modifier noOverlappingContracts(address smartMeter, uint startTime,
uint endTime) {
        for (uint i = 0; i < contractsBySmartMeter[smartMeter].length;
i++) {
            Contract c = contracts[contractsBySmartMeter[smartMeter]
[i]];
            if (doTimeslotsOverlap(startTime, endTime, c.startTime,
c.endTime)) {
                throw;
            }
        }
        _;
    }

    modifier ownsSmartMeter(address owner, address smartMeter) {
        if (owner != smartMeters.owner(smartMeter)) {
            throw;
        }
        _;
    }

    modifier buyerSmartMeterOnly(uint id) {
        if (msg.sender != contracts[id].buyerSmartMeter) {
            throw;
        }
        _;
    }

    modifier sellerSmartMeterOnly(uint id) {
        if (msg.sender != contracts[id].sellerSmartMeter) {
            throw;
        }
        _;
    }

    modifier condition(bool c) {
        if (!c) {
            throw;
        }
        _;
    }

    modifier costs(uint price) {
        if (msg.value != price) {
            throw;
        }
        _;
    }

    mapping (uint => Contract) contracts;
    mapping (address => uint[]) contractsBySmartMeter;
    SmartMeters smartMeters;

    // Minimum time from block.timestamp to startTime. The time needs to
    // be long enough, so that a few blocks are created in between, so
    // that the smart meters can be sure that the transmission is
    // approved by the blockchain
    uint minTimeFromAcceptedToStart = 60;  // 1 minute
    // Maximum time from endTime to smart meters reporting about how the
    // transmission went.
    uint maxTimeFromEndToReportDeadline = 1800;  // 30 minutes
    event LogOffer(address seller, uint id, uint price, uint
```

```
electricityAmount, uint startTime, uint endTime, address
sellerSmartMeter);
    event LogAcceptOffer(address seller, uint id, uint price, uint
electricityAmount, uint startTime, uint endTime, address
sellerSmartMeter, address buyer, address buyerSmartMeter);
    event LogResolved(uint id, address seller, address buyer, address
recipient);

    function ElectricityMarket() {
        smartMeters = new SmartMeters();
    }

    function makeOffer(uint id, uint price, uint electricityAmount, uint
startTime, uint endTime, address sellerSmartMeter)
        contractNotCreated(id)
        condition(startTime < endTime)
        noOverlappingContracts(sellerSmartMeter, startTime, endTime)
        ownsSmartMeter(msg.sender, sellerSmartMeter)
    {
        storeAndLogNewOffer(id, price, electricityAmount, startTime,
endTime, sellerSmartMeter);
    }

    function acceptOffer(uint id, address buyerSmartMeter) payable
        costs(contracts[id].price)
        contractInState(id, ContractState.Created)
        ownsSmartMeter(msg.sender, buyerSmartMeter)
    {
        // Check if contract timed out
        if ((now + minTimeFromAcceptedToStart) > contracts[id].
startTime) {
            contracts[id].state = ContractState.TimedOut;
            return;
        }

        contracts[id].buyer = msg.sender;
        contracts[id].buyerSmartMeter = buyerSmartMeter;
        contracts[id].state = ContractState.Accepted;

        LogAcceptOffer(contracts[id].seller, id, contracts[id].price,
contracts[id].electricityAmount, contracts[id].startTime, contracts[id].
endTime, contracts[id].sellerSmartMeter, msg.sender, buyerSmartMeter);
    }

    function sellerReport(uint id, bool report)
        sellerSmartMeterOnly(id)
        waitingForSellerReport(id)
    {
        if (hasReportDeadlineExpired(id)) {
            contracts[id].state = ContractState.ReadyForWithdrawal;
            return;
        }
        contracts[id].sellerReport = report;
        contracts[id].state = (contracts[id].state ==
ContractState.Accepted) ? ContractState.WaitingForBuyerReport :
ContractState.ReadyForWithdrawal;
    }

    function buyerReport(uint id, bool report)
        buyerSmartMeterOnly(id)
        waitingForBuyerReport(id)
```

```
    {
        if (hasReportDeadlineExpired(id)) {
            contracts[id].state = ContractState.ReadyForWithdrawal;
            return;
        }
        contracts[id].buyerReport = report;
        contracts[id].state = (contracts[id].state ==
ContractState.Accepted) ? ContractState.WaitingForSellerReport :
ContractState.ReadyForWithdrawal;
    }

    function withdraw(uint id)
    {
        if (contracts[id].state != ContractState.ReadyForWithdrawal) {
            makeReadyForWithdrawal(id);
        }
        contracts[id].state = ContractState.Resolved;

        address recipient;

        if (!contracts[id].sellerReport) {
            recipient = contracts[id].buyer;
        }
        else if (contracts[id].buyerReport) {
            recipient = contracts[id].seller;
        }
        else {
            recipient = this;
        }

        LogResolved(id, contracts[id].seller, contracts[id].buyer,
recipient);

        if (recipient != address(this)) {
            if (!recipient.send(contracts[id].price)) {
                throw;
            }
        }
    }

    // Assume that startTime < endTime for both timestamp pairs
    function doTimeslotsOverlap(uint startTime1, uint endTime1, uint
startTime2, uint endTime2) private constant returns (bool) {
        if ((endTime1 < startTime2) || (endTime2 < startTime1)) {
            return false;
        }
        return true;
    }

    function hasReportDeadlineExpired(uint id) private constant returns
(bool) {
        if ((contracts[id].endTime + maxTimeFromEndToReportDeadline) >
now) {
            return false;
        }
        return true;
    }

    // A helper made to avoid "stack too deep" error in makeOffer.
    function storeAndLogNewOffer(uint id, uint price, uint
electricityAmount, uint startTime, uint endTime, address
sellerSmartMeter) private {
```

```
        contracts[id].seller = msg.sender;
        contracts[id].price = price;
        contracts[id].electricityAmount = electricityAmount;
        contracts[id].startTime = startTime;
        contracts[id].endTime = endTime;
        contracts[id].sellerSmartMeter = sellerSmartMeter;
        contracts[id].state = ContractState.Created;

        contractsBySmartMeter[sellerSmartMeter].push(id);

        LogOffer(msg.sender, id, price, electricityAmount, startTime,
endTime, sellerSmartMeter);
    }

    // Change the state ReadyForWithdrawal if report deadline has
    // expired. If not succesful for any reason, then throw.
    function makeReadyForWithdrawal(uint id) private {
        if ((contracts[id].state == ContractState.Accepted
            || contracts[id].state ==
ContractState.WaitingForSellerReport
            || contracts[id].state ==
ContractState.WaitingForBuyerReport)
            && hasReportDeadlineExpired(id))
        {
            contracts[id].state = ContractState.ReadyForWithdrawal;
            return;
        }
        throw;
    }

    function getSeller(uint id) constant returns (address) {
        return contracts[id].seller;
    }

    function getBuyer(uint id) constant returns (address) {
        return contracts[id].buyer;
    }

    function getBuyerReport(uint id) constant returns (bool) {
        return contracts[id].buyerReport;
    }

    function getSellerReport(uint id) constant returns (bool) {
        return contracts[id].sellerReport;
    }

    function isCreated(uint id) constant returns (bool) {
        return contracts[id].state != ContractState.NotCreated;
    }

    function getState(uint id) constant returns (ContractState) {
        return contracts[id].state;
    }

    function getSmartMetersContract() constant returns (address) {
        return smartMeters;
    }
}
```

The issuance and ownership of the smart meters are implemented in their separate smart contract, in the Solidity file *SmartMeters.sol*:

```
pragma solidity ^0.4.8;

contract SmartMeters {

    modifier onlyIssuer() {
        if ((msg.sender != issuer) || (!issuerSet)) {
            throw;
        }
        _;
    }

    address public issuer;
    mapping (address => address) public owner;
    bool issuerSet = false;

    function changeOwner(address meterAddress, address newOwner)
        onlyIssuer()
    {
        owner[meterAddress] = newOwner;
    }

    // A function that can be run one time that sets the issuer public
    // key. This would logically belong to the constructor or be a
    // preset value, but then it would not be possible to let Truffle
    // select it from one of the accounts made available by TestRPC, and
    // the account would have to be manually changed in code when
    // testing.
    function setIssuer(address _issuer) {
        if (!issuerSet) {
            issuer = _issuer;
            issuerSet = true;
        }
    }
}
```

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 all the created sell offers and the account balances of the buyer, the seller and the electricity market smart contract. 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>Electricity 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>Offers</h3>
    <table id="offers" class="table table-bordered">
      <thead>
        <th>ID</th>
        <th>Seller</th>
        <th>Seller smart meter</th>
        <th>Buyer</th>
        <th>Buyer smart meter</th>
        <th>Price</th>
        <th>Electricity amount</th>
        <th>Start time</th>
        <th>End time</th>
        <th>State</th>
        <th>Recipient</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[10]:

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

market.LogOffer().watch(function(error, result) {
    if (!error) {
        let newOffer = result.args;
        newOffer.state = "Waiting for acceptance"
        formatOfferTimestamps(newOffer);
        offers.push(newOffer);
        updateDynatable(dynatable, offers);
    }
});

market.LogAcceptOffer().watch(function(error, result) {
    if (!error){
        let newOffer = result.args;
        let i = offers.findIndex((obj => obj.id.equals(newOffer.id)));
        if (i !== -1) {
            offers[i].state = "Accepted";
            offers[i].buyer = newOffer.buyer;
            offers[i].buyerSmartMeter = newOffer.buyerSmartMeter;
            updateDynatable(dynatable, offers);
        }
    }
});

market.LogResolved().watch(function(error, result) {
    if (!error) {
        let newOffer = result.args;
        let i = offers.findIndex((obj => obj.id.equals(newOffer.id)));
        if (i !== -1) {
            offers[i].state = "Resolved";
            offers[i].recipient = newOffer.recipient;
            updateDynatable(dynatable, offers);
```

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

```
        }
    }
});

// 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 formatOfferTimestamps(offer) {
    const momentDateFormatString = 'HH:mm:ss, MMMM Do YYYY';
    offer.endTime =
moment.unix(offer.endTime).format(momentDateFormatString);
    offer.startTime =
moment.unix(offer.startTime).format(momentDateFormatString);
}

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

Aikaisemmin ilmestynyt ETLA Raportit-sarjassa (ennen ETLA Keskusteluaiheita)

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

No 56 *Niku Määttänen – Olli Ropponen*, Listaamattomien yhtiöiden osinkoverotus, tuotantopanosten allokaatio ja tuottavuus. 26.8.2016. 16 s.

No 57 *Kristian Lauslahti – Juri Mattila – Timo Seppälä*, Älykäs sopimus – Miten blockchain muuttaa sopimuskäytäntöjä? 12.9.2016. 29 s.

No 58 *Antti Tahvanainen – Peter Adriaens – Annu Kotiranta*, Growing Pains of Industrial Renewal: Case Nordic Cleantech. 26.9.2016. 59 p.

No 59 *Hannu Karhunen – Niku Määttänen – Roope Uusitalo*, Opintotukijärjestelmän uudistaminen: Rakenteelliseen malliin perustuvia vaikutuslaskelmia. 10.10.2016. 26 s.

No 60 *Mika Maliranta – Niku Määttänen – Mika Pajarinen*, Firm Subsidies, Wages and Labor Mobility. 13.10.2016. 18 p.

No 61 *John Zysman – Martin Kenney*, The Next Phase in the Digital Revolution: Platforms, Abundant Computing, Growth and Employment. 17.10.2016. 21 p.

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.

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)