Lauri Miettinen

**KLASSISEN AUTON ELINKAAREN SEURANTA ÄLYKKÄÄLLÄ SOPIMUKSELLA**

**KLASSISEN AUTON ELINKAAREN SEURANTA ÄLYKKÄÄLLÄ SOPIMUKSELLA**

Lauri Miettinen

Opinnäytetyö

Sysksy 2017

Tieto- ja viestintätekniikan koulutusohjelma

Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu

Tieto- ja viestintätekniikan koulutusohjelma, ohjelmistokehitys

Tekijä: Lauri Miettinen

Opinnäytetyön nimi: Klassisen auton elinkaaren seuranta älykkäällä sopimuksella

Työn ohjaajat: Veijo Väisänen,

Työn valmistumislukukausi ja -vuosi: 2/2017 Sivumäärä:60 + 5 liitettä

Tiivistelmä on suppea, itsenäinen esitys, joka antaa lukijalle kuvan tehdystä työstä. Se kirjoitetaan sitten, kun työ on valmis. Tiivistelmä kirjoitetaan täydellisin virkkein ja passiivimuodossa. Siinä selostetaan lyhyesti työn aihe, tavoitteet, työn keskeisin toteutustapa ja olennaiset tulokset.

Sisältö jakaantuu yleensä kolmeen kappaleeseen:

1. työn aihe ja tavoitteet

2. menetelmät, toteutustapa ja eteneminen

3. tulokset ja esitetyt johtopäätökset.

Tiivistelmän tulee mahtua yhdelle sivulle rivivälillä 1.

Korvaa nämä tekstit omillasi.

Asiasanat: x, x, x

(Kirjoita 3–7 kpl työtäsi kuvaavaa asiasanaa*.* Käytä hyväksesi esimerkiksi yleistä suomalaista asiasanastoa YSA, <http://onki.fi/fi/browser/overview/ysa>.)

ABSTRACT

Oulu University of Applied Sciences

Degree programme, software development

Author: Lauri Miettinen

Title of thesis: Tracking the life cycle of a classic car with a smart contract

Supervisors: Veijo Väisänen,

Term and year when the thesis was submitted: 2/2017 Pages: 60 + 5 appendices

Vieraskielinen tiivistelmä (Abstract) tehdään työn tilaajan niin vaatiessa tai osaston käytänteiden mukaisesti. Sen ei tarvitse olla suora käännös suomenkielisestä tiivistelmästä, vaan sen voi kirjoittaa kansainväliselle lukijalle.

Remember to include the following in the abstract:

Why – background for the thesis, assigner

What – objective of the thesis

How – background knowledge applied, research methods, material used

Achievements – main results

Exploitability – conclusions, development proposals, further measures

The length of the abstract is 1 page.

Keywords: x, x, x

(List 3–7 keywords that describe your thesis. Use for example keywords from the following controlled vocabulary thesauri:

MeSH <http://www.yso.fi/onto/mesh/conceptscheme>

Agriforest <http://www-db.helsinki.fi/agri/agrisanasto/Welcome_eng.html>

Helecon <http://helecon3.hkkk.fi/helevoc/?lang=eng&dbname=MIX>)

ALKULAUSE

Raportissa voi olla alkusanat, joissa kiitetään toimeksiantajaa, ohjaajia ja mahdollisesti muita tukijoita tai avustajia. Alkusanat voi sisältää lyhyesti myös muuta työhön liittyvää, lukijaa kiinnostavaa tietoa, joka ei muuten tule raportissa esiin. Alkusanat päivätään ja allekirjoitetaan.

Korvaa nämä tekstit omillasi.

Sanasto

Solmu. Englanniksi ”node”

Tiiviste. Englanniksi ”hash”.

SISÄLLYS

Contents

[1 JOHDANTO 8](#_Toc496195839)

[2 TEKNIIKKA 10](#_Toc496195840)

[2.1 Lohkoketju 10](#_Toc496195841)

[2.2 Ethereum-alusta 11](#_Toc496195842)

[2.2.1 Ethereum-alusta 14](#_Toc496195843)

[2.2.2 Ethereum-virtuaalikone 14](#_Toc496195844)

[3 SUUNNITTELU JA KÄYTTÖTAPAUKSET 15](#_Toc496195845)

[3.1 Kuvaus käyttötapauksista 15](#_Toc496195846)

[3.2 Lohkoketjun hyöty sovelluksessa 16](#_Toc496195847)

[4 OHJELMISTO 17](#_Toc496195848)

[4.1 Työn kannalta oleellinen ohjelmisto 17](#_Toc496195849)

[4.1.1 Solidity-kääntäjä 17](#_Toc496195850)

[4.1.2 Web3-kirjasto 17](#_Toc496195851)

[4.2 Ohjelmointityössä käytetty ohjelmisto 17](#_Toc496195852)

[4.2.1 Browser Solidity 18](#_Toc496195853)

[4.2.2 TestRPC 18](#_Toc496195854)

[4.2.3 Truffle-sovelluskehys 18](#_Toc496195855)

[4.2.4 Meteor 19](#_Toc496195856)

[4.3 Muu työhön liittyvä ohjelmisto 19](#_Toc496195857)

[4.3.1 GoEthereum (geth) 19](#_Toc496195858)

[4.3.2 Ethereum Wallet 19](#_Toc496195859)

[5 TYÖN KULKU 20](#_Toc496195860)

[6 LOPPUTULOKSET 24](#_Toc496195861)

[6.1 Toteutuneet käyttötapaukset 24](#_Toc496195862)

[6.2 Toteutumattomia käyttötapauksia 26](#_Toc496195863)

[7 ARVIO TEKNIIKASTA 29](#_Toc496195864)

[8 LOPPUSANAT 32](#_Toc496195865)

# JOHDANTO

Lohkoketjutekniikka sai alkunsa vuonna 2009, kun Satoshi Nakamoto kirjoitti tieteellisen julkaisun, jossa hän suunnitteli tietomallin hajautetulle valuutalle. Nykymaailmassa valuutta liikkuu pankin tai muun laitoksen kautta. Nakamoton idea oli luoda valuutta, josta ei ole vastuussa kukaan keskitetty taho, jossa maksajan ei tarvitse luottaa pankin tai valuuttapalvelun tietoturvaan ja palveluun. Nakamoto suunnitteli hajautetun järjestelmän rahan lähettämiselle ja maksutapahtumien todentamiselle. Nakamoton suunnitteleman järjestelmän pohjalta toteutettiin Bitcoin-kryptovaluutta. (1.)

Bitcoin-maksut tallennetaan hajautettuun tietokantaan, jota kutsutaan lohkoketjuksi. Mikään yksi yritys tai yksi taho ei ole vastuussa Bitcoin-maksuista. Kukin maksutapahtuma lähetetään vertaisverkkoon, jossa se todennetaan. Kuka tahansa voi liittää tietokoneensa Bitcoin-verkkoon todentamaan maksutapahtumia. Maksutapahtumat todennetaan menetelmällä, joka vaatii paljon laskentatehoa. Järjestelmän huijaaminen vaatisi hyökkääjältä enemmän laskentatehoa kuin mitä on kaikilla verkossa olevilla rehellisillä osallisilla. Bitcoin-verkkoon hyökkääjän on ainakin teoriassa mahdollista luoda itselleen rahaa, jota hänellä ei oikeasti ole. Käytännössä hyökkääjällä pitäisi silloin olla käytössään enemmän laskentatehoa kuin koko muulla maailmalla. (1.)

Tekniikka herätti maailmalla mielenkiintoa. Nakamoto pohti hajautettua valuuttaa suunnitellessaan, että olisi suunnitellut hajautetun ohjelmointikielen, mutta päättikin tehdä yksinkertaisemman järjestelmän, sillä ymmärsi tekniikan olevan kokeellinen ja haastava luonteeltaan. Siitä Vitalik Buterin keksi tehdä Ethereumin, hajautetun sovellusalustan, jonka tiedonsiirtotapahtumat toimivat lohkoketjun avulla. (3.)

Lohkoketju on arvokkaan tiedon tallentamiseen perustuva järjestelmä. Bitcoinin tapauksessa arvokas tieto on varallisuus ja valuutta. Klassiset autot ovat arvoesineitä. Niistä voidaan käydä satojen tuhansien, ja joskus jopa miljoonien eurojen kauppoja. Autoharrastajat kokevat, että autot ovat maansa kulttuurille merkityksellisiä, sillä kussakin klassisessa autossa on viitteitä aikansa kulttuurista. Tässä työssä pohditaan liiketoimintamalli ja käyttötapaukset lohkoketjutekniikkaa hyödyntävälle sovellukselle. Tavoite oli luoda Ethereum-alustalle älykäs sopimus, jota voitaisiin hyödyntää klassisen auton elinkaaren seurantaan.

Autoja huollettaessa huoltotietojen tallennus olisi haviteltava ominaisuus. Sekä tavallisille autoille, että klassisille autoille. Tavallisilla autoilla varmennettuja ja arkistoituja huoltotietoja voivat hyödyntää vakuutusyhtiöt, huoltoyhtiöt, sekä auton omistajat.

Klassisia autoja on huollettava, sillä ne voivat olla kymmeniä vuosia vanhoja, ja niiden osat kuluvat väistämättä. Lohkoketjuun tallennettua tietoa on vaikea muokata jälkeenpäin. Jos klassisen auton huoltotiedot ja käyttötilastot tallennettaisiin lohkoketjuun, auton omistaja voisi vakuuttaa huutokaupoissa ostajan autonsa arvosta.

Klassiset autot liittyvät myös liikennöinnin alaan. Liikennöinnin alalla on paljon tulevaisuuden sovelluksia, joissa voitaisiin hyödyntää lohkoketjutekniikkaa. Koska tämän työn aihe liittyy oleellisesti myös liikennöintiin, tässä työssä tehdyt havainnot ja tulokset pätevät myös liikennöinnin alalla yleisesti.

Toinen osa työtä oli luoda käyttöliittymä, jolla lohkoketjuun tehtäviä merkintöjä voi tehdä ja tarkastella. Tässä työssä kuvaillaan työn suunnittelua, toteutusta ja lopputulosta.

Työn tilaaja on Hilla-ohjelma, joka edesauttaa tulevaisuuden tekniikoiden tutkimusta ja kehitystä Suomessa. Työssä oli tavoitteena tehdä sovellus, jonka voi esitellä yhdellä tietokoneella, ja joka näyttää havainnollisesti, mitä älykkäillä sopimuksilla ja lohkoketjuilla voidaan tehdä tulevaisuudessa.

# TEKNIIKKA

## Lohkoketju

Lohkoketjussa kukin tiedonsiirto tallennetaan lohkoon. Yhdessä lohkossa on monia tiedonsiirtotapahtumia, kuten Bitcoinin tapauksessa rahan siirtotapahtumia. Kullakin lohkolla on viittaus sitä edeltäneeseen lohkoon. Sana *lohkoketju* on kielikuva verkon toimintatavasta. (1)

Kun uusi tiedonsiirto tehdään, lohkoketjuverkkoon lähetetään tieto tiedonsiirrosta. Vertaisverkossa tieto leviää solmulta toiselle. Bitcoin-verkossa uudet tiedonsiirtotapahtumat ovat ensin todentamattomien tapahtumien rekisterissä. Louhijasolmujen tehtävänä on luoda uusia lohkoja ketjuun. Uuden kelvollisen lohkon löytäminen on sattumanvaraista, ja vaatii monia kokeiluja. Uusi tiedonsiirtotapahtuma päätyy lohkoketjuun seuraavasti (1):

1. Uusi tiedonsiirtotapahtuma lähetetään solmulta toiselle.
2. Kukin solmu kerää uusia tiedonsiirtotapahtumia lohkoon.
3. Kukin solmu yrittää laskea kelvollisen tiivisteen lohkollensa.
4. Kun solmu onnistuu löytämään kelvollisen lohkon, se lähettää tiedon uudesta lohkosta verkon muille solmuille.
5. Solmut tarkistavat, ovatko ilmoitetut maksutapahtumat kelvollisia, eli oliko lähettäjällä tarpeeksi varallisuutta lähettää rahoja, joita hän lähetti.
6. Mikäli solmu toteaa lohkon olevan kelvollinen, se voi hyväksyä lohkon oman lohkoketjunsa uusimmaksi lohkoksi. Solmu voi aloittaa keräämään uusia maksutapahtumia taas seuraavaan lohkoon.

Järjestelmä toimii niin kauan kuin suurin osa verkon solmujen laskentatehosta pysyvät rehellisinä. Koska lohkoketjusta on kopio jokaisella verkon tietokoneella, ja koska uuden lohkon tekeminen vaatii paljon laskentatehoa, lohkoon päätyneen tiedon muuttaminen jälkeenpäin on lähes mahdotonta. On lähes mahdotonta huijata kaikkia verkon louhijasolmuja, koska jokainen louhijasolmu tarkistaa, onko lähettäjällä varaa tehdä rahansiirtoa. (1)

Lohkon laskenut solmu saa palkkioksi Bitcoineja uuden lohkon luomisesta. Tähän palkkioon perustuu lohkoketjukonesalien liiketoiminta. Tietokoneiden kuluttama sähkö maksaa, mutta kun lohko onnistutaan luomaan, siitä saatu kryptovaluutta voidaan myydä ostajalle, joka haluaa vaihtaa tavallista valuuttaa kryptovaluutaksi. (1.)

Epärehellinen louhijasolmu voisi yrittää huijata muita solmuja lähettämällä uusi lohko verkostoon, vaikka sen maksutapahtumat eivät olisi kelvollisia. Tämän solmun omistaja saisi petoksestaan palkkioksi kryptovaluuttaa. Kun lohko lähetetään verkkoon, kaikki muut solmut huomaavat, etteivät sen maksutapahtumat olekaan kelvollisia. Louhijasolmujen kannattaa siis olla rehellisiä, koska petoksen tekemisen jälkeen heidän varallisuutensa katoaisi välittömästi. (1.)

Koska samasta lohkoketjusta on kopioita lukemattomilla tietokoneilla. Tämäkin piirre tekee lohkoketjun hakkeroinnista käytännössä hyvin vaikeaa. Jos hyökkääjä haluaisi muunnella ketjussa olevia maksutapahtumia, hänen täytyisi silloin murtautua yhtäaikaisesti kaikkiin tietokoneisiin maailmassa, joissa on kopio lohkoketjusta. (7.)

## Ethereum-alusta

Ethereum-alusta on hajautettu sovellus. Hajautetulle sovellukselle on monia toisistaan eroavia määritelmiä. Määritelmien kesken yhteisenä tekijänä on sovellusten piirre, jossa sovellukset tallentavat tietonsa vertaisverkkoon. Lohkoketju on yksi tiedon tallentamiseen suunniteltu vertaisverkko. Toinen piirre on se, ettei hajautetuilla sovelluksilla ole ketään yhtä auktoriteettia, tahoa tai ylläpitäjää, jolla on päätäntävalta sovelluksessa. Ethereum-alustalle kehitettyjä älykkäitä sopimuksia kutsutaan hajautetuiksi sovelluksiksi. (16)

Ethereum-virtuaalikone, sekä kaikki siihen liittyvä ja kaikki alustan kehittämiseen liittyvä ohjelmisto on avoimen lähdekoodin ohjelmistoa. Ethereum-ohjelmistoja kehitetään maailmanlaajuisesti ja hajautetusti GitHub-palvelun, sekä muiden versionhallintapalveluiden avulla. (16)

Bitcoin-verkossa tiedonsiirtoa ovat bitcoin-maksut. Jokaisella tiedonsiirrolla on lähettäjän tili, vastaanottajan tili, sekä siirretty rahasumma. Ethereum-verkossa on näiden lisäksi myös dataa. (6.)

Solidity on suosituin Ethereum älykkäiden sopimusten kirjoitusta varten (15). Muita kieliä ovat LLL ja Serpent.

Hajautetuissa sovelluksissa on monia hyötyjä verrattuina perinteisiin, keskitettyihin sovelluksiin. Yksi hyöty on se, että ne eivät koskaan ole pois käytöstä. Perinteisissä palveluissa voi tulla palvelinvikoja. Hajautettu sovellus ei voi kokonaan poistua toiminnasta, elleivät kaikki vertaisverkon solmut lopeta toimintaansa yhtä aikaa. (15)

Vertaisverkkosovellukset eivät ole uusi, ennennäkemätön keksintö tietotekniikan historiassa. Esimerkiksi Bitcomet, joka on tiedostonsiirtoon käytetty vertaisverkko. Monet ohjelmistot jakavat ohjelmistopäivityksensä vertaisverkossa. Muun muassa Windows-käyttöjärjestelmän päivitykset hyödyntävät vertaisverkkoa. (18.)

Ethereum-virtuaalikone on Turing-täydellinen. Alustalla voi ajaa minkä tahansa protokollan, missä tahansa lohkoketjussa. Ethereum-alusta ei tarvitse vain tietyllä tekniikalla toteutettua lohkoketjua, vaan alusta voi hyödyntää minkä tahansa mallista lohkoketjua, mikä saatetaan kehittää tulevaisuudessa. Ethereum-virtuaalikone ei ole riippuvainen mistään tietystä ohjelmointikielestä, eikä virtuaalikoneen ajama koodi ole riippuvainen Soliditystä. Ethereum ei tarvitse mitään tiettyä tiedonsiirtoprotokollaa ollakseen yhteydessä vertaisverkkoon. Nämä periaatteet olivat taustalla, kun Buterin suunnitteli Ethereum-alustaa. (16)

Yhdessä vaiheessa ihmiset alkoivat kehittämään hajautettua alustaa Bitcoinin päälle. Näille alustoille kehitettiin tukia monen eri tyyppisille tiedonsiirtotapahtumille. Kaikesta huolimatta, niidenkin rajat tulivat aikanaan vastaan, jolloin täytyi taas kehittää uusia tiedonsiirtotapahtumia. (30.)

Bitcoin-verkkoon sovelutuva hajautettu alusta oli hyvin nihkeä. Järjestelmään ei voitu kehittää mitään kunnollisia innovaatioita. Nihkeys johtui siitä, etteivät alustojen virtuaalikoneet olleet Turing-täydellisiä. (Viitteeksi se yksi Buterin-video). Turing-täydellisyys tarkoittaa sitä, että ohjelmointikielellä voi ratkaista minkä tahansa laskennallisen ongelman. Ethereum-virtuaalikone on Turing-täydellinen, joten virtuaalikoneella ei ole mitään loogisia rajoitteita liittyen siihen, mitä sillä voi tehdä. Turing-täydellistä ohjelmointikieltä rajoittaa ainoastaan tietokoneen muistin määrä. (17)

Bitcoin-vertaisverkossa on Bitcoin-rahayksikön siirtämistä varten tiedonsiirtotapahtumia. Ethereum-verkossa vastaava rahayksikkö on eetteri (engl. Ether). Eetteri on Bitcoinin kaltainen kryptovaluuttarahayksikkö. Sitä voi käyttää maksamiseen Ethereum-verkossa. (6.)

Kun tiedonsiirtotapahtuma tehdään tietokoneella, tietokoneella oleva Ethereum-solmu lähettää tiedon siitä koko Ethereum-vertaisverkkoon. Ethereum-verkossa maksutapahtuma todennetaan. Minkä tahansa maksutapahtuman tekeminen vaatii kaikkia Ethereum-verkon louhijasolmuja ajamaan sama maksutapahtuma, sekä sen sisältämät loogiset operaatiot omalla tietokoneellaan. Maksutapahtuman alkuperäisen lähettäjän on maksettava tämä hinta korvauksena kaikille louhijasolmuille. Tätä kutsutaan kaasumaksuksi (gas price). Työläämmät laskentatoimet vaativat suuremman kaasumaksun tiedonsiirtotapahtuman lähettäjältä. (6.)

Ethereum-maksutapahtumilla on maksimi kaasurajoite, kuinka paljon ne saavat kuluttaa resursseja Ethereum-verkon louhijasolmuissa. Jos verkossa on sopimus, joka ajaa raskaita laskuoperaatioita, siihen tehdyt tiedonsiirtotapahtumat voidaan hylätä, jos ne ylittävät maksimikaasurajoitteen. Ethereum-verkon kaasurajoitteen takia verkko ei mene tukkoon, vaikka sinne lähetettäisiinkin älykkäitä sopimuksia, jotka vaativat liian raskasta laskentaa. (27.)

### Ethereum-alusta

Ethereumiin kuuluu lohkoketju, joka toimii tietokantana. Alustaan kuuluu virtuaalikone, joka on älykkäiden sopimusten ajoympäristö. Ethereum-alusta luo vertaisverkon, jossa käyttäjän koneella käynnissä oleva virtuaalikone on yhteydessä muualla maailmassa oleviin Ethereum-virtuaalikoneisiin. Ethereum-verkko koostuu maailmanlaajuisesti yhteen liittyneiden Ethereum-solmujen vertaisverkosta. Jos Ethereum-verkkoa haluaa käyttää, tulee omalle tietokoneelle käynnistää Ethereum-solmu, esimerkiksi GoEthereum (ks. 4.3.1).

### Ethereum-virtuaalikone

Ethereum-virtuaalikone sisältää Ethereum-koodiajoympäristön. Ethereum-alustan alkuaikoina yksi Ethereumin perustajista, Gavin Wood, teki teknisen spesifikaation siitä, mitä käskyjä Ethereum-virtuaalikoneen kuuluu toteuttaa (5). Täten Ethereum-virtuaalikoneen voi toteuttaa kuka tahansa, lähes millä tahansa ohjelmointikielellä. Virtuaalikone on tehty esimerkiksi Go-kielellä (ks. 4.3.1), sekä Pythonilla.

# SUUNNITTELU JA KÄYTTÖTAPAUKSET

## Kuvaus käyttötapauksista

Auton omistajalla on käytössään auton elinkaarta seuraava älykäs sopimus. Auton omistajan tavoitteena on kerätä auton älykkääseen sopimukseen merkintöjä auton elinkaaressa tapahtuneista merkityksellisistä tapahtumista. Tällaisia tapahtumia voivat olla esimerkiksi auton huoltotoimenpiteet, kilpailuvoitot ja asiantuntija-arviot. Merkintöjen tavoite on vakuuttaa tuleva ostaja auton arvokkuudesta. Tässä työssä merkinnöistä käytetään nimeä *kohokohta*, eli *highlight*.

Ajatuksena verkkosovelluksessa on, että kuka tahansa ihminen maailmassa voisi mennä auton älykkään sopimuksen kotisivuille tarkastelemaan auton nykytilaa, siinä olevia kohokohtamerkintöjä ja historiaa.

Ulkopuoliset käyttäjät voivat tehdä verkkosovelluksella kohokohtapyyntöjä (highlight request) älykkääseen sopimukseen. Kohokohtapyynnön voisi esimerkiksi tehdä auton korjannut mekaanikko tai asiantuntija, joka tekee auton autenttisuudesta ja kunnosta arvion. Kohokohtapyynnön tekijä voi pyytää pienen rahallisen korvauksen vaivannäöstä. Esimerkiksi auton korjaava mekaanikko voisi tehdä kohokohtapyynnön, jossa lukee ”Huolsin tämän auton Välivainion huoltoasemalla.” Hän voisi pyytää lomakkeen täyttämisen vaivannäöstä viisi euroa. Lomakkeen täyttämiseen kuluu muutama minuutti, mutta sekin vaatii pientä vaivannäköä. Ihmiset ovat suostuvaisempia muutaman minuutin vaivannäköön, jos siitä saa pienen rahallisen korvauksen.

Kukin kohokohtapyyntö voi maksaa auton omistajalle useita euroja tai kymmeniäkin euroja per merkintä. Merkinnät voivat silti todistaa auton arvokkuuden tulevalle ostajalle. Huutokaupassa tarjouksen tekijä saattaa tehdä suuren tarjouksen johtuen merkinnöistä, jotka ovat lohkoketjussa. Tällöin auton myyjän maksamat kymmenien eurojen summat maksavat itsensä moninkertaisesti takaisin.

Lisäksi kuka tahansa ihminen maailmassa voisi tehdä tarjouksen autosta. Kaikki tarjoukset ovat julkisia, ja kaikki pystyvät tarkastelemaan niitä. Julkisista tarjouksista voisi tehdä tilastoja, jotka antavat suuntaa-antavan kuvan auton arvosta. Auton omistaja voi hyväksyä tarjouksen, jolloin hän voi toimittaa auton ostajalle. Sopimuksen omistajuusoikeus siirtyy autokaupan yhteydessä.

## Lohkoketjun hyöty sovelluksessa

Tämän sovelluksen voisi kehittää perinteisenä palveluna, joka toimii tavallisella palvelimella, ja jonka merkinnät tallennetaan perinteiseen tietokantaan. Lohkoketjussa ja älykkäissä sopimuksissa on piirteitä, joista tämä sovellus voi hyötyä.

Yksi etu hajautetussa sovelluksessa on tietysti se, että hajautetussa sovelluksessa ei ole lainkaan palvelinkustannuksia. Tavallisessa palvelussa palvelin- ja ylläpitokustannukset täytyy tienata takaisin liiketoiminnalla, kuten mainoksilla. Tämä palvelu on luonteeltaan sellainen, että sitä ei välttämättä käytetä kovin usein, jolloin liiketoiminta voi olla haastavaa. (En ole varma onko tämä edes totta)

Toinen etu on lohkoketjumerkintöjen pitkäikäisyys. Klassisen auton elinkaari voi olla vuosikymmeniä, eivätkä kovin monet yritykset ole olemassa niin kauaa. Jos tavallinen palvelu lopettaa toimintansa, tietokannassa olevat merkinnät voivat kadota. Lohkoketju on paljon pitkäikäisempi kuin perinteinen tietokanta. Kopiot tiedoista ovat jokaisella Ethereum-verkon tietokoneella, joten tiedot eivät voi tuhoutua lopullisesti.

Lohkoketjussa kaikki maksutapahtumat ovat julkisia ja läpinäkyviä. Läpinäkyvyys sopii tämän sovelluksen käyttötapauksiin hyvin. Jos kaikki auton elinkaaressa tulleet tapahtumat ovat julkisia, on huijausten ja väärinkäytösten tekeminen kannattamatonta auton omistajalle. Sovelluksella tehtyjä merkintöjä tarkkailevat voivat tehdä tutkimusta, ja havaita, jos auton elinkaaren merkinnöissä esiintyy väärää tietoa tai ristiriitoja. Sosiaalisen median aikakautena tieto petollisesta auton omistajasta leviää nopeasti ihmiseltä ihmiselle. Omistajan maineen menetyksen pelko voi olla suuri. Mikäli tällainen sovellus on oikeasti käytössä tulevaisuudessa, autonomistajat varmasti ymmärtävät, että on kannattavaa pysyä rehellisenä lisättäessä kohokohtia.

# OHJELMISTO

## Työn kannalta oleellinen ohjelmisto

Kuvaillaan työn kannalta oleellista, sekä työssä käytettyä ohjelmistotekniikkaa.

### Solidity-kääntäjä

Solidity-kääntäjä kääntää Solidity-kielen Ethereum-tavukoodiksi, jonka Ethereum-virtuaalikone muuntaa konekäskyiksi. Solidity-kieli on hyvin yksinkertainen ja helposti opittava. Se muistuttaa Javascriptiä. Ohjelmoijayhteisö on kehittänyt monia Solidity-kääntäjiä. Yksi kääntäjä on Browser Solidity (ks. 4.2.1). Kääntäjiä on myös Truffle-sovelluskehyksessä sekä Ethereum Wallet -ohjelmassa.

### Web3-kirjasto

Web3 on javascript-kirjasto, joka pystyy lähettämään kutsuja paikalliselle ethereum-solmulle. Web3:n avulla internet-selaimessa olevat sovellukset voivat vuorovaikuttaa Ethereum-verkon kanssa. Web3:sta on olemassa npm-paketti, jonka voi ottaa käyttöön helposti Meteorissa. Kirjasto on hyvin dokumentoitu esimerkkeineen Ethereumin GitHubin wiki-sivuilla. (20)

## Ohjelmointityössä käytetty ohjelmisto

Tässä kuvaillaan ohjelmistoa, jota päädyttiin käyttämään lopullisessa työssä

### Browser Solidity

Browser Solidity on selaimessa käytettävä kääntäjä. Sen etuna on, että se toimii kaikilla tietokoneilla ilman ylimääräisten ohjelmien asennusta. (28)

### TestRPC

TestRPC on Ethereum älykkäiden sopimusten kehittämistä varten tehty node-palvelinsovellus, joka on tehty mukailemaan hyvin tarkasti Ethereum-lohkoketjun toimintaa. TestRPC toimii kehittäjän paikallisella tietokoneella. Maksutapahtumien todentaminen tapahtuu testrpc:ssä hyvin nopeasti, eikä maksutapahtumista tarvitse maksaa oikeaa, rahanarvoista Eetteriä. (23)

TestRPC:n nimessä kirjaimet RPC tarkoittavat Remote Procedure Call. Tekniikka, jonka avulla eri sovellukset puhuvat Ethereum-solmulle.

TestRPC:n asentaminen on helppoa Linuxilla, käyttäen Node pakettihallintaohjelmaa (Node package manager, npm). Windowsilla asentaminen on vaikeampaa. Kaikki Windows-koneelle tarvittavat resurssit on mahdollista asentaa ilmaisen Microsoft Visual Studio Community Edition -asennuksen yhteydessä. (24) (25)

### Truffle-sovelluskehys

Truffle on Ethereum -älykkäiden sopimusten kehittämistä varten luotu sovelluskehys, jonka on tarkoitus helpottaa Solidity-kielellä älykkäiden sopimusten koodin kirjoittamista. Sovelluskehyksen avulla voi tehdä älykkäiden sopimusten jatkuvaa integraatiota Ethereum-lohkoketjuun. Trufflen avulla voi myös koodata automatisoituja testejä sopimuksille. (22)

### Meteor

Meteor on sovelluskehys, joka yhdistää Node-palvelimen ja MongoDB -tietokannan hyvin helppokäyttöisesti. Meteor on saanut maailmalla huomiota, koska sen avulla voi tehdä samanaikaisesti verkkosivuja sekä mobiilisovelluksia. Meteoriin on helppo ohjelmoida reaktiivisia käyttöliittymiä, eli tehdä sovelluksesta sellainen, että sen käyttöliittymä näyttää reaaliaikaisesti kaikki päivitykset, mitkä tulevat tietokantaan. (21)

## Muu työhön liittyvä ohjelmisto

Kuvaillaan muuta ohjelmistoa, joka liittyy työhön, tai käytettiin jossain työn vaiheessa.

### GoEthereum (geth)

Go-kielellä ohjelmoitu Ethereum-virtuaalikone. Tämän työn sovellusta kehittäessä geth-solmua ei juurikaan käytetty, koska kehityksessä käytettiin lähinnä testrpc-testiverkkoa. Geth-solmua tarvittaisiin, jos työssä kehitetty älykäs sopimus haluttaisiin sijoittaa oikeaan Ethereum-verkkoon. (11) (12)

### Ethereum Wallet

Ethereum Wallet on helppokäyttöinen ohjelma, joka käynnistää tietokoneelle geth-solmun. Sen graafisella käyttöliittymällä voi tehdä maksutapahtumia Ethereum-verkkoon. Ethereum Walletissa on Solidity-kääntäjä, ja ohjelmalla voi lähettää sopimuksensa Ethereum-verkkoon. Valitettavasti Ethereum Wallet ei toimi testrpc:n kanssa, koska testrpc toteuttaa vain osan kaikista rajapinnoista, joita geth-solmussa on. Maksutapahtumia lähettäessä Ethereum Wallet käyttää rajapintaa signAndSendTransaction, mitä ei ole toteutettu testrpc:ssä.

# TYÖN KULKU

Aloituspalaverin pitämisen jälkeen aloitettiin pohtimaan työn tarkoitusta. Ohjelmistotyön idea keksittiin, siitä tehtiin vaatimusmäärittely, jonka työn tilaaja hyväksyi.

Tutustuttiin Ethereumiin käyttämällä Ethereum Wallet -ohjelmaa. Pystytettiin geth-testiverkko tietokoneelle. ClassicCarChain-älykkään sopimuksen koodaus aloitettiin.

Työskenneltiin geth-testiverkon parissa, kunnes todettiin, että nopeaa, iteroivaa työskentelytapaa varten testiverkko oli liian hidas. Asennettiin testrpc-testiverkko tietokoneelle, jonka jälkeen älykkään sopimuksen suorituskyky näytti paranevan.

Truffle-sovelluskehyksen käyttö aloitettiin. Kokeiltiin kehittää automaattisia testejä. Automaattisten testien koodaus oli kumminkin hyvin työlästä, joten niiden kehittämisestä luovuttiin myöhemmin. Työskentelyn ajan Trufflea käytettiin lähettämään (deploy) sopimus lohkoketjuun.

Meteor-sovelluskehyksen käyttö aloitettiin. Tehtiin Meteorilla Ethereumiin liittymättömiä harjoitustöitä, jotta kehittämisen perusteet tulisivat tutuiksi.

Älykkään sopimuksen ja Meteor-sovelluksen välisessä rajapinnassa alkoi ilmetä ongelmia, johtuen Solidity-kielen ja virtuaalikoneen teknisistä rajoitteista. Älykästä sopimusta uudelleen koodattiin ja eri ratkaisuja kokeiltiin.



Kuva X. ClassicCarChain.sol-tiedostossa lopulliset tietueet, joita on kohokohdassa. Huoltotiedot (maintenanceData) ovat yksi tietueiden joukossa, mutta huoltotietoja ei käytetty lopullisessa sovelluksessa.

Vastaan tuli ongelma käyttötapauksessa ”vierailijan pitää kyetä näkemään kaikki kohokohdat, mitä autolla on”. Solidity-kielessä ei ole tietorakennetta, jonka kaikki alkiot voisi hakea yhtä aikaa web3-rajapinnan kautta Meteor-käyttöliittymään. Ongelma ratkaistiin tekemällä Meteor-sovellukseen kiertokyselyrakenne, joka hakee yksitellen kaikki alkiot sopimuksesta.

Solidity-koodilla pystyy hakemaan yksittäisiä tietueita mapping-rakenteesta. Sopimukseen tehtiin mapping-rakenne, jossa kokonaislukuavainta (uint) vastasi Highlight struct-tietorakenne. Mapping-rakenteesta pystyi hakemaan yhden tietueen, jos funktiolle annettiin sitä vastaava kokonaislukuavaimen arvo (\_id) (Kuva X1).



Kuva X1. Näytteitä tiedostosta ClassicCarChain.sol, liittyen kaikkien kohokohtien hakuun älykkäästä sopimuksesta.

Kiertokyselyrakenne on javascript-koodissa 03-eth-highlights.js (Kuva X2). Koodi ajetaan joka kerta kun havaitaan, että uusi lohko on lisätty Ethereum-lohkoketjuun (rivi 41, web3.eth.filter(’latest’).watch). Jotta kaikki kohokohdat saataisiin haettua, täytyy ensin hakea älykkäästä sopimuksesta kohokohtien määrä (rivillä 47, highlightIndex-funktio). Tämän jälkeen käydään läpi kohokohtia for-lauseessa, ja kutsutaan yksitellen jokaista indeksiarvoa kohden sopimuksesta kohokohdan tiedot (rivi 56, GetHighlight-funktio).



Kuva X2. Näyte koodista 03-eth-highlights.js. Koodissa on kiertokyselyrakenne, jolla haetaan kaikki kohokohdat Meteor-sovellukseen.

Solidity-kielessä ole null-käsitettä. Jos funktiota GetHighlight kutsutaan kysyen kokonaislukuavaimella, jota ei vastaa mikään alustettu kohokohta, silloin funktio palautta Highlight-tietueen, jonka kaikki arvot ovat oletusarvoissaan. Muussa ohjelmointikielessä kutsu olisi palauttanut null-arvon. Tämä vaikeuttaa tilannetta, kun halutaan tarkistaa, onko mapping-tietorakenteen arvoja alustettu. Ongelma ratkaistiin lisäämällä highlights mapping -tietorakenteeseen tietue ”initialized” (”alustettu”). Tämä tietue asetetaan arvoon *tosi* aina, kun tietue alustetaan. Tietue on *epätosi,* jos arvoa ei ole alustettu. Koodissa voidaan tarkistaa erikseen, palauttiko funktio arvoa, joka oli alustettu.

Hyviä ohjelmistosuunnitteluperiaatteita pyrittiin seuraamaan suunnittelemalla sopimus uudelleen siten, että jokainen kohokohtamerkintä olisi Ethereum-lohkoketjussa oma alisopimuksensa, joihin ClassicCarChain-pääsopimus tekisi kutsuja. Suunnitelmasta jouduttiin kumminkin luopumaan, koska Ethereum-virtuaalikoneessa sopimus ei voi hakea toisesta sopimuksesta arvoa, jonka pituus on muuttuva. Tällaisia tietueita ovat muun muassa merkkijonot (string), sekä taulukot (array). (viite)

Lopuksi tehtiin käyttöliittymää. Ethereum-yhteisön kehittämää dappstyles.styl-tyyliä kokeiltiin. Työssä päädyttiin tekemään oma yksinkertainen css-tyyli. (26)

# LOPPUTULOKSET

Työtä kehitettiin GitHub-repositoriossa, josta sen voi ladata. Lueminut-tiedostossa on ohjeita työn käynnistämistä varten. Kuvassa X on auton omistajan näkymä sovelluksen etusivuilta. Yläpalkissa on sovelluksessa navigointilinkit, oikeassa yläkulmassa aktiivisen tilin vaihtaminen. Yläpalkin alapuolella lomake kohokohdan lisäämistä varten. Sininen laatikko on yksi kohokohta, jossa auton omistaja näkee poistamispainikkeen, sekä tekstikentän, johon voi kirjoittaa syyn merkinnän poistamiselle.



Kuva X. Lopullinen sovellus kohokohtasivulta.

## Toteutuneet käyttötapaukset

* Auton omistaja näkee sovelluksessa eri toimintoja kuin muut käyttäjät.
* Kun käyttäjä käynnistää sovelluksen, hän pystyy vaihtamaan käyttäjätiliään. Kun käyttäjätiliä vaihtaa, käyttöliittymä päivittyy välittömästi näyttämään käyttäjälle ne ominaisuudet ja tiedot, mitä käyttäjällä on oikeus nähdä.
* Autentikaatio tapahtuu Ethereum-verkossa. Kuka tahansa voisi esimerkiksi kutsua verkkosivuilla funktiota, jolla saisi auton omistajuuden (gainOwnership). Tällöin Ethereum-verkossa havaitaan, ettei kutsun lähettänyt tili olekaan auton omistaja, jolloin mitään ei tapahdu.
* Omistaja pystyy lisäämään kohokohtia omaan autoonsa. Kohokohdassa on tietueina muun muassa merkinnän lisäämisaika, viesti ja merkinnän tekijä.
* Omistaja voi poistaa kohokohtamerkintöjä.
* Muut tilit pystyvät lisäämään kohokohtapyyntöjä. He voivat pyytää rahasumman palkkioksi auton omistajalta siitä vaivannäöstä, että he tekivät kohokohtapyynnön.
* Auton omistaja pystyy lähettämään rahaa sopimuksen saldoon omalta tililtään. Kohokohtapyynnön hyväksyessä raha siirtyy sopimuksen saldosta kohokohtapyynnön tekijälle. Jos sopimuksessa ei ole tarpeeksi saldoa, kohokohtapyyntöä ei voi hyväksyä. Tämä rajoite on sovelluksessa Ethereum-alustan turvallisuusrajoitteiden takia.
* Auton omistaja voi nostaa rahaa sopimuksen saldosta.
* Omistaja pystyy hyväksymään kohokohtapyynnön, jolloin se lisätään auton kohokohtien joukkoon. Samalla sopimuksen saldosta siirtyy pyydetty summa kohokohtapyynnön tekijälle.
* Omistaja voi kieltäytyä kohokohtapyynnöstä, jolloin se poistetaan.
* Muu kuin auton omistaja pystyy tekemään tarjouksen autosta.
* Tarjouksen tekijä voi myös poistaa tarjouksensa, jos muuttaa mieltään.
* Auton omistaja voi kieltäytyä tarjouksesta, jolloin se poistuu tarjousten listalta.
* Omistaja pystyy hyväksymään tarjouksen, jolloin sopimus siirtyy ”omistajuus on vaihtumassa” -tilaan. Tässä tilassa sovelluksen etusivuilla näkyy tiedote omistajuuden siirtymisestä kaikille käyttäjille. Omistajuuden vaihtumistilassa sekä auton ostaja että myyjä voivat perua kaupan.
* Kun auton omistajuus on vaihtumassa, auton ostaja näkee painikkeen, jolla hän pystyy saamaan sopimuksen omistusoikeudet. Ostaja voi painaa tästä painikkeesta, kun auto on toimitettu hänelle.
* Painikkeen painamisen jälkeen uusi omistaja saa kaikki käyttöoikeudet, mitä edellisellä omistajalla oli. Edellinen omistaja näkee samat ominaisuudet kuin muutkin tavalliset käyttäjät.
* Yllä mainittujen toimintojen teko tallentaa sopimukseen historiatietomerkinnän (Solidity-kielessä ”event”). Merkintöjä tallentuu myös silloin kun tietueita poistetaan. Historiatietomerkintöjä voi tarkastella kuka tahansa, eikä niitä voi poistaa kukaan.

## Toteutumattomia käyttötapauksia

Varhaisessa työn vaiheessa suunniteltiin roskapostin estojärjestelmä kohokohtapyynnöille. Nykyisessä sovelluksessa yksikin käyttäjä pystyisi lähettämään tuhansia kohokohtapyyntöjä päivässä. Tätä varten kehitettiin työn aikana ominaisuus, jota voisi luonnehtia kohokohtien pyyntöoikeuksiksi (highlight request rights). Tavoitteena oli kehittää järjestelmä, jossa auton omistaja voisi antaa eri käyttäjille oikeudet lisätä kohokohtapyyntöjä. Ominaisuus olisi hyödyllinen esimerkiksi julkisuuden henkilölle, jonka auto on kirjattu sovellukseen. Julkinen auto saa paljon huomiota sosiaalisessa mediassa, jolloin roskaposti voi olla riesa auton omistajalle. Epäsuositun auton omistaja voisi laittaa asetuksen pois päältä, jolloin kuka tahansa käyttäjä voi lähettää kohokohtapyyntöjä. Yhdessä ohjelman versiossa tätä ominaisuutta kehitettiin, mutta kehitys lopetettiin, koska päätettiin, ettei käyttötapauksen esittely ollut tarpeeksi tärkeää työn tarkoituksen kannalta. Tämä työ toimii vain yhdellä tietokoneella, mutta mikäli sopimus lähetettäisiin varsinaiseen Ethereum-verkkoon, tämä ominaisuus olisi hyödyllinen.

Toinen työn alkuvaiheissa suunniteltu käyttötapaus on rahasummista tinkiminen. Olisi mahdollista koodata sopimukseen logiikkaa, jossa auton omistaja voisi tinkiä kohokohtapyynnöstä pyydettyä summaa. Tinkimisen voisi lisätä myös autokaupassa tarjouksen tekijän ja auton omistajan välille. Tästä käyttötapauksesta luovuttiin, koska se on työläs kehittää. Käyttötapaus vaatisi hyvin suunnitellun, hyvin testatun, virheettömän koodin älykkääseen sopimukseen. Maksutapahtumia käsitellessä raha saattaa kadota ohjelmointivirheen seurauksena. Ominaisuus vaatisi lisäksi selkeän käyttöliittymän Meteor-sovellukseen, jotta käyttäjä ymmärtää, missä vaiheessa rahanvaihto ja tinkimien ovat.

Kolmas toteuttamaton käyttötapaus on historiatietojen suodattaminen ja järjestely. Lopullisessa sovelluksessa kaikki historiatiedot ovat samassa listassa peräjälkeen. Olisi käytettävyyden kannalta parempi, jos listaan voisi lisätä suotimia ja siitä voisi tehdä hakuja.

Neljäs on uusien ajoneuvojen lisääminen Ethereum-verkkoon. Sovellukseen olisi pystynyt lisäämään lomakkeella uuden auton, jolle tehdään uusi sopimus lohkoketjuun. Tätä varten tehtiin käyttöliittymä, jonka pystyy näkemään lopullisessa sovelluksessa. Käyttöliittymää pääsee katsomaan menemällä sivulle <http://localhost:3000/> käynnistetyssä sovelluksessa. Toiminnallisuus jätettiin tekemättä. Toteutus vaatisi Meteor-sovelluksen Javascript-koodin uudelleen suunnittelua. (Kuva X)



Kuva X. Lomake, jolla uuden sopimuksen voisi luoda.

Viidentenä suunniteltiin, että huoltotiedot voisi lisätä lomakkeella kohokohtaan. Yhdestä kohokohdasta voisi tehdä tyypiltään huoltotieto-tyyppisen kohokohdan, jolloin kohokohdassa olisi tekstin ja päivämäärän lisäksi myös lisätietoja tehdystä huollosta. Tämä käyttötapaus oli haastava toteuttaa, sillä Solidityssä tuki muuttuvan pituisille tietorakenteille, kuten taulukoille, on ongelmallinen. Tämä käyttötapaus olisi hyvin vakuuttavalla tavalla esitellyt älykkäiden sopimusten tuomaa hyötyä liikennöinnin alalla. Lopulta kumminkin todettiin, ettei käyttötapaus ollut elintärkeä työn tarkoituksen kannalta, eikä sitä toteutettu. Huoltotiedot suunniteltiin koodattavan aluksi vain viitenä huoltotehtävänä, joille kullekin voisi merkata huollon lopputuloksen:

**struct MaintenanceTasks {**

**MaintenanceOutcome engine;**

**MaintenanceOutcome tires;**

**MaintenanceOutcome pedals;**

**MaintenanceOutcome brakes;**

**MaintenanceOutcome interior;**

**}**

**enum MaintenanceOutcome {**

**NotChecked, Checked, CheckedAndFoundFault,**

**FoundFaultAndFixed**

**}**

Kuudentena on tietoturvan puutteet. Lopullisessa sovelluksessa auton myynnissä ja ostossa on joitakin tietoturva-aukkoja. Nykyinen järjestelmä vaatii ostajaa lähettämään koko tarjoamansa summan sopimukseen. Tämä tehdään tarjouksen tekohetkellä. Kymmenien tai satojen tuhansien eurojen ostossa, on epäkäytännöllistä kiinnittää niin suuria rahasummia pitkäksi aikaa. Olisi suositeltavaa, jos sekä ostaja, että myyjä lähettäisivät pienen osuuden kaupasta, esimerkiksi 5%, auton hinnasta sopimukseen. Auton ostaja lähettäisi rahat tarjouspyyntöä tehtäessä, ja myyjä lähettäisi tarjouspyynnön hyväksyessä. Kun molemmilla osapuolilla on rahaa kiinnitettynä sopimukseen, molemmilla on vähemmän motiiveja yrittää petosta autokaupan aikana. Auton myyjä saisi koko kaupan rahat, kun ostaja on hyväksynyt kaupan (Painamalla painiketta ”Olen saanut ajoneuvon. Ota sopimuksen omistajuus”)

Viimeisenä mainittakoon, että nykyisessä sovelluksessa sama käyttäjä voi tehdä useamman tarjouksen. Olisi parempi, jos yksi käyttäjä voisi tehdä vain yhden tarjouksen, jolloin uudempi tarjous korvaa vanhan.

# ARVIO TEKNIIKASTA

Truffle on varmasti hyödyllinen projektissa, jossa on tuhansia rivejä Solidity-koodia, sekä suurempi kehitysryhmä, jossa ainakin yksi kehittäjä voi kuluttaa aikaa testien koodaamiseen. Tämän laajuisessa työssä Truffle-sovelluskehyksestä saatu hyöty jäi vähäiseksi.

Meteorilla työskentely oli nopeaa ja sujuvaa. Tässä sovelluksessa, sekä kaikissa muissakin Ethereum-sovelluksissa Meteorin reaktiivisuus on hyvin hyödyllinen. Lohkoketjussa tapahtuvat muutokset näkyvät sovelluksen käyttöliittymässä välittömästi. Meteor sopii hyvin Ethereum-sovellusten kehittämiseen.

Solidity-kieli jätti toivomisen varaa. Kielessä on paljon teknisiä rajoitteita. Osa rajoitteista johtuu ohjelmointikielen vähäisestä kehityksestä. Tämä työ tehtiin Solidity-kielen versiolla 0.4.7. Osa solidity-kielen rajoitteista johtuu Ethereum-virtuaalikoneen rajoitteista. Solidity-kieli on hyvin monipuolinen ja helppokäyttöinen, mutta sitä käyttävien kehittäjien on hyvä olla tietoisia rajoitteista, joita kielessä on. Rajoitteista tietäminen säästää kehitysaikaa, ja vähentää turhan työn ja kokeilujen määrää.

Solidity-kielessä ei voi hakea koko mapping-tietorakenteen kaikkia alustettuja alkioita. Kehittäjän täytyy itse kirjoittaa funktiot koko mapping-tietorakenteen alkioiden haulle. Tämä tulee hidastamaan ja vaikeuttamaan hajautettujen sovellusten kehitystä, niin kauan kuin se on olemassa.

Tuotantokäytössä olevassa älykkäässä sopimuksessa tietueita voi olla satoja tai tuhansia. Yhden tietueen hakeva funktio on niin hidas suorittaa, että tässä työssä toteutettu ratkaisu ei ole käytännöllinen tuotantokäytössä olevalle älykkäälle sopimukselle.

Vaikka tietorakenne olisikin suorituskyvyltään hidas, olisi se silti tärkeä ominaisuus, koska se lisäisi innovaatioiden määrää Ethereum-sovelluksien kehittäjien keskuudessa. Älykkäiden sopimuksien kehittäjien yhteisössä innovaatiot ovat hyvin tärkeitä Ethereum-alustan tulevaisuuden kannalta.

Solidity-kielellä ei voi kutsua vieraassa sopimuksessa olevaa funktiota, joka palauttaa string-tyyppisen muuttujan. Yhdessä vaiheessa työtä tehtiin ohjelmistosuunnitelma, jossa pyrittiin tekemään kohokohdista omia sopimuksiaan. Kohokohtasopimuksista pääsopimus voisi lukea tietoja. Suunnitelma hylättiin tässä mainitun teknisen rajoitteen takia. Älykkään sopimuksen koodi olisi ollut yksinkertaisempaa ja helpompi jatkokehittää, jos rajoitetta ei olisi ollut.

Solidityssä ei ole tukea useille sopimuksen konstruktoreille. On mahdollista suunnitella ohjelmansa eri tavalla siten, että kehittäjä voi sopeutua tähän rajoitteeseen. Joissakin ohjelmissa tämä voi tehdä koodista epäselvää ja vaikeasti luettavaa.

On huomattava puute, että Solidity-koodissa ei ole null-käsitettä. Monissa ohjelmissa kehittäjä haluaisi tehdä funktioita, joka joko palauttaa arvon, kun funktion kutsu onnistuu, mutta palauttaa arvon null silloin, kun kutsu ei onnistu. Null-käsite on myös hyödyllinen virheenkäsittelyssä, kun halutaan tarkistaa, onko muuttujia alustettu.

Vaikka rajoitteet kuulostavat pieniltä, yhdessä ne rajoittavat kaikkia mahdollisia käyttötapauksia, mitä Ethereum-alustalla voisi muuten toteuttaa. Rajoitteet tulevat hidastamaan Ethereumin läpimurtoa yleiseen, maailmanlaajuiseen käyttöön. Tästä huolimatta, Soldity-kieltä kehitetään parhaillaan, ja joitakin hyvin oleellisia parannuksia on tehty viime aikoinakin, kuten (?).

Voi olla kannattavaa kertoa EtherSimistä, ja kertoa kuinka käsittämättömän hämmentävää on pysyä mukana Ethereum-ohjelmistokehityksen nykytilanteesta. Aina kaikkea uutta tehdään ja jotain heitetään roskakoriin. Vaikea löytää mikä on uusinta tietoa. Voit laittaa esimerkiksi viitteen siihen uhkapelisovelluksen, jossa kehittäjä turhautui Mist/Ethereum Wallet -sekoiluun, ja hylkäsi koko sovelluksen.

Silti kukaan ei voi kieltää Solidity-kielen mahdollisuuksia. Kieli on niin yksinkertainen, että kuka tahansa, jolla on ymmärrystä ohjelmoinnista, voi oppia sen. Kielen helppokäyttöisyydellä ja yksinkertaisuudella on suuri merkitys Ethereum-alustan maailmanlaajuisessa läpimurrossa.

# LOPPUSANAT

Työssä oli tavoitteena luoda käytännön sovellus, joka havainnollisesti esittelee, miten tulevaisuudessa voitaisiin hyödyntää älykkäitä sopimuksia. Tätä varten tutustuttiin Ethereum-alustan ohjelmistokehitykseen.

Ethereum-alustaa varten on kehitetty monia helppokäyttöisiä kehitystyökaluja. Solidity-kielen ominaisuuksissa on parannettavaa. Rajoitteet hidastavat kehitystä ja rajoittavat innovaatioita, joita Ethereumin älykkäillä sopimuksilla voi kehittää. Solidity-kieli on silti helppokäyttöinen ja se on helppo oppia.

Meteor on hyvin helppokäyttöinen sovelluskehys. Se soveltuu hyvin Ethereum-alustaan pohjautuviin sovelluksien kehittämiseen.

Lohkoketjutekniikan tutkijat puhuvat, että historia voi jakaantua ennen lohkoketjuja olleeseen, ja lohkoketjujen jälkeiseen maailmaan, että lohkoketjut ovat keksintönä yhtä merkityksellinen kuin internet. (19)

Elämmekö tulevaisuudessa maailmassa, jossa raha, tavaroiden ja palvelujen korvaukset liikkuvat ihmiseltä ihmiselle, ilman välikäsiä, kaikki käyttäen lohkoketjualustalla toimivia älykkäitä sopimuksia? Jää nähtäväksi, tuleeko ennustuksesta totta.

Lähteet

1. Nakamoto, Satoshi 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. Saatavissa: <https://bitcoin.org/bitcoin.pdf>. Hakupäivä 25.9.2017.
2. <https://www.youtube.com/watch?v=3PdO7zVqOwc> Ei voi käyttää, puolueellinen!
3. <http://futurethinkers.org/vitalik-buterin-ethereum-decentralized-future/>
4. <https://blog.ethereum.org/2016/11/01/security-alert-solidity-variables-can-overwritten-storage/>
5. Wood, Gavin. Ethereum: A Secure Desentralised Generalised Transaction Ledger Saatavissa: <http://gavwood.com/paper.pdf> tai <http://paper.gavwood.com/>. Hakupäivä 25.09.2017.
6. <https://solidity.readthedocs.io/en/latest/introduction-to-smart-contracts.html>
7. <https://www.youtube.com/watch?v=4sm5LNqL5j0&t=10s>
8. Aron, Jacob 2014. Building on Bitcoin. New Scientist, Vol. 221, Issue 2955. Saatavissa: Ebsco-tietokanta (vaatii käyttöoikeuden). Hakupäivä: 25.9.2017.
9. <https://ethereum.org/>
10. <http://ethdocs.org/en/latest/contracts-and-transactions/developer-tools.html#the-evm>
11. <http://ethdocs.org/en/latest/ethereum-clients/choosing-a-client.html#why-are-there-multiple-ethereum-clients>
12. <http://ethdocs.org/en/latest/ethereum-clients/go-ethereum/index.html>
13. <https://youtu.be/Fjhe0MVRHO4?t=9m3s>
14. <https://youtu.be/3PdO7zVqOwc?t=8m30s> Ei voi käyttää, puolueellinen!
15. Building Blockchain ProjectsBy: Narayan Prusty Publisher: Packt Publishing Pub. Date: April 27, 2017 Saatavilla: <http://proquest.safaribooksonline.com/> (vaatii käyttöoikeuden)
16. Swan Melanie 2015. Blockchain. Kappale: Ethereum: Turing-Complete Virtual Machine. O'Reilly Media, Inc. ISBN-13: 978-1-4919-2049-7. Saatavilla Safari Books Online (vaatii käyttöoikeuden).
17. Computerphile 2016. Turing Complete. Saatavissa: <https://www.youtube.com/watch?v=RPQD7-AOjMI>. Hakupäivä 25.09.2017.
18. <https://youtu.be/Fjhe0MVRHO4?t=1m54s> 1:54-8:53 D2P3 - TNABC 2015 - VITALIK BUTERIN FOUNDER ETHEREUM - Bitcoin 2.0 - Ideas and Applications
19. <https://www.youtube.com/watch?v=40ikEV6xGg4> Block chain demystified
20. <https://github.com/ethereum/wiki/wiki/JavaScript-API> Web3 JavaScript app API
21. <https://www.meteor.com/>
22. <http://truffleframework.com/>
23. <https://github.com/ethereumjs/testrpc>
24. <https://github.com/ethereumjs/testrpc/wiki/Installing-TestRPC-on-Windows>
25. <https://www.visualstudio.com/downloads/>
26. <https://github.com/ethereum/dapp-styles>
27. <https://ethereum.org/ether>
28. <https://remix.ethereum.org/>
29. <https://hn.premii.com/#/article/14817557>
30. Buterin, Vitalik 2016. DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin. Saatavissa: <https://youtu.be/gjwr-7PgpN8> Hakupäivä 27.10.2017
31. Wood, Gavin 2015. DEVCON1: Ethereum for Dummies - Dr. Gavin Wood <https://youtu.be/U_LK0t_qaPo?t=2m59s> 2:59 alkaen.Hakupäivä 27.10.2017