Lauri Miettinen

**KLASSISEN AUTON ELINKAAREN SEURANTA ÄLYKKÄÄLLÄ SOPIMUKSELLA**

**KLASSISEN AUTON ELINKAAREN SEURANTA ÄLYKKÄÄLLÄ SOPIMUKSELLA**

Lauri Miettinen

Opinnäytetyö osat 2 ja 3

Syksy 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, Janne Kumpuoja, Marko Jurmu

Työn valmistumislukukausi ja -vuosi: 2/2017 Sivumäärä:36 + 1 liite

Lohkoketjutekniikka sai alkunsa vuonna 2009, kun Satoshi Nakamoto kirjoitti tieteellisen julkaisun, jossa hän suunnitteli tietomallin hajautetulle valuutalle. Nakamoton suunnitteleman järjestelmän pohjalta toteutettiin BitCoin-kryptovaluutta. Bitcoin-maksut tallennetaan hajautettuun tietokantaan, jota kutsutaan lohkoketjuksi. Siitä Vitalik Buterin keksi tehdä Ethereumin, hajautetun sovellusalustan, jonka tiedonsiirtotapahtumat tallennetaan lohkoketjuun.

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. Tässä työssä suunniteltiin liiketoimintamalli ja luotiin sovellus, jolla voisi esitellä havainnollisesti, miten tulevaisuudessa voitaisiin hyödyntää älykkäitä sopimuksia liikennöinnin alalla. Työn tilaaja on Hilla-ohjelma, joka edesauttaa tulevaisuuden tekniikoiden tutkimusta ja kehitystä Suomessa.

Ohjelmistotyön ensimmäinen osa oli luoda Ethereum-alustalle älykäs sopimus, jota voitaisiin käyttää klassisen auton elinkaaren seurantaan. Toinen osa oli luoda käyttöliittymä, jolla sopimuksen kanssa voisi vuorovaikuttaa. Älykäs sopimus kirjoitettiin Solidity-kielellä, käyttöliittymä luotiin Meteor-sovelluskehyksellä.

Työssä onnistuttiin toteuttamaan suurin osa suunnitelluista käyttötapauksista. Työskentelyä vaikeuttivat eniten Solidity-kielen puutteelliset ominaisuudet. 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.

Asiasanat: ohjelmistokehitys, lohkoketju, BitCoin, Ethereum, JavaScriptABSTRACT

Oulu University of Applied Sciences

Degree programme in information technology, software development

Author: Lauri Miettinen

Title of thesis: Developing a smart contract for tracking the life cycle of a classic car

Supervisors: Veijo Väisänen, Janne Kumpuoja, Marko Jurmu

Term and year when the thesis was submitted: 2/2017 Pages: 36 + 1 appendix

The block chain technology began in 2009 when Satoshi Nakamoto wrote a paper on a system that allows a decentralized currency. The BitCoin-cryptocurrency was invented based on Nakamoto’s system. BitCoin-transactions are saved into a decentralized data based called the block chain. Years later, a man named Vitalik Buterin invented Ethereum, a decentralized computing platform whose transactions are saved into the block chain.

The block chain is a system for saving any valuable information. In BitCoin’s case, the valuable information is wealth and currency. Classic cars are valuable. Classic car sales can cost hundreds of thousands, even millions of euros. In this thesis a business model was designed, and a decentralized application was created. The goal of the project was to show in a concrete way how smart contracts could be used in the future. This thesis was made for the Hilla-program – a Finnish research project for studying future technologies.

The first part of the application was to create a smart contract for the Ethereum-platform for tracking the life cycle of a classic car. The second part was to create an interface to allow users to interact with the smart contract. The smart contract was written with the Solidity-language and the interface was created with the Meteor-framework.

Most of the planned use-cases were successfully implemented. The greatest obstacle in the project were the lacking features of the Solidity-language. The limitations will slow down development and restrict innovations that could be developed into smart contracts. Solidity-language is, however, easy to learn. Meteor is a very useful and easy to use framework. It can be easily applied to interact with Ethereum -smart contracts.

Keywords: software development, block chain, BitCoin, Ethereum, JavaScript

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.

SISÄLLYS

[1 JOHDANTO 7](#_Toc499572280)

[2 TYÖN TEKNIIKKAAN TUTUSTUMINEN 9](#_Toc499572281)

[2.1 Bitcoin ja lohkoketju 9](#_Toc499572282)

[2.2 Hajautetut sovellukset 10](#_Toc499572283)

[2.3 Ethereum-alusta 11](#_Toc499572284)

[3 SUUNNITTELU JA KÄYTTÖTAPAUKSET 14](#_Toc499572285)

[3.1 Kuvaus käyttötapauksista 14](#_Toc499572286)

[3.2 Älykkään sopimuksen hyöty sovelluksessa 15](#_Toc499572287)

[4 TYÖHÖN LIITTYVÄ OHJELMISTO 17](#_Toc499572288)

[4.1 Solidity-kääntäjä 17](#_Toc499572289)

[4.2 Web3-kirjasto 17](#_Toc499572290)

[4.3 Browser Solidity 17](#_Toc499572291)

[4.4 TestRPC 17](#_Toc499572292)

[4.5 Truffle-sovelluskehys 18](#_Toc499572293)

[4.6 Meteor 18](#_Toc499572294)

[4.7 GoEthereum (geth) 18](#_Toc499572295)

[4.8 Ethereum Wallet 18](#_Toc499572296)

[5 TYÖN KULKU 20](#_Toc499572297)

[6 LOPPUTULOKSET 24](#_Toc499572298)

[6.1 Toteutuneet käyttötapaukset 24](#_Toc499572299)

[6.2 Toteutumattomia käyttötapauksia 26](#_Toc499572300)

[7 ARVIO TEKNIIKASTA 30](#_Toc499572301)

[8 LOPPUSANAT 32](#_Toc499572302)

# 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 tallennetaan lohkoketjuun. (2.)

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. Tässä työssä pohditaan liiketoimintamalli ja käyttötapaukset lohkoketjutekniikkaa hyödyntävälle sovellukselle. Ohjelmistotyön ensimmäinen osa 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.

# TYÖN TEKNIIKKAAN TUTUSTUMINEN

Älykkään sopimuksen koodaaminen vaati Ethereum-alustan perustoiminnallisuuden ymmärrystä. Koska Ethereum-alusta hyödyntää lohkoketjua, paras tapa ymmärtää lohkoketjujen toiminta oli ymmärtää alkuperäisen lohkoketjusovelluksen – BitCoin-sovelluksen – perustoiminnallisuus. Työn alussa tutustuttiin BitCoin-sovelluksen, lohkoketjutekniikan ja Ethereum-alustan perusteisiin. Tekniikasta tehtiin selvitykset, jotka kuvaillaan tässä.

## Bitcoin ja 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, joihin tiedonsiirtotapahtumat tallennetaan. Uuden kelvollisen lohkon löytäminen on sattumanvaraista, ja vaatii monia kokeiluja. Uusi tiedonsiirtotapahtuma päätyy lohkoketjuun seuraavasti (1):

1. Uusi tiedonsiirtotapahtuma lähetetään solmulta toiselle. Tiedonsiirron informaatio leviää koko verkon louhijasolmuille.
2. Kukin louhijasolmu kerää uusia tiedonsiirtotapahtumia lohkoon.
3. Kukin solmu yrittää laskea kelvollisen tiivisteen (engl. hash) lohkollensa.
4. Kun solmu onnistuu laatimaan 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 pysyy rehellisinä. Koska lohkoketjusta on kopio jokaisella verkon tietokoneella ja koska uuden lohkon tekeminen vaatii paljon laskentatehoa, lohkoon päätyneen tiedon muuttaminen jälkeenpäin on lähes mahdotonta. Jokainen louhijasolmu tarkistaa, onko lähettäjällä varaa tehdä rahansiirtoa. (1.)

Lohkon laskenut solmu saa palkkioksi Bitcoineja uuden lohkon luomisesta. Tähän palkkioon perustuu lohkoketjukonesalien liiketoiminta. Kun louhijasolmu onnistuu luomaan uuden lohkon, palkkioksi saatu kryptovaluutta voidaan myydä ostajalle, joka haluaa vaihtaa tavallista valuuttaa kryptovaluutaksi. (1.)

Epärehellinen louhijasolmu voisi yrittää huijata muita solmuja lähettämällä uuden 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.)

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. (3.)

## Hajautetut sovellukset

Ethereum ja Bitcoin ovat hajautettuja sovelluksia. 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. (4.)

Hajautetuissa sovelluksissa on monia hyötyjä verrattuna 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. (5.)

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 tulevat vertaisverkosta. (6.)

Bitcoinin historiassa ohjelmoijayhteisö alkoi kehittämään hajautettua alustaa, joka perustuu Bitcoin-verkkoon. Näille alustoille kehitettiin tukia monen eri tyyppisille tiedonsiirtotapahtumille. Ajan mittaan ohjelmoijayhteisö keksi uusia käyttötapauksia, jolloin olemassa olevien tiedonsiirtotapahtumatyyppien rajat tulivat vastaan. Silloin alustan kehittäjien täytyi taas luoda uusia tiedonsiirtotapahtumatyyppejä. Näin alustalla kehitettävät innovaatiot olivat riippuvaisia siitä, minkä tyyppisiä tiedonsiirtotapahtumia kehitettiin ohjelmoijayhteisössä. (7.)

## Ethereum-alusta

Bitcoin-verkossa voi ainoastaan lähettää valuuttaa tililtä toiselle. Ethereum-alustassa on myös tilejä, valuuttaa ja rahansiirtotapahtumia, mutta Ethereum-lohkoketjuun voi myös tallentaa koodia. Kehittäjät voivat kirjoittaa koodia, ja luoda omia hajautettuja sovelluksiaan. Ethereum-alustalle kehitettyjä sovelluksia kutsutaan *älykkäiksi sopimuksiksi*. (4; 8.)

Solidity on suosituin ohjelmointikieli Ethereum -älykkäiden sopimusten kirjoitusta varten. Muita suosittuja kieliä ovat muun muassa LLL ja Serpent. (5.)

Ethereum-solmuun, oli kyseessä yksityisen käyttäjän solmu tai louhijasolmu, kuuluu Ethereum-virtuaalikone, joka on älykkäiden sopimusten ajoympäristö. Ethereum-virtuaalikone voi ajaa koodia, joka suorittaa loogisia operaatioita. Ethereum-alusta luo vertaisverkon, jossa käyttäjän koneella käynnissä oleva virtuaalikone lähettää tiedonsiirtotapahtumia Ethereum-verkkoon. 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-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. (4, 7.)

Ethereum-virtuaalikone on Turing-täydellinen, joten virtuaalikoneella ei ole mitään loogisia rajoitteita liittyen siihen, mitä sillä voi tehdä. Turing-täydellisyys tarkoittaa sitä, että ohjelmointikielellä voi ratkaista minkä tahansa laskennallisen ongelman. Turing-täydellistä ohjelmointikieltä rajoittaa ainoastaan tietokoneen muistin määrä. (9.)

BitCoin-vertaisverkossa on bitcoin-rahayksikön siirtämistä varten tiedonsiirtotapahtumia. Ethereum-verkossa vastaava rahayksikkö on eetteri (engl. ether). Eetteri on bitcoinin kaltainen kryptovaluuttarahayksikkö, jota voi käyttää maksamiseen Ethereum-verkossa. (8.)

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 saman maksutapahtuman 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. (8; 7.)

Ethereum-maksutapahtumilla on maksimikaasurajoite, kuinka paljon ne saavat kuluttaa resursseja Ethereum-verkon louhijasolmuissa. Jos verkossa on sopimus, joka ajaa raskaita laskuoperaatioita, siihen tehdyt tiedonsiirtotapahtumat voidaan hylätä, jos niiden suorittaminen ylittää maksimikaasurajoitteen. Ethereum-verkon kaasurajoitteen takia verkko ei mene tukkoon, vaikka sinne lähetettäisiinkin älykkäitä sopimuksia, jotka vaativat liian raskasta laskentaa. (10; 7.)

Ethereum-virtuaalikone, kaikki siihen liittyvä ohjelmisto ja kaikki alustan kehittämiseen liittyvä ohjelmisto on avoimen lähdekoodin ohjelmistoa. Ethereum-ohjelmistoja kehitetään maailmanlaajuisesti ja hajautetusti GitHub-palvelun sekä muiden versionhallintapalveluiden avulla. (4.)

# SUUNNITTELU JA KÄYTTÖTAPAUKSET

Suunnitteluvaiheessa pohdiskeltiin työssä luotavan sovelluksen käyttötapauksia. Sovellus pyrittiin suunnittelemaan siten, että se hyödyntää mahdollisimman paljon lohkoketjujen ja Ethereum-alustan piirteitä.

## 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.

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 he saavat siitä pienen rahallisen korvauksen.

Kukin kohokohtapyyntö voi maksaa auton omistajalle useita euroja tai kymmeniäkin euroja jokaisesta merkinnästä. Merkinnät voivat silti todistaa auton arvokkuuden tulevalle ostajalle. Huutokaupassa tarjouksen tekijä saattaa tehdä suuren niiden merkintöjen perusteella, 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.

## Älykkään sopimuksen hyöty sovelluksessa

Autosovelluksen 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.

Toinen etu on lohkoketjumerkintöjen pitkäikäisyys. Klassisen auton elinkaari voi olla vuosikymmeniä, eivätkä kovin monet yritykset ole olemassa niin kauan. 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.

Lohkoketjuissa on piirre, että lisättyjä merkintöjä ja tiedonsiirtoja ei voi poistaa jälkikäteen. Epärehellinen autonomistaja ei siis pysty peittelemään jälkiään poistamalla tai muokkaamalla vanhoja merkintöjä. Tämä vahvistaa luottamusta sovelluksen käyttäjäkunnassa auton omistajan ja autosta kiinnostuneiden ostajien välillä.

# TYÖHÖN LIITTYVÄ OHJELMISTO

## Solidity-kääntäjä

Solidity-kääntäjä kääntää Solidity-kielen Ethereum-tavukoodiksi, jonka Ethereum-virtuaalikone muuntaa konekäskyiksi. Solidity-kieli on hyvin yksinkertainen ja helposti opittava. Kirjoitusasultaan 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. (11.)

## Browser Solidity

Browser Solidity on selaimessa käytettävä kääntäjä. Sen etuna on, että se toimii kaikilla tietokoneilla ilman ylimääräisten ohjelmien asennusta. (12.)

## 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. (13.)

Tätä työtä kehittäessä TestRPC:n asentaminen Windows-käyttöjärjestelmälle vaati ohjelmistoa, jota ei ollut valmiina Windowsissa. Asentamisessa käytettiin Microsoft Visual Studio Community Edition -ohjelman mukana tulleita resursseja. TestRPC:n kehittäjien mukaan 25.10.2017 Windows-asennus on helpottunut, eikä ylimääräisten resurssien asennusta vaadita enää. TestRPC on yhdistynyt Truffle-sovelluskehyksen (Ks. 4.5) ominaisuudeksi, ja ottanut nimekseen Ganache CLI. (14; 15.)

## Truffle-sovelluskehys

Truffle on Ethereumin älykkäiden sopimusten kehittämistä varten luotu sovelluskehys. Sen on tarkoitus helpottaa Solidity-kielellä älykkäiden sopimusten koodin kirjoittamista. Sovelluskehyksen avulla voi tehdä älykkäiden sopimusten jatkuvaa integraatiota Ethereum-lohkoketjuun. Trufflen avulla voi myös kirjoittaa automatisoituja testejä sopimuksille. (16.)

## 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. Meteorilla on helppo luoda reaktiivisia käyttöliittymiä, jolloin sovelluksen käyttöliittymä näyttää reaaliaikaisesti kaikki päivitykset, mitkä tulevat tietokantaan. (17.)

## 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. (18; 19.)

## Ethereum Wallet

Ethereum Wallet on helppokäyttöinen ohjelma, jolla käyttäjä voi vuorovaikuttaa Ethereum-verkon kanssa. Ohjelma käynnistää tietokoneelle geth-solmun automaattisesti, vaatimatta käyttäjältä ymmärrystä tekniikan yksityiskohdista. Ethereum Walletin graafisella käyttöliittymällä voi tehdä maksutapahtumia Ethereum-verkkoon. Ohjelmassa 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*, jota ei ole toteutettu TestRPC:ssä. (20.)

# TYÖN KULKU

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

Tutustuttiin Ethereumiin käyttämällä Ethereum Wallet -ohjelmaa. Työtä varten perustettiin git-repositorio (21). ClassicCarChain-älykkään sopimuksen koodaus aloitettiin. (Lopullinen ohjelmakoodi on liittessä 1.)

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.

Työssä kokeiltiin kehittää automaattisia testejä Truffle-sovelluskehyksellä. Automaattisten testien koodaus oli hyvin työlästä, joten niiden kehittämisestä luovuttiin myöhemmin. Työskentelyn ajan Trufflea käytettiin lähettämään (engl. *deploy*) sopimus TestRPC-testiverkkoon.

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.

Solidity-kielessä ole null-käsitettä. Jos funktiota GetHighlight kutsutaan kysyen kokonaislukuavaimella, jota ei vastaa mikään alustettu kohokohta, silloin funktio palauttaa Highlight-tietueen, jonka kaikki arvot ovat oletusarvoissaan. Muussa ohjelmointikielessä kutsu olisi saanut aikaan poikkeuksen. Solidity-kielen toteutus 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. Huoltotiedot (maintenanceData) ovat yksi tietueiden joukossa, mutta huoltotietoja ei käytetty lopullisessa sovelluksessa. (Kuva 1.)



KUVA 1.ClassicCarChain.sol-tiedostossa lopulliset tietueet, joita on kohokohdassa.

Vastaan tuli ongelma käyttötapauksessa ”vierailijan pitää kyetä näkemään kaikki kohokohdat, mitä autolla on”. Web3-rajapinnassa ei ole keinoa hakea kaikki mapping-tietorakenteen alkioita yhdellä funktiokutsulla. Ongelma ratkaistiin tekemällä Meteor-sovellukseen kiertokyselyrakenne, joka hakee yksitellen kaikki alkiot sopimuksesta. (Kuva 2.)

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 2. Kohokohdan haku älykkäästä sopimuksesta tiedostossa ClassicCarChain.sol.

Kiertokyselyrakenne on javascript-koodissa 03-eth-highlights.js (kuva 3). 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-funktiokutsu).



KUVA 3. Näyte tiedostosta 03-eth-highlights.js. Koodissa on kiertokyselyrakenne, jolla haetaan kaikki kohokohdat Meteor-sovellukseen.

Hyviä ohjelmistosuunnitteluperiaatteita pyrittiin seuraamaan suunnittelemalla sopimus uudelleen siten, että jokainen kohokohtamerkintä olisi Ethereum-lohkoketjussa oma alisopimuksensa, joihin ClassicCarChain-pääsopimus tekisi kutsuja. Suunnitelmasta jouduttiin 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). (22.)

Lopuksi tehtiin käyttöliittymää. Ethereum-yhteisön kehittämää dappstyles.styl-tyyliä kokeiltiin (23). Työssä päädyttiin tekemään oma yksinkertainen css-tyyli.

# LOPPUTULOKSET

Älykkään sopimuksen Solidity-koodi on liitteessä 1. Työn GitHub-repositorion lueminut-tiedostossa on ohjeita Meteor-sovelluksen käynnistämistä varten (21). Kuvassa 4 on auton omistajan näkymä sovelluksen etusivuilta. Yläpalkissa on navigointilinkit, oikeassa yläkulmassa aktiivisen tilin vaihtaminen. Yläpalkin alapuolella on lomake kohokohdan lisäämistä varten. Sininen laatikko on yksi kohokohta, jossa auton omistaja näkee poistamispainikkeen sekä tekstikentän, johon hän voi kirjoittaa syyn merkinnän poistamiselle.



KUVA 4. Lopullinen sovellus kohokohtasivulta.

## Toteutuneet käyttötapaukset

Työssä suunnitelluista käyttötapauksista toteutui suurin osa. Kaikki tärkeimmät käyttötapaukset, kuten kohokohtien ja -pyyntöjen lisääminen, toteutuivat.

* 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.
* Todennus 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. Kun omistaja hyväksyy kohokohtapyynnön, raha siirtyy sopimuksen saldosta pyynnön tekijälle. Jos sopimuksessa ei ole tarpeeksi saldoa, kohokohtapyyntöä ei voi hyväksyä. Tämä ominaisuus on sovelluksessa Ethereum-alustan turvallisuusrajoitteiden takia.
* Auton omistaja voi nostaa rahaa sopimuksen saldosta. Tätä varten on olemassa funktio sopimuksessa, sekä verkkosovelluksessa, mutta käyttöliittymää ei ole toteutettu. Rahan noston voi tehdä selaimen JavaScript-konsolilla.
* 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 hän 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.
* Lähes kaikki yllä mainittujen toimintojen teko tallentaa sopimukseen historiatietomerkinnän (Solidity-kielessä ”event”). Merkintöjä tallentuu myös silloin kun tietoja 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. Vähän huomiota saavan 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 käyttäjät ymmärtävät, 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 sovelluksella. 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 koodissa olevien ongelmien takia. Toteutus vaatisi Meteor-sovelluksen Javascript-koodin uudelleen suunnittelua. (Kuva 5.)



KUVA 5. 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. Tätä ominaisuutta suunniteltiin, mutta huomattiin, että se olisi ollut liian haastava toteuttaa. Solidity-kielessä on puutteita muuttuvan pituisille tietorakenteille, kuten taulukoille. Toteutettuna 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 kehitetty. Huoltotiedot suunniteltiin koodattavan aluksi vain viitenä huoltotehtävänä, joille kullekin voisi merkata huollon lopputuloksen. Huollon toimitukselle voisi merkata yhden neljästä tilasta: ei tarkistettu (NotChecked), tarkistettu (Checked), tarkistettu ja vika löydetty (CheckedAndFoundFault), vika löydettu ja korjattu (FoundFaultAndFixed):

**enum MaintenanceOutcome {**

**NotChecked, Checked, CheckedAndFoundFault,**

**FoundFaultAndFixed**

**}**

**struct MaintenanceTasks {**

**MaintenanceOutcome engine;**

**MaintenanceOutcome tires;**

**MaintenanceOutcome pedals;**

**MaintenanceOutcome brakes;**

**MaintenanceOutcome interior;**

**}**

Nykyinen järjestelmä vaatii auton 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 0.4.6. Solidity-kieli on hyvin 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 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 käyttötapauksia, mitä Ethereum-alustalla voi toteuttaa. Rajoitteet tulevat hidastamaan Ethereumin läpimurtoa yleiseen, maailmanlaajuiseen käyttöön.

# 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 (24). 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. Future Thinkers Podcast. Vitalik Buterin: What is Ethereum and How to Build a Decentralized Future. Saatavissa: <http://futurethinkers.org/vitalik-buterin-ethereum-decentralized-future/> Hakupäivä 22.11.2017
3. GO-Science 2016. Block chain technology. Saatavissa: <https://www.youtube.com/watch?v=4sm5LNqL5j0> Hakupäivä 22.11.2017
4. Swan, Melanie 2015. Blockchain. Kappale: Ethereum: Turing-Complete Virtual Machine. O'Reilly Media, Inc. Saatavissa: Safari Books Online (vaatii käyttöoikeuden). Hakupäivä 22.11.2017
5. Prusty, Narayan 2017. Building Blockchain Projects. Packt Publishing. Saatavissa: Safari Books Online (vaatii käyttöoikeuden). Hakupäivä 22.11.2017
6. Buterin, Vitalik 2015. Bitcoin 2.0 - Ideas and Applications. Bitcoinist.net. Luento. Saatavissa videolla: <https://youtu.be/Fjhe0MVRHO4?t=1m54s> kohdat 1:54-8:53. Hakupäivä 22.11.2017
7. Buterin, Vitalik 2016. DEVCON1: Understanding the Ethereum Blockchain Protocol. Ethereum. Luento. Saatavissa videolla: <https://youtu.be/gjwr-7PgpN8> Hakupäivä 27.10.2017
8. Ethereum 2017. Solidity Docs - Introduction to Smart Contracts. Read the Docs, Inc. Saatavissa: <https://solidity.readthedocs.io/en/latest/introduction-to-smart-contracts.html> Hakupäivä 25.11.2017.
9. Computerphile 2016. Turing Complete. Video. Saatavissa: <https://www.youtube.com/watch?v=RPQD7-AOjMI> Hakupäivä 25.09.2017.
10. Ethereum 2017. Ether, The crypto-fuel for the Ethereum network. Saatavissa: <https://ethereum.org/ether> Hakupäivä 22.11.2017
11. Ethereum 2017. Web3 JavaScript app API. GitHub, Inc. Saatavissa: <https://github.com/ethereum/wiki/wiki/JavaScript-API> Hakupäivä 22.11.2017
12. Ethereum-yhteisö 2017. Remix - Solidity IDE. Saatavissa: <https://remix.ethereum.org/> Hakupäivä 25.11.2017
13. Ethereum-yhteisö 2017. Ganache CLI. GitHub, Inc. Saatavissa: <https://github.com/trufflesuite/ganache-cli> Hakupäivä 22.11.2017
14. Ethereum-yhteisö 2017. Ganache CLI Wiki - Installing TestRPC on Windows. GitHub, Inc. Saatavissa: <https://github.com/trufflesuite/ganache-cli/wiki/Installing-TestRPC-on-Windows> Hakupäivä 25.11.2017
15. Visual Studio. Microsoft 2017. Saatavissa: <https://www.visualstudio.com/downloads/> Hakupäivä 22.11.2017
16. Consensys 2017. Truffle - Your Ethereum Swiss Army Knife. Saatavissa: <http://truffleframework.com/> Hakupäivä 22.11.2017
17. Meteor Development Group Inc. 2017. Meteor - The fastest way to build JavaScript apps. Saatavissa: <https://www.meteor.com/> Hakupäivä 22.11.2017
18. Ethereum-yhteisö 2016. Why are there multiple Ethereum clients? Read the Docs, Inc. Saatavissa: [http://ethdocs.org/en/latest/ethereum-clients/choosing-a-client.html](http://ethdocs.org/en/latest/ethereum-clients/choosing-a-client.html#why-are-there-multiple-ethereum-clients) Hakupäivä 22.11.2017
19. Ethereum-yhteisö 2016. Ethereum Clients, go-ethereum. Read the Docs, Inc. Saatavissa: <http://ethdocs.org/en/latest/ethereum-clients/go-ethereum/index.html> Hakupäivä 22.11.2017
20. Ethereum Foundation 2017. Smart money, smart wallet. Saatavissa: <https://ethereum.org/> Hakupäivä 22.11.2017
21. Miettinen, Lauri 2017. Classic Car Chain. GitHub, Inc. Työn GitHub-repositorio. Saatavissa: <https://github.com/Miettine/ClassicCarChain> Hakupäivä 22.11.2017
22. Ethereum-yhteisö 2017. Frequently asked questions - Can you return an array or a string from a solidity function call? Read the Docs, Inc. Saatavissa: <http://solidity.readthedocs.io/en/develop/frequently-asked-questions.html#can-you-return-an-array-or-a-string-from-a-solidity-function-call> Hakupäivä 22.11.2017
23. Ethereum-yhteisö 2017. Dapp styles. GitHub, Inc. Saatavissa: <https://github.com/ethereum/dapp-styles> Hakupäivä 22.11.2017
24. Gasteiger, Daniel 2016. Block chain demystified. TEDx Talks. Saatavissa: <https://www.youtube.com/watch?v=40ikEV6xGg4> Hakupäivä 22.11.2017

LIITTEET

Liite 1. ClassicCarChain.sol -älykkään sopimuksen Solidity-koodi.

pragma solidity ^0.4.6;

contract ClassicCarChain {

address public vehicleOwner;

string public vehicleModel;

uint public originBlockNumber;

uint public vehicleManufacturingYear;

uint public acceptedOfferAmount=0;

bool public ownershipBeingTransferred = false;

uint public contractBalance = 0;

function Withdraw(uint \_amount) OnlyByOwner public returns (bool){

if(msg.sender.send(\_amount)){

return true;

}

return false;

}

address public upcomingOwner;

function ClassicCarChain(string \_model, uint \_year) {

vehicleOwner = msg.sender;

//The one who created this contract to the network becomes the first vehicle owner.

originBlockNumber = block.number;

vehicleModel = \_model;

vehicleManufacturingYear = \_year;

ownershipBeingTransferred = false;

}

/// This index is used as an identifier of Highlights. It is incremented whenever a new highlight request is made.

uint public highlightIndex=0;

modifier OnlyByOwner() {

if (msg.sender == vehicleOwner) {

\_;

}

}

modifier NotByOwner() {

if (msg.sender != vehicleOwner) {

\_;

}

}

modifier OnlyByUpcomingOwner() {

if (msg.sender == upcomingOwner) {

\_;

}

}

modifier OnlyByOwnerOrUpcomingOwner() {

if (msg.sender == upcomingOwner || msg.sender == vehicleOwner) {

\_;

}

}

modifier OnlyIfOwnershipBeingTransferred(){

if (ownershipBeingTransferred) {

\_;

}

}

modifier OnlyIfOwnershipNotBeingTransferred(){

if (!ownershipBeingTransferred) {

\_;

}

}

function BeginOwnershipChange(address \_upcomingOwner, uint \_amount) private OnlyIfOwnershipNotBeingTransferred {

upcomingOwner = \_upcomingOwner;

acceptedOfferAmount= \_amount;

ownershipBeingTransferred=true;

}

function CancelOwnershipChange() public OnlyByOwnerOrUpcomingOwner OnlyIfOwnershipBeingTransferred{

if (upcomingOwner.send(acceptedOfferAmount)){

acceptedOfferAmount=0;

upcomingOwner=0;

ownershipBeingTransferred=false;

}

}

function AcceptOwnershipChange() public OnlyByUpcomingOwner OnlyIfOwnershipBeingTransferred {

if (vehicleOwner.send(acceptedOfferAmount)){

GiveVehicleOwnership(upcomingOwner);

acceptedOfferAmount=0;

upcomingOwner=0;

ownershipBeingTransferred=false;

}

}

/////////////////////////////////////////

mapping(uint => CCClib.Highlight) private highlights;

/////////////////////////////////////////

//mapping(uint => address) allOffers ;

mapping(uint => CCClib.Offer) private offers;

uint public offerIndex = 0;

function GetOffer(uint \_index) public returns (bool \_initialized,address \_maker, uint \_amount){

CCClib.Offer memory foundOffer = offers[\_index];

\_initialized = foundOffer.initialized;

\_maker = foundOffer.maker;

\_amount = foundOffer.amount;

}

event EOfferRemoved(address maker, uint amount, uint dateTime);

event EOfferRejected(address maker, uint amount, uint dateTime);

function RemoveOrRejectOffer(uint \_index) public returns (bool){

CCClib.Offer memory foundOffer = offers[\_index];

address sender = msg.sender;

bool senderIsVehicleOwner = sender == vehicleOwner;

bool senderIsOfferMaker = sender == foundOffer.maker;

if (senderIsVehicleOwner || senderIsOfferMaker){

if (foundOffer.maker.send(foundOffer.amount)){

delete offers[\_index];

if (senderIsVehicleOwner) {

EOfferRejected(foundOffer.maker, foundOffer.amount,now);

} else {

EOfferRemoved(foundOffer.maker, foundOffer.amount,now);

}

return true;

}

}

return false;

}

event EOfferAccepted(address maker, uint amount, uint dateTime);

function AcceptOffer(uint \_index) OnlyByOwner public returns (bool) {

CCClib.Offer memory foundOffer = offers[\_index];

if (foundOffer.initialized){

delete offers[\_index];

EOfferAccepted(foundOffer.maker, foundOffer.amount, now);

BeginOwnershipChange(foundOffer.maker, foundOffer.amount);

return true;

}

return false;

}

event EOfferMade(address maker, uint amount, uint dateTime);

function MakeOffer() NotByOwner public payable {

address sender= msg.sender;

CCClib.Offer memory newOffer = CCClib.Offer({id:offerIndex,

initialized:true,

maker:sender,

amount:msg.value});

offers[offerIndex]=newOffer;

offerIndex++;

EOfferMade(msg.sender, msg.value, now);

}

/////////////////////////////////////////

function AddNewToHighlights(CCClib.Highlight \_h) private {

highlights[\_h.id]=\_h;

highlightIndex += 1;

}

function GetHighlight(uint \_id) public returns (

uint \_highlightType,

bool \_initialized,

address \_maker,

uint \_requestCreationDateTime,

uint \_reward,

string \_message,

bool \_approvedToChain,

bool \_madeByOwner,

uint \_additionToChainDateTime

) {

CCClib.Highlight h = highlights[\_id];

\_highlightType = h.highlightType;

\_initialized = h.initialized;

\_maker = h.maker;

\_requestCreationDateTime = h.requestCreationDateTime;

\_reward = h.reward;

\_message = h.message;

\_approvedToChain= h.approvedToChain;

\_madeByOwner = h.madeByOwner;

\_additionToChainDateTime = h.additionToChainDateTime;

}

function AddHighlightAsOwner (string \_message) OnlyByOwner() public {

CCClib.Highlight memory h = CCClib.NewHighlight(highlightIndex, \_message);

AddNewToHighlights(h);

EmitEvent\_HighlightSavedToChain(h);

}

function MakeHighlightRequest(uint \_reward, string \_message) NotByOwner() public {

CCClib.Highlight memory h = CCClib.NewHighlightRequest (highlightIndex, \_reward, \_message);

AddNewToHighlights(h);

EmitEvent\_HighlightRequestMade(h);

}

function DeleteExistingHighlight(uint \_id, string \_reasonForDeletion) OnlyByOwner() public returns (bool) {

//Deleting a key in a mapping replaces the struct of that

//key with a struct posessing default-values.

EmitEvent\_HighlightDeleted(highlights[\_id], \_reasonForDeletion);

delete highlights[\_id];

}

function RejectHighlightRequest(uint \_id) OnlyByOwner() {

EmitEvent\_HighlightRequestRejected(highlights[\_id]);

delete highlights[\_id];

}

function AcceptHighlightRequest(uint \_id) OnlyByOwner() payable public returns (bool) {

//TODO: Find out if this function needs to have the payable-keyword.

//Is there some security restriction, that a contract cannot send funds if

// the message sender doesn't send them?

// Check if the owner actually has enough money.

CCClib.Highlight handledRequest = highlights[\_id];

// Send the money to the maker

contractBalance += msg.value;

uint requestedReward = handledRequest.reward;

uint difference = msg.value-requestedReward;

if ( handledRequest.maker.send(msg.value)) {

// //CCClib.PromoteHighlightRequest(highlights[\_id]);

//Disabled, because suspecting that this causes problems

handledRequest.approvedToChain=true;

handledRequest.additionToChainDateTime=now;

EmitEvent\_HighlightSavedToChain(highlights[\_id]);

return true;

}

return false;

}

function GiveVehicleOwnership(address \_newOwner) private {

if (\_newOwner!=vehicleOwner){

address oldOwner = vehicleOwner;

EVehicleOwnershipPassed( oldOwner, \_newOwner, now);

// The now-keyword returns the current block timestamp, as soon as this transaction finds its way into a mined block.

// I remember hearing that in the real Ethreum network, blocks are mined each 10 minutes. The timestamp is quite accurate.

vehicleOwner = \_newOwner;

}

}

/// Events

event EHighlightRequestMade(

uint highlightId,

address maker,

uint requestCreationDateTime,

uint requestedReward,

string message

);

function EmitEvent\_HighlightRequestMade (CCClib.Highlight \_h) private {

EHighlightRequestMade(\_h.id, \_h.maker, \_h.requestCreationDateTime, \_h.reward, \_h.message);

}

event EHighlightSavedToChain(

uint highlightId,

address maker,

uint requestCreationDateTime,

uint paidReward,

string message,

bool madeByOwner,

uint additionToChainDateTime

);

function EmitEvent\_HighlightSavedToChain (CCClib.Highlight \_h) private {

EHighlightSavedToChain(\_h.id, \_h.maker, \_h.requestCreationDateTime, \_h.reward, \_h.message, \_h.madeByOwner, \_h.additionToChainDateTime);

}

///

event EHighlightRequestRejected(

uint rejectionDateTime,

uint highlightId,

address maker,

uint requestCreationDateTime,

uint requestedReward,

string message

);

function EmitEvent\_HighlightRequestRejected (CCClib.Highlight \_h) private{

EHighlightRequestRejected(now, \_h.id, \_h.maker, \_h.requestCreationDateTime, \_h.reward, \_h.message);

}

event EHighlightDeleted(

uint deletionDateTime,

string reasonForDeletion,

uint highlightId,

address maker,

uint requestCreationDateTime,

uint paidReward,

string message,

bool madeByOwner,

uint additionToChainDateTime

);

function EmitEvent\_HighlightDeleted (CCClib.Highlight \_h, string \_reasonForDeletion) private{

EHighlightDeleted(now, \_reasonForDeletion, \_h.id, \_h.maker, \_h.requestCreationDateTime, \_h.reward, \_h.message, \_h.madeByOwner, \_h.additionToChainDateTime);

}

event EVehicleOwnershipPassed(address oldOwner, address newOwner, uint dateTime);

}

library CCClib {

struct Offer{

uint id;

bool initialized;

address maker;

uint amount;

}

struct Highlight{

uint id;

bool initialized;

uint highlightType;

address maker;

uint requestCreationDateTime;

uint reward;

string message;

bool approvedToChain;

bool madeByOwner;

uint additionToChainDateTime;

MaintenanceTasks maintenanceData;

}

struct MaintenanceTasks {

bool engine;

bool tires;

bool pedals;

bool brakes;

bool interior;

}

function CreateMaintenanceData (bool[] \_status) internal returns (MaintenanceTasks) {

if (\_status.length != 5){

throw;

}

MaintenanceTasks memory tasks = MaintenanceTasks({

engine:\_status[0],

tires:\_status[1],

pedals:\_status[2],

brakes:\_status[3],

interior:\_status[4]

});

return tasks;

}

enum HighlightTypes {

Review,

Maintenance

}

//https://ntgroup.studio.crasman.fi/pub/web/vianor/pdf/Vianor\_perushuolto\_plus.pdf

function GetMaintenanceStatus(Highlight \_h) internal returns (

bool \_eng,

bool \_tires,

bool \_pedals,

bool \_brakes,

bool \_interior

){

MaintenanceTasks memory data = \_h.maintenanceData;

\_eng = data.engine;

\_eng = data.tires;

\_eng = data.pedals;

\_eng = data.brakes;

\_eng = data.interior;

}

function NewHighlightRequest (uint \_id, uint \_reward,string \_message)internal returns ( Highlight){

Highlight memory h;

h.id=\_id;

h.initialized=true;

h.highlightType = uint(HighlightTypes.Review);

h.maker= msg.sender;

h.requestCreationDateTime=now;

h.reward=\_reward;

h.message=\_message;

h.approvedToChain=false;

h.madeByOwner=false;

return h;

}

function NewHighlight (uint \_id, string \_message) internal returns ( Highlight){

Highlight memory h;

h.id=\_id;

h.initialized=true;

h.highlightType = uint(HighlightTypes.Review);

h.maker= msg.sender;

h.requestCreationDateTime=now;

h.reward=0;

h.message=\_message;

h.approvedToChain=true;

h.madeByOwner=true;

h.additionToChainDateTime = now;

return h;

}

function NewMaintenanceHighlightRequest (uint \_id, uint \_reward,string \_message, uint[] \_maints, uint[] \_status) internal returns ( Highlight){

Highlight memory h = NewHighlightRequest(\_id, \_reward, \_message);

h.highlightType= uint(HighlightTypes.Maintenance);

//h.maintenanceData = CreateMaintenanceData( \_status) ;

return h;

}

function NewMaintenanceHighlight (uint \_id, string \_message, uint[] \_maints, uint[] \_status) internal returns ( Highlight){

Highlight memory h = NewHighlight(\_id, \_message);

h.highlightType= uint(HighlightTypes.Maintenance);

//h.maintenanceData = CreateMaintenanceData(\_status);

return h;

}

function PromoteHighlightRequest ( Highlight storage h) internal {

h.approvedToChain=true;

h.additionToChainDateTime=now;

}

}