TEKNILLINEN TIEDEKUNTA

LOHKOKETJUTEKNOLOGIA TERÄS-RAKENTEISIIN LIITTYVÄN DIGITAALISEN TIEDON VARMISTAMISESSA

Mikko Paavola

KONETEKNIIKAN TUTKINTO-OHJELMA

Diplomityö 2018

TIIVISTELMÄ

Lohkoketjuteknologia teräsrakenteisiin liittyvän digitaalisen tiedon varmistamisessa
Mikko Paavola
Oulun yliopisto, Konetekniikan tutkinto-ohjelma
Diplomityö 2018, 80 s.
Työn ohjaaja: TkL, IWE Timo Kauppi

Diplomityössä tarkastellaan teräsrakenteiden, erityisesti hitsattujen teräsrakenteiden tuotannossa ja niiden koko elinkaaren aikaisessa käytössä syntyvän dokumenttiaineiston digitaalista tallentamista, siirtoa verkossa ja muuta käyttöä. Dokumenttiaineisto voi käsittää automaattisesti tai manuaalisesti tuotettuja tekstejä, kuvia, kaavioita, piirustuksia ja auditiivisia tallenteita. Dokumentteja hallitaan dokumenttialustalla, jonka suunnitteluperusteet esitetään tässä työssä.

Työ on menetelmätutkimus. Sovellettavaksi menetelmäksi on valittu lohkoketju-teknologia. Sitä ja sen mahdollisuuksia tarkastellaan varsinaisen lohkoketjun lisäksi myös kyseiseen teknologiaan läheisesti liittyvien tai liitettävien tietoteknisten ratkaisujen kautta. Näitä ovat muun muassa älykäs automaatio ja tiedonsiirron varmistamismenetelmät. Tietoturvaongelmien ratkaiseminen on keskeisessä asemassa.

Dokumenttialusta perustuu tietokoneohjelmalle, jonka ohjelmointiperiaatteet ja -menetelmät esitetään ja perustellaan. Oleellisia elementtejä ovat muun muassa hajautettu tietokanta, funktionaalinen ohjelmointi ja asynkroninen tiedonsiirto. Menetelmien soveltamisessa on toki lähdetty oppikirjamaisista ja tavanomaisista ratkaisuista, mutta osin niiden rajoittaviin ominaisuuksiin on suhtauduttu kriittisesti ja uusiin tapoihin pyrkien. Lähtökohtana on itse ongelma, tässä tapauksessa digitaalisen tiedon tietoturvallinen tallentaminen ja jakaminen, jonka tarvitsemat ratkaisut määrittelevät käytettävät ohjelmistokielet ja -työkalut.

Vaikka työn lähtökohtana on terästeollisuuden digitalisaatio, esitettyjen ratkaisujen perustana oleva tietokoneohjelma on geneerinen. Valituille menetelmille antaakin vahvan tukensa se, että työn tekijä on ne jo toteuttanut useissa integrointiohjelmissa, joita on otettu käyttöön kymmenissä yrityksissä globaalisti.

Asiasanat: dokumentaatio, lohkoketju, teräsrakenne, tiedonsiirto, tietoturva

ABSTRACT

Block chain technology for securing digital information on steel structures
Mikko Paavola
University of Oulu, Degree Programme of Mechanical Engineering
Master's thesis 2018, 80 p.
Supervisor: Lic. Sc. (Tech.), IWE Timo Kauppi

The thesis investigates the secured digital data recording, transferring and delivering in particular welded steel structures. The entire life cycle of the steel structures is included. Document material can be automatically or manually generated texts, images, charts, drawings, and audio recordings. Documents are managed on a document platform, whose design basics are presented in this work.

The work is a methodology study. The applied technology is a block chain technology. IT solutions closely related to the block chain technology are also studied. These IT solutions include for example intelligent automation and data transfer methods. Solving related security problems is a key factor through the study.

Programming principles and methods are presented and justified in this work. Essential elements include decentralised database, functional programming and asynchronous data transfer. The application has been based on conventional solutions, but some of their restrictive features have been critically addressed and sought new ways using the latest scientific research. The starting point is the problem itself: a data safe storage and sharing of the digital information. The solution is determined before the selection of the software languages, other programming tools and operating systems.

The work is based on the digitalisation of the steel industry. The computer program underlying the proposed solution is generic and therefore useful to any other branch of the industry. The selected methods are strongly supported by the fact that the author has already implemented similar methods in a number of integration programs that have been introduced in dozens of companies globally.

Keywords: block chain, data transfer, documentation, security, steel structure

ALKUSANAT

Työuraa ohjelmistotuotannossa tehneenä minulle vihdoinkin tarjoutui tilaisuus saattaa loppuun DI-opintoni. Digitalisaation tarve konetekniikassakin on mahdollistanut käytännön työkokemukseni ja konetekniikan yhdistämisen diplomityön muodossa.

Tehtävänä on ollut hahmotella toteutettavan mallin perusperiaatteet teräsrakenteisiin, nimenomaan hitsattuihin teräsrakenteisiin liittyvien dokumenttien tietoturvalliseen säilyttämiseen ja jakeluun. Aihetta tarjottiin helmikuussa 2018 Lapin ammattikorkea-koulusta ArcDigi-projektiin liittyvänä taustoitustyönä. Projektin päämääränä on hyödyntää digitalisaatiota teräsrakenteiden toteutuksessa ja laadunhallinnassa.

Kiitän työni ohjaajaa, TkL, IWE Timo Kauppia ja muita Lapin ammattikorkeakoulun ArcDigi-projektiin osallistujia hankkeen perustietojen tarjoamisesta, ohjaajaa erikseen rakentavista kommenteista sekä DI Matti Uusimäkeä ohjaamisesta kyseisen projektin pariin. Oulun yliopiston konetekniikan aiemmasta henkilökunnasta erityisen kiitoksen ansaitsevat tietokoneiden maailmaan minut aikoinaan ohjanneet prof. Antti Pramila ja prof. Jussi A. Karjalainen. Nykyisistä konetekniikan henkilökunnan jäsenistä varsinkin Hannu Liedes, Jouko Heikkala, Juho Alatalo, Jari Laukkanen, Tapio Korpela, Reijo Saari ja Juhani Niskanen ovat joutuneet kuuntelemaan innostunutta digitalisaatio-puhettani. Kiitän heitä keskusteluista, jotka ovat muokanneet ajattelutapaani monella eri tavalla.

Erityisesti kiitän työni valmistumista kärsimättömästi odottanutta perhettäni. Vaimoni, FT ja useiden opinnäytetöiden ohjaaja, on toiminut kierroslukurajoittimena ja korostanut kirjoitusprosessin kurinalaisuutta innovatiivista ajattelua väheksymättä. Tyttäreni, XR-asiantuntija, on keskusteluissamme avartanut näkymiäni ICT-maailmaan ja puhuu isänsä kanssa samaa kieltä.

Oulussa 28.8.2018

Mikko Paavola

SISÄLLYSLUETTELO

MERKINNÄT JA LYHENTEET

ABI	Application Binary Interface
API	Application Programming Interface
CCDCOE	NATO Cooperative Cyper Defence Centre of Excellence
FHE	Fully Homomorphic Encryption
FIPS	Federal Information Processing Standards
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IIoT	Industrial Internet of Things
IoT	Internet of Things
IP	Internet Protocol
MTBF	Mean Time Between Failures
NTP	Network Time Protocol
PGP	Pretty Good Privacy
RFC	Request for Comments
SPF	Single Point Failure - Nasan käyttämä lyhenne
SPoF	Single Point of Failure
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
Unicode	Universal Character Set (UCS) ISO/IEC 10646
WoT	Web of Trust
ZKP	Zero-Knowledge Proof

1 JOHDANTO

Bruce Schneier, tietoturva-asiantuntija Harvardin yliopistosta, oli Helsingin Sanomien toimittajan haastateltavana Naton kyberturvallisuuskeskuksen CCDCOE:n CyCon X 30.5. -1.6.2018 konferenssissa Tallinnassa. Toimittajan mukaan hän totesi ajankohtaisiin tietoturvaongelmiin seuraavasti: "Meidän pitäisi ryhtyä suunnittelemaan laitteita ja järjestelmiä turvallisuus etunenässä". (Halminen 2018) Schneierin näkemystä myötäillen tässä diplomityössä ensisijainen huomio kiinnitetään suunniteltavan järjestelmän tietoturvallisuuteen (Schneier 2003, s. 271).

Työn aiheena on teräsrakenteiden, erityisesti hitsattujen teräsrakenteiden, suunnittelussa, tuotannossa, asentamisessa ja ylläpidossa syntyvän digitaalisen dokumenttiaineiston hallinta ja varmistaminen yleisen tietoverkon avulla. Hallinta annetaan dokumenttialustalle, jonka suunnitteluperiaatteet hahmotellaan tässä työssä. Alustalle on tarkoitus tallentaa manuaalisesti tai automaattisesti syntyvä digitaalinen materiaali teräsrakenteiden koko elinkaaren ajalta tarjouspyynnöstä lähtien. Lyhyyden vuoksi työssä käytetään termiä "dokumenttialusta" "suunniteltavan dokumenttialustan" sijaan.

Dokumenttialustan on hallittava eri lähteistä ja toimijoilta tulevan tiedon muuttaminen jaettavaan muotoon. Se myös huolehtii dokumenttien automaattisen ja ajantasaisen jakelun asianomaisille tahoille. Samoin se huolehtii aineistoon pääsyn sallimisen ja rajoittamisen, sekä tavan säilöä tietoa loogisiin tietovarantoihin tietoturvallisesti. Tässä työssä ei pohdita tieto-, informaatio- ja data-termien merkityseroja. Suurimmaksi osaksi niitä käytetään synonyymisesti.

Aihepiiriä tarjottiin Lapin ammattikorkeakoulusta. Sieltä myös sain tutustumista varten esimerkkejä digitaalisista dokumenteista. Vaikka tässä diplomityössä lähtökohtana ovat nimenomaan terästeollisuuden vaatimukset, on syytä korostaa työssä esitetyn tieto-teknisen ratkaisun yleispätevyyttä. Perustana dokumenttialustalle on vuosikausia omassa yrityksessäni kehittämäni ja kymmenissä sovelluksissa käytetty integrointityökalu, joka on sovitettu tekemään tiedonsiirtoa hyvin erilaisten ympäristöjen välillä automaattisesti ja tietoturvallisesti ilman ylläpitohenkilöstöä. Työ-kalun sisäinen rakenne antaa mahdollisuudet käyttää uusia tehokkaita keinoja testata

määrittelemättömien tietovirtojen hallintaan tehtyjä hajautettuja järjestelmiä tuotantoympäristöissä varsinaista tuotantoa häiritsemättä ja hidastamatta.

Tutkimuskysymyksiin vastataan yleensä kolmen toisiinsa sidoksissa olevan tekijän pohjalta. Nämä ovat teoria, menetelmät – voidaan kutsua myös tutkimusmetodiksi – ja aineisto eli data. Koska nämä tekijät painottuvat eri tavoin ja ovat sisällöllisestikin varioivia eri tieteenaloilla, jopa tieteenalan sisälläkin, ne vaativat hieman määrittelyä.

Teknillisissä tieteissä teoria käsitetään yleensä käytännön vastakohtana tai käytäntöä perustelevana tietona. Siinä merkityksessä se esiintyy myös tässä työssä. Toisaalta teorialla voidaan tarkoittaa kattavaa paradigmaa tai koko työtä ohjaavaa näkökulmaa, sitä miten tutkimusongelmaa lähestytään. Näkökulmavalintani on lähinnä heuristinen analyysi. Se tarkoittaa, että ratkaisua haetaan toisaalta asiantuntijoiden tarjoamasta tiedosta ja ajatuksista kirjallisen lähdeaineiston kautta, toisaalta omasta pitkän työuran aikana kertyneestä kokemuksesta eli esiymmärryksestä tai formuloimattomasta esitiedosta, jota osin voi nimittää intuitioksikin. Tieteenteoreettisesti tätä hieman epä-formaalia lähestymistapaa voidaan luonnehtia myös etenemiseksi hermeneuttisella kehällä (Niiniluoto 1983, s. 176), varsinkin kun tutkimuksen eteneminen tapahtuu dialogina, johon osallistuvat teoriataustoituksen ja työntekijän lisäksi myös data, tieto-tekninen ympäristö, tietoturvavaatimukset ja muut ratkaisuun vaikuttavat tekijät.

Menetelmät puolestaan käsittävät ne keinot, joilla dokumenttialustaa rakennetaan toimivaksi järjestelmäksi. Pääpaino on lohkoketjuteknologiassa ja siihen läheisesti liittyvissä menetelmissä. Dokumenttialustan suunnitteluperiaatteet edellyttävät menetelmien laajahkon teoreettisen taustoituksen. Dokumenttialustan tietoteknisenä toimintaympäristönä käytetään hyperdivergoitua infrastruktuuria. Se käsittää kaikki sähköiseen kommunikointiin kykenevät laite- ja ohjelmistokokonaisuudet. Olemassa olevia ohjelmistostandardeja ja -protokollia ei muuteta, mutta sivuutetaan osittain. Normaalisti sovelluskehityksestä piilotettuja tai huomiotta jätettyjä osia nostetaan tarkastelun kohteeksi, koska niiden vaikutus ohjelman luotettavuudelle ja tieto-turvallisuudelle on hyvin suuri. Näiden osien käsittelyssä käytetään tekoälyä soveltavasti tavoitteen saavuttamiseksi.

Koska työn rakenteessa painottuu dialogi, erillistä kirjallisuuskatsauslukua ei ole. Tekijän argumentaation tueksi tai perustaksi käydään kriittistä keskustelua aiemman tutkimuksen kanssa yksittäisten tutkimuskysymysten käsittelyn yhteydessä.

Lopputulemana pyritään esittämään dokumenttialustan suunnitteluun tarvittavat periaatteet, joissa fokus on nimenomaan tietoturvakysymyksissä. Alussa siteerattu Schneier esitti jo 20 vuotta sitten monipuolisesti ja esimerkein, miten turvallisempi tietojärjestelmä olisi mahdollista rakentaa, ja mitä pitäisi ottaa huomioon sitä suunniteltaessa. Samalla hän myös totesi, että ohjelmistosuunnittelijat tekevät samoja vanhoja suunnitteluvirheitä turvallisuuden suhteen yhä uudestaan ja uudestaan. Hän myös määritteli turvallisen järjestelmän olevan sellainen, ettei se pelkästään torju kaikkia olemassa olevia uhkia, vaan myös tulevaisuuden tietoturvauhkia. (Schneier 1998) Tässä työssä ei ole erillistä tietoturvaan liittyvää lukua, koska tietoturvakysymykset tulevat esille ja ovat sisäänrakennettuina lähes kaikissa käsittelyluvuissa.

2 TERÄSRAKENTEET

2.1 Teräsrakenteiden dokumentointitarve

Teräksen määritelmät tai katsaus teräksestä tehtäviin rakenteisiin eivät ole tässä työssä oleellisia tutkimuskysymyksiä, vaan sen perusteleminen, miksi teräsrakenteiden luotettava dokumentointi on tärkeää ja miten tuo vaadittu luotettavuus voidaan saavuttaa. Tarkastelun kohteena ovat kuitenkin tapaustutkimuksen tapaan nimenomaan hitsatut ja yksittäistuotetut teräsrakenteet niihin liittyvien digitaalisten dokumenttien kautta. Teräsrakenteiden hitsaustapahtuman dokumentointi on tärkeää, koska sen avulla varmistutaan oikeiden kappaleiden yhteen liittämisestä, hitsiliitoksen suunnitelman mukaisesta tekemisestä ja tehdyistä korjaustoimenpiteistä virheiden ja puutteiden jälkeen. Taustalla vaikuttaa luonnollisesti, mihin rakenteisiin eri teräslaadut soveltuvat parhaiten ja mitä suunnitteluparametreja on käytetty rakenteen suunnittelussa, valmistuksessa ja asentamisessa. Dokumentoinnin tarve lisääntyy, kun samaan rakenteeseen tai kohteeseen käytetään useita erilaisia ja eri tavoin käsiteltyjä teräslaatuja.

Sekä suunniteltava rakenne itse että teräslaji tai -laatu, käyttötarkoitus, maantieteellinen ympäristö, kansalliset vaatimukset ja lopulta asiakkaan vaatimat asiakirjat vaikuttavat koko prosessin dokumenttien määrään ja sisältöön. Dokumentit käsittävät siis laajan ja heterogeenisen kokoelman teräsrakenteisiin liittyvää digitaalista tietoa. Tässä työssä tavoitteena on yksinkertaistaa dokumenttien tallennusta ja hallintaa. Tämä hyödyttää nimenomaan teräsrakenteiden vastuullisia tuottajia. Lisäksi kerätty ja ylläpidetty tieto mahdollistaa eri prosessien osien kehittämisen.

Dokumentointitarve kulminoituu siihen, että teräsrakenteiden loppukäyttäjä hyötyy dokumentoinnista. Se edellyttää, että teräsrakenteiden suunnittelun, rakentamisen, käytönaikaisiin ja käytöstä poistamisen dokumentteihin voidaan luottaa. Loppu-käyttäjällä tässä tarkoitetaan sekä yksityisiä että yhteiskunnallisia tahoja, joilla on valvontavelvollisuus esimerkiksi rakennemääräysten toteutumiseen ja infrastruktuurin yleiseen turvallisuuteen.

2.2 Dokumentit hitsattujen teräsrakenteiden laadun varmistamisessa

Teollisessa tuotannossa laadulle on monta määritelmää, näkökulmasta riippuen osin keskenään ristiriitaisiakin. Laatua ovat esimerkiksi: asiakkaan tarpeen huomioiminen, kestävyys, tasalaatuisuus, valmistuksen suunnitelmallisuus, ohjeet, suorituskyky, hinta-laatu -suhde, nollavirheellisyys, toleranssit, tuotannonpoikkeamat, hävikki, vaihtoarvo, viimeistely, esteettisyys ja jopa statuksellisuus. (Martikainen 2013, s. 74; Anttila & Jussila 2016) Dokumenttialusta mahdollistaa luotettavat lähtötiedot muun muassa edellä mainittujen - tai ainakin osaan niistä - asioiden tarkasteluun, mittaamiseen ja edelleen kehittämiseen ja sitä kautta lopputuotteen laadun parantamiseen.

Dokumentit eivät siis välttämättä kerro suoraan itse lopputuotteen laadusta. Sen sijaan ne voivat kertoa prosessin laadusta, jolla tuote suunnitellaan, valmistetaan, asennetaan ja käytetään. Parhaimmillaan automaattisia tai manuaalisia dokumentteja luodaan jatkuvasti tuotteen koko elinkaaren aikana.

Myös ulkopuoliset satunnaiset tekijät, esimerkiksi sähkökatkot, sääolosuhteet, huolto-toimenpiteet, laiteviat ja työtapaturmat ovat tallennettavia tietoja. Ne auttavat selvittämään jälkikäteen, millaisessa ympäristössä suunniteltu prosessi toimi. Vastuu-kysymyksien jälkikäteisessä selvittelyssä oleellista on osoittaa dokumenttien avulla kunkin tahon riittävän huolellinen ja ammattimainen toiminta. Laatuna voidaan pitää lisäksi sitä, että kohteen tunnistaminen on mahdollista automaattisesti esimerkiksi itse kappaleisiin lisätyn tunnistetiedon avulla, joka mahdollistaa materiaalien internetin (Kaksonen 2018).

Hitsin silmämääräisen tarkastelun lisäksi digitaalisina dokumentteina tallennetaan kuvat tai kuvanauhoitus hitsatuista liitoksista. Tällöin voidaan varmistua heti siitä, että silmä-määräinen tarkastus on suoritettu ja vieläpä ammattimaisesti. Samalla voidaan todistaa, jos hitsattuun liitokseen on tehty jälkikäteen muutoksia tai korjauksia, joita ei ole dokumentoitu asianmukaisesti. Kuvat ovat siis luotettavan dokumentoinnin rekursiivista varmistamista.

Jos kaikkia hitsejä ei tarkisteta esimerkiksi magneettijauhe-, tunkeumaneste-, ultraääni- tai röntgentarkastuksella (Leino 2006, s. 55) on järkevää tarkistaa erityisesti ne liitokset, joiden hitsauksen aikana tapahtui dokumenttien perusteella jotain poikkeuksellista.

Samoin tarkistuksia voidaan kohdistaa niihin kohteisiin, joita ei ole tarkistettu ja dokumentoitu aikaisemmassa työvaiheessa.

Laadun varmistamista tarkastellaan tässä työssä siis ainoastaan dokumenttien näkökulmasta. Kun tietyt vaaditut dokumentit ovat olemassa, voidaan lopputuotteen laadusta olla varmempia kuin oltaisiin ilman kattavaa ja tallennettua dokumentaatiota.

Automaattisesti syntyvät dokumentit ovat luotettavuudeltaan varmempia kuin ihmisten laatimat dokumentit, joihin ei esimerkiksi ole olemassa selkeää dokumenttilomaketta tietojen syöttöä varten. Dokumentteihin liittyy siis epävarmuustekijöitä, varsinkin inhimillisiä virheitä. Toisaalta lopputuote voi olla laadultaan erinomainen, vaikka sen dokumentaatio olisi puutteellinen. Tällainen tapaus on kuitenkin sattumanvarainen, eikä perustu systemaattiseen toimintaan, eikä niin ollen kumoa dokumentointitarvetta.

3 TERÄSRAKENTEIDEN TUOTANTOON JA KÄYTTÖÖN LIITTYVÄT DOKUMENTIT

3.1 Taustaa

Tässä luvussa käsitellään nimenomaan hitsattujen teräsrakenteiden elinkaareen liittyviä dokumentteja; työn lähtökohtanahan oli juuri hitsattujen terästuotteiden dokumentaatio. Suunnitellun dokumenttialustan kattavuus on tosin paljon sitä laajempi. Eräin osin myös dokumentointikäytännöt eikä pelkästään dokumenttialustaohjelma soveltuu mihin tahansa teräsrakenteiden tuotantoon ja käyttöön yksittäisessä yrityksessä, yritysten muodostamassa alihankintaketjussa tai yritysverkossa.

Hitsattujen teräsrakenteiden koko elinkaaren aikaisessa dokumenttituotannossa oleellista on luonnollisesti, mistä valmistusprosessin ja käytön aikaisista yksityiskohdista dokumentteja on syytä tuottaa. Relevanttia on pohtia myös, mistä dokumentteja ei ole sovittu laadittavan lainkaan ja millä perusteella.

Aikaisemmin kustannustekijät ovat olleet yksi tärkeimmistä rajoittavista tekijöistä. Dokumenttialustan suunnittelussa tämä tekijä on pyritty minimoimaan. Tätä edesauttaa tallennus- ja muu tietokonelaitekehitys samoin olemassa olevien tietovarastojen suora hyödyntäminen. Lisäksi nykyisin monet tietokoneohjelmat automaattisesti tuottavat tallennettavaa tietoa, esimerkiksi IoT (Verronen et al. 2016; Kemppi 2017). Jatkuvasti kehittyvät anturit tuottavat muuhun järjestelmään integroitavaa tietoa käyttämällä hyväkseen jo järjestelmässä olevaa tietoa (Pervilä 2018). Antureiden avulla saadaan digitaalista tietoa laitteiden toimivuudesta ja jo etukäteen viitteitä toimimattomuudesta (IIoT) (Pittman 2017). Antureiden laskentakapasiteetin lisäys mahdollistaa reunalaskennan (Edge Computing) hyödyntämistä, mikä tarkoittaa, että hajauttamalla laskentaa järjestelmän kokonaissuorituskyky kasvaa (Tseng et al. 2018, s. 8 - 19). Tallennettavien dokumenttien määrällä ei siis ole ratkaisevaa merkitystä kustannusten kannalta.

Dokumentit ovat pääosin sanallisia tai kuvallisia, siis visuaalisia. Visuaalisuuden ei kuitenkaan tarvitse olla ainoa kriteeri. Yhtä hyvin dokumenteiksi voidaan käsittää myös esimerkiksi konepajaan asennettujen mikrofonien kautta saatu auditiivinen tallenne.

Mikrofonit voidaan asentaa siten, että nauhoituksessa saadaan äänilähteen 4D-informaatio selville. Tallennettu tieto voi kertoa sellaisista työvaiheista, joista ei ole muuta dokumenttia, esimerkiksi viimeistelyssä tai akuutissa prosessin aikaisessa korjaamisessa käytetystä "moukaroinnista". Tietoa voidaan käyttää myöhemmin päätettäessä pintavaurioiden ja säröjen satunnaistarkastuksista. (Lepola 2009, s. 372)

Dokumenttialustan suunnittelussa dokumenttien sisällöllä, muodolla tai määrällä ei ole merkitystä. Siksi dokumentteja ei tässä työssä käsitellä yksityiskohtaisesti. Vasta yrityskohtaisessa pilotointivaiheessa dokumenttien merkitys kasvaa sovelluskohtaisesti.

3.2 Tuotannosta vaaditut dokumentit

3.2.1 Asiakkaiden vaatimat dokumentit

Asiakkaan kanssa tehdään sopimus, jossa määritellään toimituksen yhteydessä muun muassa toimitettavat dokumentit, noudatettavat yleissopimukset ja standardit. Tämä sopimus on myös dokumentti, joka voidaan tallentaa dokumenttialustaan. Toimituksen sisältö saattaa muuttua yhteisestä sopimuksesta tilauksen jälkeenkin. Tällöin dokumenttialusta tarjoaa paikan tallentaa tehdyt sopimusmuutokset tavalla, joka vastaa kysymykseen, ketkä ovat sopineet, milloin ja mitä on sovittu muutettavaksi.

Dokumenttialustaa käyttävällä asiakkaalla on mahdollisuus seurata tilauksensa etenemistä toimittajalla reaaliaikaisesti. Samoin asiakkaan toimituksen jälkeen tekemät tai teettämät korjaukset ja muutokset tulevat toimittajalle tiedoksi automaattisesti, jos ne on tallennettu dokumenttialustaan.

3.2.2 Standardien vaatimat dokumentit

Tarkoituksena ei ole luetella hitsattavien teräsrakenteiden kansallisia tai kansainvälisiä standardisoimisjärjestöjen luomia standardeja. Standardit ovat jatkuvan evaluoinnin alaisina kansallisten ja kansainvälisten lakien ja säännösten muutosten vuoksi (Kauppi 2013, s. 33; Hurula & Kauppi 2015, s. 9). Koska standardit määrittävät yhteisiä toimintatapoja ja toisaalta toimintatapamuutokset generoivat uusia standardeja, standardien huomioon ottaminen on oleellista ja muutoksia on seurattava.

Standardit ovat dokumentteja, jotka voidaan tallentaa yksinkertaisimmin dokumenttialustan käyttäjää imitoivalla automaattisella robottisyöttöohjelmalla. Dokumenttialustan kannalta on yhdentekevää, onko standardit tallennettu erillisinä tiedostoina hakemistorakenteessa tai dokumenttitietokantana. Dokumenttialusta ei rajoita millään tavalla dokumentin eri suojaustapojen käyttöä. Suojaustapoja voidaan joutua käyttämään tekijänoikeuksellista syistä dokumentin tiedostomuodon esitysohjelmassa. Minimissään dokumenttialusta tarjoaa tiedon muuttuneista standardeista kaikille, jotka ovat jo aikaisemmin tarvinneet kyseistä standardia tai sen muuttumistietoa.

3.2.3 Viranomaisten vaatimat dokumentit

Säädellyllä toimialalla pitää olla lupa toimia eli lupadokumentti. Luvan hankkimista varten tarvitaan varmistettuja dokumentteja. Tämän lisäksi tarvitaan luvan jakelu kaikille niille, joiden pitää varmistua henkilön tai yrityksen luvista. Lupia voidaan poistaa, ja tieto siitä pitää toimittaa edelleen automaattisesti. Viranomaisilla on luvan saaneiden valvontavelvollisuus. Valvontavelvollisuuden täyttäminen edellyttää kaikkien tarvittavien dokumenttien saamista. Valvonta on parhaillaan jatkuvaa, eikä vaadi erillisiä pyyntöjä asiakirjojen saamista varten. Tähän dokumenttialusta tarjoaisi työkalun myös viranomaisille.

3.3 Tuotannon aikana syntyvät dokumentit

Tuotannossa syntyy manuaalisesti ja automaattisesti monenlaisia dokumentteja tuotannon eri vaiheissa (Kuva 1).

Kuva 1. "*Osa kantavan teräsrakenteen toteutukseen liittyvää, perinteistä prosessia ja siihen liittyvää dokumentaatiota"* (Kauppi & Toppila 2018, s. 17).

Kunkin komponentin jäljittämiseen voidaan käyttää materiaalitiedoista, valmistuserästä, komponentin mitoista ja valmistajan tiedoista laskettua tiivistefunktion arvoa, joka voidaan merkitä värillisellä 2D-viivakoodilla komponentin pintaan lasertekniikalla (Karsikas 2013).

Tuotannossa syntyvät hidastumiset, poikkeamat ja häiriöt saattavat aiheuttaa laatupoikkeamia. Tämä kunnossapitotieto on oleellinen osa laatuorientoitunutta tuotantoa (Asp et al. 2018). Dokumenttialustaan tallennettu kunnossapitotieto voidaan synkronoida ajallisesti muun tuotantoprosessitiedon kanssa. Tällä tiedolla voidaan muun muassa ohjata lisätarkastusten kohdistamista tuotteen tiettyihin osiin tai jälkikäteen selvittää lopputuotteessa oleva virhe tai puute. Yhdistetyllä tiedolla voidaan siis parantaa lopputuotteen tai tuotantoprosessin laatua.

IoT eli Internet of Things lisää radikaalisti itsenäisten keskenään kommunikoivien tietokoneiden määrää tietoverkossa ja varsinkin keskenään viestivien tietokoneiden määrää (Huovinen 2017). Tämä viestitys voidaan tallentaa sellaisenaan vaikka pätkittyinä paloina dokumenttialustaan tarvittavaa myöhäisempää käyttöä varten.

Olemassa olevat yrityksen tietovarastot muun muassa varastoinnista, kuljetuksista, tilauksista, toimituksista voivat syöttää suoraan tietoa dokumenttialustaan. Kaikkien näiden tarkoituksena on lisätä tietoa, mitä milläkin ajanhetkellä on yrityksessä tapahtunut. Dokumenttialusta huolehtii kyseisen tiedon jakelusta sovittujen periaatteiden mukaisesti automaattisesti asiakkaille, viranomaisille ja muille sidosryhmille, jotka tätä tietoa tarvitsevat.

3.4 Käytön aikaiset dokumentit

Teräsrakenteiden elinkaaren oletetaan olevan pitkä. Mitä pitempi aika on kyseessä, sitä todennäköisempää on rakenteelle tehtävät muutokset. Itsestään selvää on, että muutokset pitää dokumentoida. Dokumenttialusta voi tallentaa nämä tiedot, mutta lisäksi se automaattisesti välittää tiedot tehdyistä muutoksista valmistajalle, koska valmistaja on tehnyt muutoksen kohteena olevan tuotteen valmistusdokumentit. Riippuen siitä, mihin kaikkiin dokumentteihin muutokset kohdistuvat, tieto muutoksista kulkeutuu komponentin suunnittelijalle ja alihankkijoille automaattisesti. Tämän

tehtävän suorittamista varten dokumenttialustan tarvitsee minimissään tietää tehtyä muutosta kuvaava dokumentti ja sitä koskevan komponentin tai komponenttien viivakoodi.

4 DOKUMENTIT DIGITAALISENA TIETONA

Dokumentit voidaan tallentaa monella eri formaatilla. Koska tiedostomuotoja on monenlaisia, loppukäyttäjällä ei usein ole tietoa itse tiedoston sisäisestä rakenteesta tai koodaustavasta. Suunniteltava dokumenttialusta käsittelee kuitenkin vain yhdenlaisia tiedostoja eli binääritiedostoja.

Yleensä erotellaan erikseen niin sanotut tekstitiedostot muista binääritiedostoista. Tekstitiedostot eivät ole yksikäsitteisiä, vaan niille pitää pystyä sopimaan merkistö; sovitaan, mikä tietyn mittainen bittijono vastaa käyttäjän näkemää merkkiä. Useita merkistöjä on standardisoitu, ja joillakin isoilla ohjelmistotoimittajilla on omiaankin. Bittijonon pituus voi olla muun muassa 7, 8, 16, 21 tai 32 bittiä. Tilan säästämiseksi ja osittain yhteensopivuuden varmistamiseksi bittijonot voidaan koodata myös vaihtelevasti 8 - 32 bitin mittaisiksi. Tehokkain tilan säästö syntyy, jos suurin osa merkeistä voidaan koodata 8 bitillä, jolloin syntyy myös yhteensopivuus vanhempien 8 bitin järjestelmien kanssa. Bittijonon pituuden kasvaessa suuremmaksi kuin 8 bittiä on huomioitava myös tietokoneen sisäinen tavujärjestys, jota voidaan muuttaa joskus uudelleenkäynnistyksen yhteydessä. Yleisimmät tavujärjestykset ovat "little endian" ja "big endian". (Patterson & Hennessy 2005, s. 56; Hinnant 2016) Joskus käytetään myös "middle-endian"-tavujärjestystä (Wolter 2005). Jos tavujärjestystä ei tiedetä tai huomioida oikein, on tuloksena sekavaa tekstiä. Mikäli näitä merkkeihin liittyviä asioita ei huomioida heterogeenisissä ja hajautetuissa sovellusympäristöissä, ei ole mitään mahdollisuutta saada merkistöltään poikkeavaa omaa virallista nimeäänkään oikein esitettyä esimerkiksi kansallisissa väestörekisterin tietojärjestelmissä (Väestörekisterikeskus 2018).

Tämän lyhyen taustoituksen tarkoituksena ei ole käydä läpi erilaisia tapoja esittää tavallinen teksti binäärimuodossa tietokoneelle, vaan selventää sitä, miksi dokumentti-alusta käsittelee vain yhdenlaisia tiedostoja eli binääritiedostoja. Tämän takia dokumenttialustaan voidaan tallentaa mikä tahansa tiedosto sellaisenaan, riippumatta valitusta merkistöstä, koodaustavasta tai tavujärjestyksestä ja saada se takaisin täsmälleen alkuperäisenä. Sisäisesti dokumenttialusta käyttää Unicoden versiota 11.0.0 (Unicode 2018). Koodaukseen käytetään tapaus- ja ympäristökohtaisesti UTF-8, UTF-16BE, UTF-16LE, UTF-32LE tai UTF-32BE:tä. Oletettavaa on, että uusi Unicoden

versio tulee vuoden 2019 aikana, mikäli aikaisempien versioiden antama vihje julkaisu-aikavälistä (v.7 - 2014, v.8 - 2015, v.9 - 2016, v.10 - 2017, v.11 - 2018) pitää paikkaansa (The Unicode Standard 2018, s. 924).

Muita binääritiedostoja on käytössä paljon useampia kuin niin sanottuja tekstitiedostoja. Joissakin tapauksissa käyttäjän kannalta sama tieto saattaa olla useammassa erilaisessa binäärimuodossa. Esimerkkinä on PDF-tiedosto, joka sisältää kuvan sivusta ja tunnistettavan tekstin samasta alkuperäisestä paperitulosteesta. Kuva muodostuu digitoitaessa paperitulosteita tietokoneelle. Ensin siirretään pelkkä kuva digitaaliseen muotoon ja sen jälkeen OCR-muunnoksella yritetään vaihtelevin menestyksin muuttaa kuvana olevat kirjaimet tekstimerkeiksi. Tavallisesti tiedostoon jätetään alkuperäinen kuva tulostamista varten ja tulkitut tekstit hakuja ja tulkintaa varten. (Baker & Maidukov 2015)

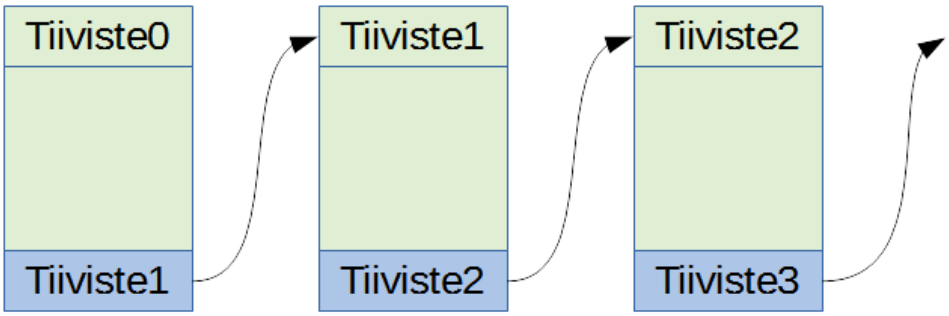
Varsinaisen tiedoston lisäksi dokumenttialustassa tallennetaan metatietona käyttäjä, alkuperäisen tiedoston aikamerkit (luonti-, muokkaus- ja viimeinen katseluajankohta), tiedoston nimi, tiedostoliite, tiedostoliitteeseen liitetty suoritustiedosto eli se ohjelma, jolla kyseinen tiedosto oletusarvoisesti aukaistaan. Samalla laitetaan talteen tiedoston sisällä olevat tunnistetiedot (Salminen 2005).

Dokumenttien säilytysajan olisi hyvä olla pisimmillään koko teräsrakenteen käyttöikä, eli esimerkiksi 100 vuotta, joskus jopa pitempi. Esimerkiksi Tukholman Riddarholmenin kirkontornin valurautainen huippu on vuodelta 1846 (Södervall 2014. s. 13). Mikäli sitä joudutaan korjaamaan, rakentamisaikaiset dokumentit olisivat erinomainen väline nopeuttaa työtä; ei olisi tarpeen tehdä muun muassa materiaalianalyysejä. Digitaalisten dokumenttien säilymiseen kohdistuvat luonnollisesti samat uhkat kuin ylipäänsä koko digitaaliseen infrastruktuuriin. Myös tiedon tallentamiseen, hyväksyttävään tiedon salaustasoon ja käyttöön liittyvien laitteiden kehitys edellyttää tallennetun digitaalisen tiedon jatkuvaa aktiivista muokkaamista ja uudelleentallentamista.

5 LOHKOKETJUTEKNOLOGIA

5.1 Lohkoketjun periaatteet

Yksinkertaisimmillaan lohkoketju on lohko tietoa, josta lasketaan tiivistefunktiolla eli hajautusalgoritmilla luku, joka lisätään seuraavaan lohkoon ja sitä kautta se vaikuttaa seuraavan lohkon tiivistefunktiolla laskettavan lukuun. Näin muodostuu aukoton ketju lohkoja (Kuva 1). (Chandrayan 2018)



Kuva 1. Lohkoketjun periaateperiaate.

Tässä työssä lohkoketjuteknologialla ymmärretään kuitenkin myös useita lohkoketjuihin läheisesti liittyviä teknologioita, eikä pelkästään edellä mainittua lohkoketjun perusperiaatetta. Näitä ovat esimerkiksi epäsymmetrinen salausteknologia, hajautetun verkon palveluorientoinut verkkoliiketoiminta, käytönvalvontateknologiat ja tietoturva. (Lahti 2016; Salonen et al. 2018, passim) Yhteistä näiden teknologioiden soveltamiselle on luotetun, välittävän ja varmistavan kolmannen osapuolen puuttuminen.

Eräänlaista lohkoketjuteknologiaa käyttävässä kirjanpidossa kirjanpitolaista seuraa vaatimus, että tosite, siis tietolohko, sisältää yksilöivän tiedon ja päivämäärän. Tällä yksilöivällä tiedolla tosite voidaan löytää muiden tositteiden joukosta ja tositteet voidaan järjestää haluttuun järjestykseen. (Kirjapitolautakunta 2011, s. 12) Varsinaisessa lohkoketjuteknologiassa tiivistefunktiolla laskettava tiiviste ei ole sataprosenttisesti yksilöivä, vaan kahdelle tietolohkolle voi tulla sama tiiviste, vaikka tietolohkot olisivat sisällöltään täysin erilaiset. Tätä tilannetta kutsutaan törmäykseksi

tai konfliktiksi. Toisaalta mikäli tietolohkot olisivat sisällöltään identtiset, niin silloin myös tiivistefunktiolla laskettava tiiviste olisi täysin sama. (Karvi 2012, s. 29)

Lohkoketjuja käytettäessä olisi mahdollista estää tositteiden kirjaamisjärjestyksen muuttaminen jälkikäteen. Idea on vanha, koska normaalissa kirjanpidossakin tositteiden peräkkäinen numerointi hankaloittaa numeroinnin jälkikäteistä muuttamista (Kauppakamari 2011). Tämä pätee mihin tahansa muuhunkin provenienssiperiaatetta eli dokumenttien alkuperäistä järjestystä noudattavaan dokumenttien arkistointitapaan (Koski 2015, s. 2) tai esimerkiksi diaarinumerointiin. Tosin tositteiden yksilöinti ei aina tapahdu peräkkäisillä numeroilla (Lehtomäki 2016).

Tositteiden peräkkäinen numerojärjestys on analogia lohkoketjujen yhdelle ominaisuudelle, kuten sekin, että kirjanpitotosite voi käsittää joko yhden tapahtuman tai useita tapahtumia (koontitosite). Uuden tositteen tai koontitositteen liittäminen olemassa olevien tositteiden väliin aiheuttaa järjestyksessä seuraavien tositteiden numeroinnin uudelleenlaskennan. Tositteiden numeroinnin laskenta on yksinkertainen ja helppo suorittaa: kasvatetaan vain jäljellä olevien tositteiden numeroa yhdellä.

Lohkoketjussa ei lasketa numerointia kasvattamalla numerointia yhdellä, vaan laskemalla tiivistefunktio tositteen tai dokumentin sisällöstä yhdistettynä edellisen tositteen tai dokumentin numeroon (Karvi 2012, s. 29). Näin kytketään peräkkäiset tositteet tai dokumentit toisiinsa, kuitenkin vain yhteen suuntaan eli eteenpäin.

Tiivistefunktio on helppo laskea tietokoneella, ja funktion tuloksena on tietyn mittainen bittijono eli iso kokonaislukunumero (Comer 2009. s. 515). Bittijonon pituus riippuu käytettävästä tiivistefunktiosta (Stallings 2006, s. 353). Tiivistefunktiolla on tarkoitus laskea tiedon "sormenjälki", joka voi olla tiedosto, viesti tai mikä tahansa data-kokonaisuus (Stallings 2006, s. 335). Tiivistettävän datan määrälle ei ole käytännössä rajoituksia, mutta hyvälle tiivistefunktion sormenjäljelle on useita vaatimuksia: syntynyt sormenjälki on vakiokokoinen, sormenjäljen laskenta on nopea ja helppo suorittaa sekä ohjelmallisesti että laitteistolla, jos alkuperäinen data tunnetaan. Sormenjäljestä ei voi helposti laskea tai saada selville alkuperäistä dataa. Näin on käytännössä miltei mahdotonta löytää toista dataa, joka tuottaisi saman tiivisteen. Samoin on melko mahdotonta löytää kaksi erilaista lähtödataa, jotka tuottavat saman tiivisteen (Stallings

2006, s. 335). Teoreettisesti tiivistefunktiota ei ole mahdollista tehdä puhtaasti satunnaista lukua tuottavaksi, koska lähtötieto ei ole satunnaista (Knuth 2000, s. 515).

Lohkoketju voi helposti haarautua, eli yhdestä ketjusta tuleekin kaksi tai useampi ketju, koska lohkoketju on kytketty vain eteenpäin. Toisaalta mikään ei estä, etteikö lohkoketjuja voisi olla monta voimassa yhtä aikaa. Mikäli halutaan lohkoketjuja olevan vain yksi yhteinen tiettyä tarkoitusta varten, täytyy erikseen jollakin tavalla varmistaa, että niitä todella on vain yksi. Varmistaminen voidaan tehdä hajauttamalla laskentaa keskenään kilpailevien tahojen kesken (Proof of Work) (Gupta 2017, s. 17). Tässä tavassa tahot voivat olla myös lohkoketjun ulkoisia tahoja, jopa kenen tahansa internetin käyttäjän selain. Toinen tapa varmistaa on valita sisäisesti lohkoketjun omistajien joukosta sellainen taho tai sellaiset tahot varmistamaan, jotka menettäisivät eniten lohkoketjun hajoamisesta (Proof of Stake). Varmistamista ei saada koskaan täysin varmaksi (Schumann 2018). Enemmistö päättää, eli vähintään yli puolet osallistuneista tahoista riittää päättämään, mikä yksi haara lohkoketjuun jää lopulta voimaan (Schumann 2018).

5.2 Varmentamismenetelmät

5.2.1 Tiedon salaaminen

Lohkoketjun lohkossa tallennettu tieto voi olla salattuna symmetrisellä tai epäsymmetrisellä avaimella. Tiivistefunktion tulosta voidaan tällöin käyttää ainoastaan tarkistamaan, ettei lohkoon tallennettua salattua tietoa ole mitenkään muutettu jälkikäteen. Selväkielisestä lohkosta voidaan laskea oma tiivistefunktion tulos, joka puolestaan varmistaa salaamattoman tiedon sisällön. Symmetrinen salaus tarkoittaa, että on vain yksi avain, jolla lohko voidaan salata ja purkaa. Tämä salaus on käyttökelpoinen, jos vain yksi taho tietää riittävän pitkän ja satunnaisen avaimen. Taho on ennalta määritelty ryhmä. (Kaarnalehto 2011, s. 16)

Epäsymmetrisestä salauksesta esimerkkinä on PKI (Public Key Infrastructure). Julkisen avaimen perusrakenne on peräisin X.509-standardista, joka määrittelee julkisen avaimen sertifikaattitiedoston muodon (Rouse 2014). Julkisen avaimen salaus perustuu kahden avaimen olemassaoloon. Toinen avain on salainen avain, jonka vain yksi salatun viestityksen osapuoli tietää. Toinen avain on julkinen avain, jonka kaikki osapuolet

saavat tietää. Selväkielinen teksti salataan toisella avaimella ja salattu teksti saadaan selväkieliseksi jäljelle jääneellä avaimella. (Rouse 2014)

Tätä epäsymmetrisyyden ominaisuutta voidaan käyttää usealla tavalla. Mikäli alkuperäinen teksti salataan julkisella avaimella, niin ainoastaan salaisen avaimen haltia saa tekstin auki.

Kuvitteelliset henkilöt Alice ja Bob lähettävät toisilleen salattuja viestejä seuraavasti:

1. Alice salaa siirrettävän tekstin omalla salaisella avaimellaan ja Bobin julkisella avaimella. Järjestyksellä, kumpi salaus tehdään ensin, ei ole väliä.
2. Bob purkaa saamansa viestin omalla salaisella avaimellaan ja Alicen julkisella avaimella.

Tällöin kukaan muu kuin Alice ei ole voinut viestiä salata, eikä kukaan muu kuin Bob pysty sitä purkamaan. (Rouse 2014)

Hajautetun rakenteensa vuoksi dokumenttialusta käyttää käyttäjiensä luottamusverkostoa (Ryabitsev 2014) toteamaan ja varmistamaan kunkin käyttäjän tai organisaation identiteetin. Kukin käyttäjä tai organisaatio saa paljastaa tai olla paljastamatta julkisen avaimensa yhteydessä haluamansa yksilöivät tiedot. Oleellista on vain, että julkisen avaimen vahvistavat tahot tietävät, kenellä on julkisen avaimen salaisen avainparin hallinta. Freeware-ominaisuutensa takia dokumenttialusta on käyttäjilleen maksuton, jolloin myös tämä identiteetin vahvistaminen on maksuton. Hajautettu luottamusverkon käyttö tekee järjestelmästä tietoturvallisemman; heterogeenisyyden takia keskitetyt tietoturvahyökkäykset ovat mahdottomia. Dokumenttialusta voi myös käyttää PKI:n varmennehierarkian mukaisesti toimitettuja sertifikaatteja, mutta ainoastaan epäsymmetristen avainparien osalta. Tällöin kaikki sertifikaatin varmennehierarkian tiedot jäävät hyödyntämättä tietoturvasyistä (Kirk 2018).

Dokumenttialusta pystyy arkkitehtuurinsa takia käyttämään FIPS PUB 140 -dokumentin (FIPS 2002) mukaisia erillisiä salausmoduuleita, jotka mahdollistavat salaisen salausavaimen tallentamisen tietoturvallisesti. Samalla näiden moduuleiden sisältämät erikois-

piirit nopeuttavat tarvittavaa laskentaa. Dokumenttialustan täytyy vain päästä salaamaan tietoa näiden erillisten moduuleiden sisältämien salaisten avaimien avulla.

Salausavaimet eivät sisällä tietoa siitä, kenen ne ovat, jolloin käyttäjä pystyy pysymään täysin pseudoanonyyminä. Mikäli käyttäjä vaihtaa avaimia riittävän usein eikä käytä avaimia mihinkään muuhun, on mahdollista saavuttaa todellinen anonymiteetti. Dokumenttialustan suunnittelukriteerinä on pyrkimys täydelliseen anonymiteettiin, koska silloin käyttäjä voi itse päättää, ketkä saavat hänet tunnistaa.

Salaus perustuu matemaattisen salausyhtälön laskennan vaikeuteen. Laskennan vaikeus on riippuvainen käytettävissä olevista algoritmeista ja tietokonekapasiteetista, joihin kvanttitietokoneet tulevat tuomaan uusia mahdollisuuksia. Tämän takia salauksessa ei voida sitoutua vain yhteen menetelmään, vaan salausmenetelmät ovat jatkuvan päivittämisen kohteena - myös dokumenttialustassa. Koska tieto on dokumenttialustassa koko ajan aktiivisena tietokoneiden käsittelyssä, salausmenetelmän vaihtaminen voidaan toteuttaa automaattisesti. Homomorfinen salaus (FHE) mahdollistaa salatun tiedon laskentakäytön ilman salauksen purkua (Halevi 2018). Kaikki tieto ei ole salattu kuitenkaan samalla tavalla, vaan osa, varsinkin käsittelyä varten tarvittava metatieto-osuus on tallennettu eri avaimilla.

5.2.2 Tiedonsiirron varmentaminen

Lohkoketjut voivat olla täysin yksityisiä yhden tahon ylläpitämiä, mutta tällöin lohkoketjujen suurin hyöty jää saavuttamatta. Koska jos on vain yksi taho, joka pitää kaikkia lohkoketjun osia hallussaan, lohkoketjun keskeltä voidaan muuttaa mikä tahansa lohko ja sitten laskea uudelleen tiivistefunktioiden arvot ja saada näin jälleen "puhdas" lohkoketju.

Jos lohkoketjut ovat kokonaan tai osittain julkisia, tätä uudelleenlaskentaa ei voida suorittaa ilman, että se havaitaan. Tämän takia lohkoketjujen osia täytyy pystyä siirtämään internetissä eri osapuolien välillä. Uudelleenlaskennan vaikeuttamiseksi lohkoja siirretään myös tuntemattomille tahoille, eli lohkon lisääjä ei saa tietää, mihin kaikkialle hänen tallentamansa lohkoketju tallennetaan. Tämä edellyttää tiedonsiirron varmentamista dokumenttialustassa.

Niin kauan kuin tietoa on tallennettu ja siirretty, on ollut ongelmia suojata informaatiota ulkopuolisilta (Bartee 1991, s. 269). Internetissä tiedonsiirron varmentamisella oletetaan yleensä tarkoitettavan ainoastaan tiedonsiirron aikaista salausta. Tässä työssä tiedonsiirron varmentaminen määritellään laajemmin ja tarkoittamaan järjestelmän A ylläpitämän salatun tiedon kopioimista järjestelmän B ylläpitämäksi salatuksi tiedoksi. Tietokoneessa paikallisesti tallennettu ja siirrettävä tieto on koko ajan salattu epäsymmetrisillä avaimilla.

HTTPS:n tiedonsiirron aikana itse tieto on salattu symmetrisellä avaimella, jonka satunnaisuus ja pituus vaikuttaa oleellisesti siirrettävän tiedon salauksen paremmuuteen (Held 1993, s. 6). Tämä salattu tieto sisältää myös salaamatonta ylläpitotietoa, joka minimissään on salaamattoman ja salatun tiedon tiivistefunktioiden arvot itse tiedon tunnistamista varten. Periaatteessa yksi tiivistefunktion arvo riittäisi tiedon tunnistamiseen, mutta se ei riitä salauksessa ja sen purkamisessa tapahtuvien virheiden havaitsemiseen. Se myös estäisi eri salausmenetelmien käyttämisen rinnakkain, koska eri salausmenetelmillä tehdyillä saman tiedon salauksilla on eri tiivistefunktioiden arvot.

5.2.3 Tiedon tallennuksen varmistaminen

Lohkoketjun perusideana on ketjun tallentaminen useita kertoja ja usealle eri taholle verkossa. Tällöin yksittäinen taho ei pysty jälkikäteen uudelleenlaskemaan ketjun koossa pitäviä tiivisteitä. Samalla salattu tieto tulee tallennettua useaan kertaan. Yhden tai useamman tallennuslaitteen vikaantuminen, hukkaaminen tai varastaminen ei aiheuta tietojen katoamista.

Laitekustannukset kasvavat eksponentiaalisen rajusti, jos halutaan keskitetyn järjestelmän tietojen tallentamista parantaa 99, 99,9 tai 99,99 prosenttiin. Myös erilaiset varmuustallennusjärjestelmät lisäävät kustannuksia ja tuovat uusia tietoturvaongelmia kokonaisjärjestelmään. Hajautetussa järjestelmässä ei varsinaisia varmuustallennuksia edes tarvita. Edellä mainittua moneen kertaan tallentamista ei käsitetä varmuuskopioinniksi, koska kukin kopio on aktiivinen osa järjestelmää. Lohkoketju pitää tallennetun tiedon muuttumattomana. Siihen voidaan lisätä uusia lohkoja, jotka tallentuvat riittävän monille eri tahoille. Tietoa voidaan siis tavallaan päivittää, ei muuttamalla vanhaa tietoa, vaan lisäämällä uutta. Samalla estetään lukkotilanteet

päivityksessä: kaksi tai useampi taho voi päivittää tietoa yhtä aikaa ja toisistaan riippumatta.

Dokumenttialusta ei ota kantaa siihen, mikä päivitys on sisällöltään oikein. Esimerkiksi jos yksi päivitykseen oikeutettu taho haluaa muuttaa tallennetun dokumentin sisältämää päivämäärää viikolla eteenpäin ja toinen samoilla oikeuksilla oleva taho puolestaan viikolla aikaisemmaksi, niin ne voivat sen tehdä. Jos päivämäärä tulee myöhemmin ongelmaksi, dokumenttialustassa on tieto alkuperäisestä päivämäärästä ja tehdyistä päivityksistä aikaleimoineen ja tekijöiden tunnistetietoineen. Näiden tietojen perusteella ristiriita voidaan selvittää.

Moneen kertaan tallentaminen on järkevää myös tiedon saannin nopeuttamisen ja kuormituksen skaalautumisen takia. Verkossa toimivat dokumenttialustat ovat keskenään täysin riippumattomia toistensa toiminnasta ja niin myös toimimattomuudesta.

5.3 Funktionaalinen ohjelmointi

Lohkoketjut eivät toimiakseen vaadi funktionaalista ohjelmointia. Funktionaalinen ohjelmointitapa kuitenkin soveltuu hyvin sellaiseen ympäristöön, missä lohkoketjuja on tarkoitus käyttää.

Funktionaalista ohjelmointia (Hughes 1990, passim; Michaelson 2011, passim) käytetään dokumenttialustan ohjelmoinnissa ylimmällä tasolla, koska se suoraan tarjoaa ajatusmallit ja työkalut käsitellä oikein ja helposti rinnakkaisuutta, geneerisyyttä ja ohjelman oikeaksi todistamista. Ohjelmien rinnakkaisuus on puhtaassa funktionaalisessa ohjelmoinnissa sisään rakennettuna ominaisuutena. Koska suoritettavien funktioiden laskentajärjestyksellä ei ole väliä, ne voidaan suorittaa toisistaan riippumatta missä järjestyksessä tahansa. Geneerisyys on helppo toteuttaa, koska funktion antama arvo on riippuvainen vain lähtöarvosta, eikä funktiolla ole sivuvaikutuksia (Labrosse 2002b, s. 68).

Funktionaalisessa ohjelmoinnissa funktion sisäinen rakenne on kokonaisuuden kannalta sivuseikka ja voi olla toteutettu vaikka deklaratiivisesti, proseduraalisesti, logiikka-pohjaisesti tai olio-ohjelmointia soveltaen. Ohjelman oikeaksi todistaminen

funktionaalisessa ohjelmoinnissa tapahtuu matemaattisen induktion avulla, joka perustuu vaiheittaisen rekursion hyväksikäyttöön. Tämän takia funktionaalisessa ohjelmoinnissa ei käytetä toistolauseita vaan rekursiota, eikä käytetä sijoituslauseita, vaan muuttuja säilyttää saman arvon koko olemassaolonsa ajan. Molemmat mahdollistavat ja helpottavat matemaattisten todistusten käyttöä (Pierce 1991, 53 - 57; Pierce 2002, s. 51). Tarkat määritykset ovat ehdoton edellytys koodin uudelleenkäyttöä (code reuse) ajatellen (Prowell et al.1999, s. 16).

Funktionaalisessa ohjelmoinnissa on analogiapiirteitä lohkoketjuihin nähden: kerran tehtyä muuttujaa ei saa eikä voida muuttaa jälkikäteen (immutable object). Lisäksi kummallakaan ei ole sivuvaikutuksia (side effects), vaan molemmat antavat saman tuloksen joka kerta.

Funktionaalista ohjelmointia on pidetty vaikeana suorittaa sen matemaattisuuden ja vaaditun osaamistason takia, mutta vähitellen on alkanut muutos sen käyttöfrekvenssissä (Korkishko 2018). Funktionaalinen ohjelmointi onkin työläs ja hankalaksi koettu menetelmä, jos sillä on tarkoitus tehdä kaikki sovelluksen vaatimat ohjelman osat, esimerkiksi rajapinnat. Dokumenttialustan toteutuksessa funktionaalista ohjelmointia käytetään vain siellä, missä se on soveltuvimmillaan, eli tasolla, missä sovellus, esimerkiksi dokumenttialustaohjelma, määritellään. Yksinkertaisimmillaan se on pelkkä pääohjelman kutsu, joka määritellään suoritettavan ohjelman ulkopuolella ja käännöksen jälkeen datana. Data siis määrittelee pääohjelman ja sitä kautta sen, mitä ohjelma tekee. Käännetty ohjelma on yksi ja sama riippumatta siitä, mitä datan määrittelemä sovellus tekee. Toisesta näkökulmasta kyseessä on siis eräänlainen ohjelmasimulaattori, joka pystyy simuloimaan useita eri sovelluksia. Käännettävästä ohjelmasta ei tarvitse tehdä eri versioita, koska käytetään funktioiden monimuotoisuutta eli polymorfismia luokkatasolla (Partanen 2003, s. 12). Toisaalta datana määritelty pääohjelma voi olla dokumenttialustan tulkittavissa ja ajettavissa oleva satojen funktioiden kokoelma.

```
<component>
   <title>Registrator 1.0</title>
   <configuration>
      <program>
         <funcall>
            <name>copy_message_from_agrajag</name>
            <params>
               <param>
                  <transfer>
                     <input>..\AGRAJAG\REGISTRATION</input>
                     <output>REGISTRATION</output>
                     <header>
                        <message>
                           <sender>DRUPAL</sender>
                           <receiver>REGISTRATION</receiver>
                           <type>message</type>
                           <msg>
                              <funcall>
                                 <name>source_message</name>
                              </funcall>
                           </msg>
                        </message>
                     </header>
                  </transfer>
               </param>
               <param>
                  <transfer>
                     <input>..\AGRAJAG\WEBFORM</input>
                     <output>WEBFORM</output>
                     <header>
                        <message>
                           <sender>DRUPAL</sender>
                           <receiver>WEBFORM</receiver>
                           <type>message</type>
                           <msg>
                              <funcall>
                                 <name>source_message</name>
                              </funcall>
                           </msg>
                        </message>
                     </header>
                  </transfer>
               </param>
            </params>
         </funcall>
      </program>
   </configuration>
</component>
```

Kuva 2. Esimerkki yksinkertaisesta pääohjelman kutsusta.

Vaikka XML:ää ei ole luokiteltu ohjelmointikieleksi vaan pelkäksi tekstidataksi, mikään ei estä sitä käytettävän kuvaamaan mitä tahansa tekstiä, kuten esimerkiksi ohjelman lähdekoodia (Kuva 2).

Dokumenttialustan sisäinen esitysmuoto on muunnettavissa useampaan ulkoiseen esitysmuotoon. Samoin dokumenttialusta kykenee tulkitsemaan useampaa erilaista ulkoista esitysmuotoa. Jos esimerkiksi tehdään muutamia mekaanisia tekstimuunnoksia pääohjelman kutsulle (Kuva 2) sisältöön puuttumatta, saadaan helposti Lisp-ohjelmointikieltä muistuttava alla oleva listaus.

```
(component
  (title "Registrator 1.0")
  (configuration (program
    (funcall (name copy_message_from_agrajag)
      (params
        (param (transfer
          (input "..\AGRAJAG\REGISTRATION")
          (output REGISTRATION)
          (header (message
            (sender DRUPAL)
            (receiver REGISTRATION)
            (type message)
            (msg (funcall (name source_message)))
        ))))
        (param (transfer
          (input "..\AGRAJAG\WEBFORM")
          (output WEBFORM)
          (header (message
            (sender DRUPAL)
            (receiver WEBFORM)
            (type message)
            (msg (funcall (name source_message)))
        ))))
      )
    )
  ))
)
```

6 KÄYTÄNNÖN SOVELLUS - DOKUMENTTIALUSTA

6.1 Suunnitteluehdot

Dokumenttialustalta edellytetään, ettei mikään yksittäinen vika tai ongelma estä järjestelmän normaalia käyttöä. Toisaalta minkään yksittäisen laitteen ei tarvitse olla koko ajan päällä tai käyttäjän kytkeytyneenä dokumenttialustaan.

Dokumenttialustan täytyy ensimmäisen kerran käynnistyessään virtuaalisessa tai fyysisessä tietokoneessa luoda itselleen automaattisesti yksilöllisen virtuaali-identiteetin, jolla ohjelma kytkeytyy dokumenttialustajärjestelmäverkkoon.

Mikäli tietokoneella on nimetty käyttäjä, hänen täytyy pystyä luomaan itselleen salasanan tai salalauseen, jolla pääsee käyttämään dokumenttialustaohjelmaa vuorovaikutteisesti. Käyttäjän antamaa salasanaa tai salalausetta ei saa tallentaa selväkielisenä, salattuna eikä tiivistettynä tai muutoinkaan dokumenttialustaan tai itse tietokoneelle. Tämän ehdon toteuttamiseen tarvitaan nollatiedon salasanoja (ZKP) (Geraud 2017). Dokumenttialusta saa siis ainoastaan hyvin todennäköisen tiedon siitä, että käyttäjä tietää itse oman salasanansa tai salalauseensa. Tämän menetelmän ansiosta käyttäjä tarvitsisi vain yhden salasanan tai salalauseen kaikkiin käyttämiinsä tietokonejärjestelmiin. Dokumenttialustassa käyttäjälle voidaan määritellä tarvittaessa symmetrinen avain tiedon henkilökohtaista salausta varten. Salasanan tai salalauseen lisäksi käyttäjän täytyy pystyä määrittelemään itselleen käyttöön kaksivaiheinen tunnistautuminen. Käyttäjä voi myös määritellä itselleen järjestelmän tappokytkimenä toimivan salasanan tai salalauseen tai jonkin kaksivaiheinen tunnistautumisen menetelmän. Tappokytkimen tulee "poistaa" tiedostot käytöstä paikallisessa dokumenttialustassa. Näin käyttäjä voi olla varma, että liikesalaisuudet eivät vuoda, vaikka joutuisikin aukaisemaan tietokoneensa tai ohjelmansa ulkopuolista tarkastusta varten, esimerkiksi rajamuodollisuuksissa. Käyttäjän pitää pystyä valitsemaan käyttöönsä julkisia avainpareja ja luomaan julkisen salauksen avainpareja tarpeen mukaan.

Dokumenttialustan on käytettävä ensisijaisesti asynkronista tiedonsiirtoa. Mikäli vain synkroninen on mahdollista, sitä pitää käyttää asynkronisesti. Dokumenttialustan täytyy

toimia ilman jatkuvaa ihmisen suorittamaa ylläpitoa. Dokumenttialustan on käytettävä vain tunnettuja ja hyviksi todettuja salausmenetelmiä. Salausmenetelmää pitää pystyä muuttamaan tuotantoympäristössä ilman uutta dokumenttialustan ohjelmaversiota.

Käyttäjän täytyy pystyä tallentamaan tiedosto ja sitä määrittävää metatietoa avain-data -pareina dokumenttialustaan ja suojaamaan tiedot valitsemillaan julkisen salakirjoituksen avainpareilla. Käyttäjän pitää pystyä hakemaan dokumentteja dokumenttialustan sisältämän metatiedon avain-data -pareilla.

Dokumenttialustan täytyy toimia yleisimmissä käytössä olevissa käyttöjärjestelmissä pois lukien ehkä vain MINIX 3 (Tanenbaum 2017).

6.2 Suunnitteluperiaatteita

6.2.1 Yleistä

Tietokoneohjelman koodissa on aina yksi nimetty pääohjelma, joka toimii lähtökohtana ja ylimpänä määritelmänä, mitä ohjelmalla halutaan tehdä (Haataja 2000, s. 27). Dokumenttialustassakin koodissa on yksi nimetty pääohjelma, jonka tehtävänä on simuloida mitä tahansa muuta sovellusta. Tätä simulointia varten sen pitää suorittaa sille annettu dataviesti suoritettavana ohjelmakoodina. Tämä ohjelmakoodi määrittelee sovelluksen pääohjelman ja sen, mitä ohjelmalla halutaan tehdä. Näin menetellen voidaan ajaa rajaton määrä eri sovelluksia tai eri versioita samasta ohjelmasta yhdellä ja samalla etukäteen käännetyllä ohjelmakoodilla. Ohjelmakoodin simulointi on yleensä hidasta ja paljon tietokoneresursseja käyttävää. Näin on nytkin, jos simuloitaisiin koko ohjelma. Dokumenttialustan ohjelmassa simuloidaan kuitenkin vain pääohjelmaa ja niitä osia ohjelmasta, jotka ovat vain kyseiselle sovellukselle spesifisiä. Tällöin simuloitu sovellus ei eroa millään tavalla suorituskyvyn tai resurssien käytön suhteen simuloimattomasta sovelluksesta.

Monoliittisessa ohjelmarakenteessa kaikki datalle muokkausta tekevät funktiot ovat yhden pääohjelman alla. Modulaarisessa ohjelmarakenteessa kukin funktio on eriytetty omaksi itsenäiseksi osaksi eli moduuliksi oman pääohjelmansa alaisuuteen (Pressman 2005, s. 267, 847). Nämä moduulit kommunikoivat keskenään lähettämällä viestejä (Hietanen 2001, s. 7).

Dokumenttialustan ohjelma voidaan asettaa simuloimaan sekä monoliittista että modulaarista ohjelmarakennetta. Modulaarisen ohjelmarakenteen simulointi tapahtuu kopioimalla ohjelma n-kertaa ja antamalla kullekin niistä käynnistyksen yhteydessä datassa tieto, mitä funktiota kunkin erillisen ohjelman pitää suorittaa. Monoliittinen ohjelmarakenne eli yhden ainoan suoritettavan ohjelman simulointi tehdään asettamalla sisään tulevaan dataviestiin myös tieto suoritettavasta funktiosta ja ulostulevaan viestiin lisätään tieto seuraavan vaiheen funktiosta. Sen jälkeen dataviesti kokonaisuudessaan annetaan saman ohjelman suoritettavaksi uudestaan. Tätä jatketaan, kunnes ei enää ole seuraavia suoritettavia funktioita. Kokonaisjärjestelmän suorituskykyä voidaan nostaa kopioimalla tätä yhtä suoritettavaa dokumenttialustan ohjelmaa yhdessä koneessa, kunnes kaikki laitteiston tarjoama suorituskyky on käytetty tehokkaasti hyödyksi (multi-core processors, multiprocessing).

Mikään ei estä kopioimasta dokumenttialustan ohjelmaa usealle tietokoneelle, jolloin suorituskyvylle ei ole varsinaista ylärajaa, vaikka tämän tapainen rinnakkainen suoritus on työläämpi tehdä ja vaikeampi testata (Ben-Ari 1982, s. 6). Tällöin yksittäisen verkon, laitteen tai ohjelman vikaantuminen ei estä tai edes häiritse millään tavalla järjestelmän käyttäjälle näkyvää toimintaa, ellei vikaantumiskohteena ole käyttäjän oma ohjelma, kone tai verkkoliityntä. Koska mikään ohjelma, tietokone, sähköverkko, tai tieto-liikenneverkko ei ole 100-prosenttisen varma, (Feldman 1999, s. 46) järjestelmä pitää suunnitella ja rakentaa kestämään tämä epävarmuus, varsinkin, jos vikaantumisesta aiheutuneet jälkikustannukset ovat moninkertaiset verrattuna suunnittelu- ja toteutus-kustannuksiin.

6.2.2 Tekoäly - älykäs automaatio

Tekoälystä käytetään useita nimityksiä, muun muassa: heikko tekoäly, sovellettu tekoäly, vahva tekoäly, yleinen tekoäly, keinoäly, oppiva keinoäly, koneoppiminen, älykäs automaatio ja ohjelmistorobotiikka. Termit sisältävät hieman toisistaan poikkeavia merkityksiäkin.

Työ- ja elinkeinoministeriön asettaman tekoälyajan työtä käsittelevän työryhmän raportissa esitetty funktionaalisuuteen pohjautuva määritelmä tekoälylle on "*järjestelmän kyky toimia tavoitteellisesti ja ympäristöään ennakoiden*". Määritelmä ei ole kattava tekoälysovellusten osalta. Työryhmän puheenjohtajan Osmo Soininvaaran

mukaan tekoälyn ja digitalisaation vaikutuksia ei voi erottaa toisistaan. Yleisesti tekoälyn soveltaminen voidaankin käsittää digitalisaation synonyyminä. (Koski & Husso 2018, s. 10, 54) Toisaalta tekoälyä hyvin kuvaa paradoksaalinen lause: "*Tekoäly on sitä, mitä ei vielä pystytä ohjelmoimaan tietokoneelle*" (Ailisto et al. 2017, s. 2). Teslerin teoreema ilmaisee asian hieman toisin sanoin: "*Älykkyys on mitä tahansa, mitä koneet eivät ole vielä tehneet.*" (Tesler 1970).

Nykyiset tekoälyn saavutukset eivät johdu uusista algoritmeista, vaan pikemminkin käytettävissä olevan datan määrän lisääntymisestä (Ailisto et al. 2017, s. 2). Käytettävissä olevan laskentakapasiteetin ja datan määrän kasvu aiheuttaa mielikuvan älykkäästä eli ihmisen kykyä vastaavasta tai sen ylittävästä toiminnasta. Pelkästään nopeuden takia ihmistä nopeampaa konetta tuskin pidetään älykkäänä. Sama koskee esimerkiksi Wikipediaa, vaikka se sisältää suunnattoman määrän dataa.

Konetekniikassa on pitkään ollut käytössä termi "automaatio". Automaatiolla tarkoitetaan itsetoimivaa laitetta tai järjestelmää. Automaatiota edistää suuresti laskentatehon lisäys ja ympäristöstä saatavan datan määrän kasvaminen. Tämä tosiasiallinen laskentatehon ja datan lisäyksen hyödyntäminen on se, mitä tässä työssä käsitetään tekoälyllä, digitalisaatiolla tai oikeammin älykkäällä automaatiolla.

Älykkääseen automaatioon perustuva dokumenttialusta voidaan käytännössä toteuttaa useilla eri tavoilla. Tässä työssä valittu tapa pohjautuu mahdollisimman pitkälle lambda-kalkyylin käyttöön. Jo 1950-luvulla "tekoäly"-termiä käyttänyt John McCarthy (Myers 2011) kehitti listojen käsittelyyn perustuvan Lisp-ohjelmointikielen, joka mahdollisti koodin käsittelyn datana ja datan käsittelyn koodina. Lispin perusidea on lambdakalkyyli, jota on nykyään jo monissa muissakin ohjelmointikielissä. Puhtaan lambdakalkyylin, joka sisältää itse asiassa vain funktioiden määritelmiä, syntaksi määritellään seuraavasti:

e::= x	*muuttuja*
| λx.e	*abstraktio*
| e1 e2	*sovellus*

missä abstraktio λx.e on funktion määritelmä ja lauseke e on funktion runko ja muuttuja x on argumentti. (Chong 2018, s. 1) Lambdakalkyyli antaa vahvan matemaattisen ja

teoreettisen pohjan, jossa teorioiden oikeaksi todistaminen on mahdollista (Chong 2018, s. 4). Ohjelmoinnissa käytettyjä algoritmejakin voidaan parhaassa tapauksessa pitää todistettujen lauseiden joukkona eli matemaattisina teorioina.

Dokumenttialustassa tietoa käsitellään ja esitetään nimenomaan abstrakteina listoina (Stavely 1999, s.149). Lisäksi Lisp-ohjelmointikielen tapaan ei tehdä mitään eroa suoritettavan koodin ja sisään luettavan ja uloskirjoitettavan datan välillä. Tavalla ei ole tarkoitus helpottaa ohjelmoijan koodin suunnittelu- ja kirjoitustyötä, vaan valmiin järjestelmän testaajan työtä, ja lopulta poistamaan ideaalitapauksessa erillinen valmiin järjestelmän testaustyö kokonaan. On lisäksi perusteltua kysyä, onko tällaisten hajautettujen ja funktionaalisten kokonaisjärjestelmien testaustekninen suoritus erityisesti ja nimenomaan ohjelmointivirheiden etsimisessä käytännössä mahdollista alalla yleisesti käytetyillä tavoilla ja työkaluilla. Aina ei kuitenkaan voida käsitellä samalla tavalla koodia ja dataa, koska useat ajo- ja kehitysympäristöt vaativat lähtö-kohtaisesti niiden erottelun.

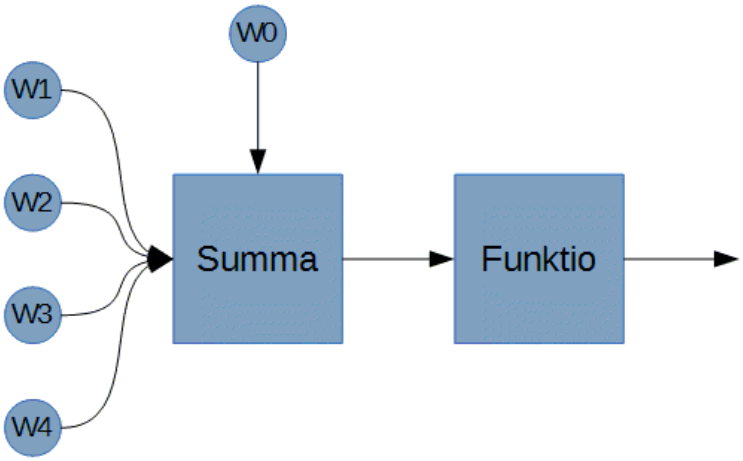
Lambdakalkyylissä funktioiden laskentajärjestys on täysin vapaa, suorituksen osien rinnakkaisuutta voidaan hyödyntää rajattomasti ja itse tietokoneohjelma on rakenteeltaan geneerinen eli yleiskäyttöinen (Michaelson 2011, s. 1 - 14). Rinnakkaisuus tuo mahdollisuuden kasvattaa laskentakapasiteettia hyödyntämällä jo olemassa olevaa laitekapasiteettia. Nämä ovat ominaisuuksia, jotka mahdollistavat omalta osaltaan tekoälysovellusten tekemisen.

Älykäs automaatio esiintyy dokumenttialustassa useilla tavoilla. Alusta ei tarvitse ihmisen ylläpitoa toimiakseen, vaan ylläpitotoiminnot on automatisoitu. Näitä järjestelmän automaattisia ylläpitotehtäviä ovat muun muassa:

- annotointi eli tietoa kuvailevan tiedon tuottaminen eli metatiedon automaattinen generointi,
- dokumenttityyppien tunnistaminen,
- dokumenttien asynkroninen siirto,
- dokumenttien säilyvyyden varmentaminen useaan paikkaan monistamalla,
- dokumenttien salaus,
- hajautetun tietokannan päivittäminen,

- yksittäisen tietokoneen rajallisten resurssien huomioon ottaminen,
- tiedonsiirtokapasiteetin rajoituksien huomioiminen,
- keskusmuistin ja levytilan rajoituksien huomioiminen,
- laitevioista selviytyminen ja siihen liittyvä tilannetietoisuus,
- sekä tahallisten että tahattomien väärinkäytösten vaikutuksen minimointi,
- alustan suorituskyvyn seuranta,
- käyttäjien seuranta ja heidän profiiliensa hallinta,
- ohjelman pysäyttäminen ja uudelleen käynnistäminen hallitusti.

Jo 1980-luvulla Teuvo Kohonen kehitti biologista neuronia vastaavan toimintamallin tietokoneille (Kohonen 1987). Malli ei ole muuttunut, vaikka nykyisin puhutaan mieluummin tekoälyn neuroverkoista ja kerroksellisesta syväoppimisesta (Ojansuu 2018, s. 33 - 35). Mallina tekoneuroni sisältää useita sisääntuloja, joista kullakin on oma painotuskertoimensa (wn). Painotuskerroin voi olla positiivinen tai negatiivinen. Kaikkien sisääntulojen arvot lasketaan yhteen. Mikäli summa ylittää asetetun raja-arvon (Summa), niin käynnistetään funktio (Funktio), jonka arvo tulee tekoneuronin ulostuloksi seuraavaa vaihetta varten (Kuva 3) (Kohonen 1987). Dokumenttialustassa tekoneuronimallia käytetään älykkään automatisoinnin toteutuksissa. Sen merkitys on siinä, että ohjelma pystyy tekemään päätöksen painottamalla erilaisia mitattavia tekijöitä.



Kuva 3. Tekoneuroni (mukaillen Kohonen 1987).

Suositussa client-server -arkkitehtuurissa lähdetään oletuksesta, että palvelin on aina client-ohjelmien käytössä päällä eikä pysähdyksiä sallita (Rasheed 2018). Todellisuudessa katkoksia, joko suunniteltuja tai suunnittelemattomia, tulee aina ja ne vaativat ylläpitohenkilöstön erityistoimenpiteitä.

Jotta dokumenttialustassa ohjelma voitaisiin pysäyttää kesken kaiken turvallisesti ja sitten käynnistää automaattisesti uudestaan, tarvitaan tarkka ohjelman tilannetieto ajalta juuri ennen pysäytyshetkeä. Kutsua edeltänyt tilanne pitää pystyä palauttamaan täydellisesti. Funktiolla ei siis saa olla mitään sivuvaikutuksia, joita on hankala palauttaa. Puhdas funktionaalinen ohjelmointi takaa, ettei sivuvaikutuksia ole. Oletuksena on, että pysähtynyt ohjelma käynnistetään uudestaan. Joskus pitää tehdä päätös, ettei ohjelmaa käynnistetä uudestaan. Kokonaisjärjestelmä toimii ilman, että minkään yksittäisen osan käynnissä olo on tarpeellinen. Valitussa arkkitehtuurissa ennakoidut ja ennakoimattomat pysähdykset tulevat osaksi järjestelmän normaalia ja suunniteltua toimintaa, eikä oleteta jatkuvaa toimivuutta. Arkkitehtuurin perusolettamus on pikemmin, ettei mikään laite tai verkon osa ole koko ajan päällä.

6.2.3 Tiedontallentaminen

Dokumenttialustassa tiedon tallentamisen perusyksikkö on tiedosto, jonka tallentamisessa käytetään kulloinkin saatavilla olevan käyttöjärjestelmän tukemaa tiedostorakennetta. Levytilaksi käsitetään mikä tahansa pysyväismuistiksi käsitettävä fyysinen laite, jossa tietokoneen käyttöjärjestelmä ylläpitää omaa tiedosto- ja mahdollista hakemistorakennettaan.

Dokumenttialusta estää tiedostojen fragmentoitumisen ja liialliset kirjoituskerrat niillä levytyypeillä, joilla sillä on merkitystä (Rudo 2011). Dokumenttialusta myös erottelee toiminnallisesti eri levytyypit kolmeen ryhmään sallittujen kirjoituskertojen lukumäärän suhteen:

- kirjoitetaan kerran,
- kirjoitetaan muutamia kertoja,
- kirjoitetaan monta kertaa.

Tämän lisäksi dokumenttialusta erottelee levyt loogisen sijainnin suhteen kolmeen ryhmään:

- paikallinen levy (fyysisesti samassa koneessa),
- verkkolevy (levytila tarjolla lähiverkossa tai yleisessä verkossa),
- siirtolevy (fyysisesti irrotettava levytila, enimmäkseen irti).

Kaikissa näissä tapauksissa dokumenttialusta olettaa, ettei levytila ole välttämättä koko ajan käytössä. Ilman levyäkin dokumenttialusta toimii, tosin rajallisesti. Dokumenttialusta kuitenkin tulkitsee virhetilanteeksi levyn, levytilan tai yhtenäisen levytilan puuttumisen, tietyissä tilanteissa jopa fataaliksi virheeksi ja lopettaa aktiivisen toimintansa. Alusta jatkaa toimintaansa automaattisesti tilanteen muuttumisen jälkeen.

Kaikki levylle tallennettu tieto salataan vähintään yhdellä parilla julkisen avaimen salakirjoitusavaimista. Tämä minimiavainpari on tietokonekohtainen, eikä se ole välttämättä tallennettu mihinkään muualle.

6.2.4 SPoF eli Single Point of Failure torjuttavana uhkana

SPOF:illa tarkoitetaan järjestelmän osan vikaantumista yhdessä pisteessä ja tämän vian vaikutusten leviämistä laajemmalle järjestelmään, pahimmassa tapauksessa estämään koko järjestelmän toiminnan (Lynch 2015, s. 212 - 216). Vaikka SPoF-ilmiötä terminä käytetään tekniikassa yleisesti, tässä työssä se rajataan koskemaan vain tietojärjestelmän ylläpidon aikana tapahtuvia virheitä.

Tietokonetekniikassa tämän tyyppisiä häiriöitä pyritään estämään muun muussa hankkimalla kaksoiskappaleita järjestelmän eri komponenteista, luomalla rinnakkaisia varajärjestelmiä ja varareitittämällä verkkoja (Technet 2017). Ensimmäisistä tietokoneista lähtien tietokonelaitteiden MTBF (Dear 2014) eli keskimääräinen vikaantumisaika on huomattavasti pidentynyt. Julkisuuteen vuotaneiden tietojen perusteella voidaan todeta, että ohjelmalliset ja varsinkaan tiedonsiirtoviat eivät ole samalla tavalla vähentyneet. Vian leviämisen estämiseen käytetään "ohjelmallisia murtosokkia" (Paavola 2018).

Yhtenä esimerkkinä ohjelmallisista murtosokista on sovelluksen lataaman aliohjelmakirjaston tarkkailu. Ennen aliohjelmakirjaston funktiokutsua tallennetaan kutsua edeltävä tilanne ja funktion suoritusta tarkkaillaan. Mikäli ajoaika tai resurssien käyttö on epänormaalia ja asetettujen rajojen ulkopuolella, koko aliohjelmakirjasto poistetaan muistista ja ladataan uudestaan levyltä. Tämän jälkeen jatketaan automaattisesti kyseisen funktion kutsua edeltävästä tilanteesta. Mikäli sama tilanne toistuu samanlaisena uudestaan, kyseinen mikropalvelu pysäytetään ja käynnistetään uudestaan ja mikropalvelun kutsu toistetaan automaattisesti. Mikäli tämäkin toistuu liian monta kertaa, jätetään kyseinen mikropalvelun kutsu kokonaan suorittamatta ja laitetaan kutsu ja sen sisältö omalle karanteenialueelleen.

Dokumenttialustan suunnittelussa SPoF:n torjunta on toteutettu siten, että mikä tahansa yksittäinen laite tai erillinen ohjelma voi olla määrättömän ajan pois päältä ilman, että kokonaisuuden toiminta käyttäjän näkökulmasta pysähtyisi. Tämä on mahdollista, koska kaikki alustan toiminnalliset ja dataa sisältävät osat on monistettu usealle toisistaan riippumattomalle tietokoneelle. Poikkeuksen tästä monistamisesta tekee tietysti käyttäjän oma tietokone tai mobiililaite. Mitä lähempänä alustan rajapinta sijaitsee käyttäjää ja käyttäjän laitetta, sen helpompi on erilaiset häiriötilanteet hoitaa järkevällä tavalla ilman SPoF-tilanteen syntymistä.

Julkisuudessa usein esiintyneiden palvelunestohyökkäyksien tehokkuus perustuu nimenomaan SPoF-ilmiön esiintymiseen käytetyissä arkkitehtuuriratkaisuissa. Dokumenttialustan arkkitehtuurissa SPoF-ilmiötä ei esiinny. Siksi palvelunestohyökkäykset dokumenttialustaa vastaan vaatisivat toimiakseen kokonaan uudenlaista suunnittelua.

6.2.5 Henkilötietojen tietosuoja - GDPR

"*GDPR tulee sanoista General Data Protection Regulation (yleinen tietosuoja-asetus). Se on uusi henkilötietojen käsittelyä sääntelevä laki, jota sovelletaan kaikissa EU-maissa 25.5.2018 alkaen.*

GDPR antaa paremman suojan henkilötiedoillesi ja enemmän keinoja hallita tietojesi käsittelyä." (https://tietosuoja.fi/gdpr)

Kaikki ne dokumentit, jotka vaativat tai sisältävät henkilötietoa tai henkilöä yksilöivää tietoa, vaativat erikoiskäsittelyn ennen tallentamista dokumenttialustaan. Tietosuojalain-

säädännössä määritellään asetuksen koskevan kuluttaja-asiakasta, eikä kaikkia fyysisiä henkilöitä, kuten esimerkiksi yrityksen työntekijöitä työtehtäviinsä kuuluvissa tehtävissään. (EUR-Lex 2016, s. 13) Dokumenttialusta on kuitenkin henkilötietoja sisältävä rekisteri, joten kaikkia henkilöitä käsitellään samalla huolellisuudella, tarkkuudella ja jäljitettävyydellä. Tietosuoja-asetus ei tarkasti yksilöi, mitä nämä henkilötiedot ovat. Asetuksen mukaan kaikki henkilöä yksilöivä tieto, tai tieto mistä voidaan päätellä tai tunnistaa tietty henkilö, on henkilötietoa. (EUR-Lex 2016)

Dokumenttialustassa henkilötietoa koskeva erikoiskäsittely on sähköisen allekirjoituksen hyväksikäyttö. Saman dokumentin voi allekirjoittaa kaikki ne, jotka on mainittu kyseisessä dokumentissa, tai jotka voidaan tunnistaa kyseisen dokumentin perusteella. Samalla he antavat maantieteellisesti kohdennetun (EU tai globaali) käyttöluvan. Koska dokumenttialusta salaa kaiken käsiteltävän tiedon, tiedon sijoituslupaa ei tarvita. Käyttölupa voidaan myös rajoittaa ajallisesti muodossa: alkuaika - loppuaika. Näitä aikarajoja voi olla useita. Allekirjoitukset ja tiedonkäyttörajoitukset tallennetaan dokumenttien metatietoihin.

Sähköinen allekirjoitus on myös hakuehtona metatiedoissa, jos ja kun henkilö haluaa dokumenttialustassa häntä koskevat tiedot itselleen. Myös dokumentin vastuuhenkilön tai henkilöiden allekirjoitukset ovat sähköisessä muodossa. Itse sähköisessä dokumentissa voi olla tarvittaessa käsinkirjoitettuja allekirjoituksia tai nimikirjaimia.

Ottamatta kantaa kulloinkin eri maissa olevaan lainsäädäntöön ja niiden mahdollisiin muutoksiin, dokumenttialustassa taataan yksilön mahdollisuus tulla unohdetuksi. Se toteutetaan salaamalla hänen tietonsa viranomaisen sitä tarkoitusta varten varaamilla "tuhoamisavaimilla". Tällöin itse tietoa ei siis tuhota, se ainoastaan ei ole enää luettavissa ilman viranomaisen tiedossa olevaa salaista avainta. Esimerkiksi aikaisemmin globaalisti saatavissa oleva tieto onkin luettavissa ainoastaan EU-alueella salausavaimia vaihtamalla. Tämä mahdollistaa myös henkilön oman mielipiteen vaihtamisen, eli esimerkiksi mahdollisuuden myöhemmin pyörtää lupa julkaista tai olla julkaisematta tietoa sähköisessä muodossa EU-alueen ulkopuolella.

Sähköistä allekirjoitusta käytetään kolmanteenkin tarkoitukseen, eli minkä tahansa oikeussubjektin, kuten yrityksen tunnistamiseen. Koska dokumenttialusta huolehtii päivitysten automaattisesta toimittamisesta, myös dokumentin saajien ja lukijoiden

oikeushenkilöiden allekirjoitus tallennetaan itse dokumenttiin tai tarkemmin sen meta-tietoihin. Tämä ei ole kuitenkaan aivan välttämätöntä tapauksissa, joissa ei haluta käyttää automaattista päivitystietoa. Tällöin päivitetyssä dokumentinvälityksessä voi kuitenkin esiintyä viivettä, jolloin dokumentin hyödyntäjän allekirjoitus on siis muuttuneen tiedon saannin kannalta nopeuttava tekijä.

Lohkoketjuteknologian perusperiaatteena on varmistaa kaiken tiedon säilyminen katkeamattomana ketjuna, joten tiedon lopullinen tuhoaminen ketjusta ei tule kyseeseen. Lohkoketjujen käyttö ei kuitenkaan estä tiedon salaamista homomorfisilla menetelmillä käyttökelvottomiksi kaikille, jotka eivät tiedä salaista avainparia. Lopullinen tuhoaminenhan myös estäisi virheellisesti tuhottujen tietojen palauttamisen. Tämä puolestaan olisi dokumenttialustan suunnitteluperiaatteiden vastaista.

6.3 Ohjelmointiperiaatteet

6.3.1 Yleinen periaate

Tietokoneohjelmien toimivuuteen vaikuttaa joukko tekijöitä, joita voidaan pitää teoriassa itsestäänselvyyksinä, mutta jotka eivät käytännössä sitä ole. Tietokoneohjelmat voivat toimia tai olla toimimatta verrattuna tietokoneohjelman loppukäyttäjän odotuksiin. Ohjelmoijan tehtävänä on pyrkiä tekemään virheetöntä koodia ja lisäksi muuttaa tekemäänsä koodia vastaamaan loppukäyttäjän odotuksia. Mitä isommasta ohjelmasta on kyse, sitä haastavampi jälkimmäinen ehto on. Loppukäyttäjien lukumäärän kasvaessa myös odotuksien määrä kasvaa ja vaatimukset monimutkaistuvat. Odotukset voivat olla osin ristiriitaisiakin.

Tässä työssä tarkastellaan sovelluksen toimivuutta kuitenkin vain tietokoneohjelmoijan näkökulmasta. Valitusta näkökulmasta tarkasteltuna valmistusvioiksi tulkittavissa olevia virheitä ovat muun muassa nollalla jakamiset, pyöristysvirheet, lukualueen yli- ja ali-vuodot, puskurin yli- ja alivuodot, ikuiset luupit, lukittautumiset kilpailutilanteessa, virheelliset tyyppimuunnokset, virhetilanteiden tarkistusten puuttumiset, käännösvirheet, tietoturvattomuus, tarpeeton hitaus, SPoF, mahdollisen korjauksen aiheuttamat virheet ja ohjelman kaatuminen rajallisten resurssien loppumiseen. Valmistusvikoja ovat myös asiakkaan määritysten vastainen toiminta, mutta sitä ei tässä tarkastella.

Ohjelmavirheet ja niiden esiintyminen koodissa ovat koodiin ohjelmoijan tekemiä (Hunt & Thomas 2011, s. 95 - 96). Siksi niiden esiintymistä voidaan yksittäisen ohjelmoijan huolellisuudella ja suunnitelmallisuudella hänen itsensä tekemässä koodissa estää, ja siten saavuttaa mahdollisimman virheetön koodi. Sovelluksen ajon aikana kaikki sen suorittama koodi ei kuitenkaan ole vain yhden ohjelmoijan, vaan satojen ellei tuhansien muiden ohjelmoijien tekemää. Kaikkien ensisijaisena päämääränä ei välttämättä ole ollut mahdollisimman virheetön koodi. Siksi dokumenttialustan rakenteen suunnittelun lähtökohtana on, etteivät tietokoneohjelmat tai niiden osat toimi juuri koskaan oikein ohjelmoijan näkökulmasta. Rakenteen suunnittelu mahdollistaa kuitenkin sen, että olemassa olevien virheiden vaikutus on mahdollisimman pieni ja käyttäjälle jopa näkymätön. Tämä suunnittelu mahdollistaa myös sen, että mahdolliset fyysiset laite- ja ympäristöviat eivät häiritse loppukäyttäjän tehtävää.

Edellä esitetty vaikuttaa monella tavalla dokumenttialustan rakenteeseen. Tässä työssä pyritään tarkastelemaan näitä tietoturvaan, luotettavuuteen ja ylläpidettävyyteen vaikuttavia tekijöitä. Yleisenä periaatteena on siis, ettei tehdä oletusta ohjelman osien tai ympäristön virheettömästä toiminnasta, vaan pyritään virheiden vaikutuksen minimointiin.

Lähdettäessä ratkaisemaan mitä tahansa ongelmaa ICT-alalla otetaan ensimmäiseksi kantaa käytettäviin työkaluihin. Työkaluista tärkein on ohjelmointikieli, ellei kyseistä ongelmaa varten ole jo tehty valmista ohjelmaa. Ratkaisuun vaikuttaa yhteisön tai yrityksen aikaisemmat päätökset tiettyjen työkalujen käytöstä. Tätä puoltaa yleisesti käytetty ja hyväksytty perustelu ohjelmointikielen ”Turingin yhteensopivuudesta”, eli sama ongelma on ratkaistavissa periaatteessa millä ohjelmointikielellä tahansa, jos ne ovat Turing-täydellisiä. Jos ohjelmointikieli on suosittu, sille löytyy myös paljon valmiita kirjastoja, joskus jopa niin paljon, että on vaikeuksia valita sadoista eri mahdollisuuksista soveltuvin. Usein onkin niin, että valmiit kirjastot rajoittavat kaikkein eniten tekemistä. Tämä on ongelmana varsinkin silloin, kun pitäisi liittää useita erilaisia järjestelmiä toisiinsa.

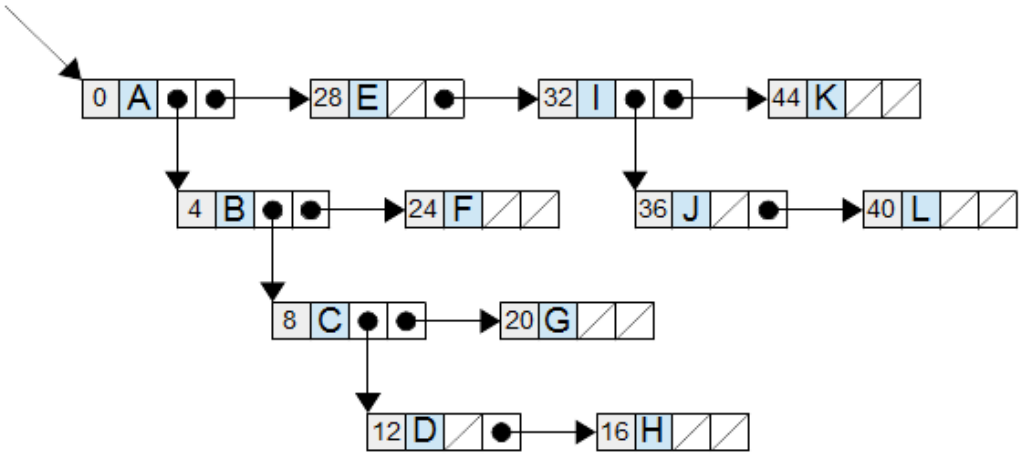
Tässä diplomityössä on pyritty ratkaisemaan annettua tehtävää riippumatta siitä, mitä työkaluja ja ohjelmointikieliä joudutaan käyttämään. On selvää, että kehitysympäristöjä työkaluineen ja ohjelmointikielineen ei ole syytä määritellä tai rajoittaa kehitystyössä tarpeettoman aikaisin. Silloin ne eivät pääsee rajoittamaan innovatiivisia ratkaisuja

omien rajoituksiensa takia. Ohjelmointitekniset syyt ovat olleet valintakriteereinä ja ne puoltavat heterogeenisten toimintaympäristöjen vuoksi useiden erilaisten kehitysympäristöjen ja työkalujen samanaikaista käyttöä. Ohjelmointiteknisiä kompromisseja joudutaan tekemään lopulta optimoidun toiminnan saavuttamiseksi, mutta vasta kun ajon aikaisia mittaustuloksia järjestelmästä on olemassa päätöksenteon perustaksi.

Kombinatorisen räjähdyksen (Grindal 2007; Kitzelmann 2009) estämiseksi heti tämän työn alussa on ylimmän tason menetelmäksi valittu funktionaalisen ohjelmoinnin lähestymistapa. Valinnalle ei ole pelkästään ohjelmateknistä perustetta, vaan myös perin inhimillinen syy: ohjelmat ovat ihmisten tekemiä ja ihmisille on vaikeuksia hallita liian monia yhtäaikaisia riippuvuuksia päin vastoin kuin verkossa toimivilla tietokone-ohjelmilla. Funktionaalinen ohjelmointitapa ohjaa suunnittelemaan ohjelmarakenteen ennen yhdenkään ohjelmointirivin kirjoittamista. Kokeilemalla aikaan saaduille ohjelmille puolestaan on tyypillistä versio-, ympäristö- ja kirjastoriippuvaisuus (Rouse 2016).

6.3.2 Rakenne

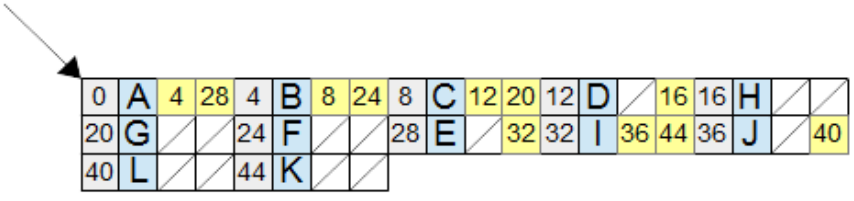
Dokumenttialustassa ylimmällä ohjelmallisella hierarkiatasolla kaikki tieto on tallennettuna listoina, jotka sisältävät koodin lisäksi myös dataa. Tällöin prosessin tila voidaan tallentaa tallentamalla pelkästään kyseinen lista. Kun lista tallennetaan käytettäväksi täsmälleen samassa tietokoneessa, voidaan listan serialisointia yksinkertaistaa ja nopeuttaa. Listan serialisointi tarkoittaa listan esitysmuodon muuttamista keskusmuistin esitysmuodosta tekstiksi. Lista koostuu alkioista, jotka voivat olla tietokoneen muistissa hajallaan pitkin keskusmuistia missä järjestyksessä tahansa. Alkioissa olevat osoittimet eli pointterit (nuolet kuvassa 4) osoittavat joko samalla tasolla tai alemmalla tasolla olevaan seuraavaan alkioon.



Kuva 4. Yksinkertaistettu kaavio listan rakenteesta.

Kuvan 4 esittämä rakenne voidaan esittää Lisp -ohjelmointikielen tekstimuodossa seuraavasti: (A (B (C (D H) G) F) E I (J L) K), missä A, B, C, D ja J voidaan tulkita tarpeen mukaan Lispin tavoin funktioiksi tai dataksi. Lispissä kunkin listan ensimmäinen nimi voi esittää funktiota. Listan määritelmä on rekursiivinen. Lista koostuu atomeista ja kukin atomi voi olla myös lista.

Listan alkiot tallennetaan puskuriin peräkkäin sellaisenaan ilman sisällönmuokkausta. Tätä varten lasketaan ensin kunkin alkion etäisyys tavuina puskurin alusta käymällä lista kokonaan läpi rekursiivisesti. Etäisyys (offset) tallennetaan itse alkioon. Kun alkiot kirjoitetaan puskuriin, vaihdetaan samalla kaikki osoittimien arvot alkuperäisistä alkio-osoittimista niiden osoittamien alkioiden offset-arvoihin.



Kuva 5. Lista koottuna yhtenäiseen puskuriin.

Samalla puskurissa (Kuva 5) olevalle alkiolle laitetaan lippu (flag) päälle osoittamaan alkion olevan osa isompaa itsenäistä muistivarausta eikä erikseen alkioittain varattu

muistialue. Puskurimuistialue on varattu käyttöjärjestelmän ytimen (kernel) hallitsemasta jaetun muistin (shared memory) alueesta. (Rana 2018) Muistialue on suoraan muiden prosessien käytettävissä tai kirjoitettavissa pysyvämpään muistiin tiedostoksi. Ikään kuin sivutuotteena saadaan keskusmuistin eheyttäjä (defragmentor), kun poistetaan alkuperäinen lista ja otetaan käyttöön yksi iso ja yhtenäinen muistialue. Keskusmuistin eheyttämisellä on sama tarkoitus kuin kovalevyn eheytyksellä eli toimintojen nopeuttaminen. Keskusmuistilla on tosin rajoitus verrattuna levymuistiin. Yhteen tiedostoon eli yhdellä nimellä osoitettavaan varattavaan alueeseen voidaan käyttää useampia pienempiä osia pitkin kovalevyä, kun taas keskusmuistissa kerralla varattavan ja osoittimella osoitettavan alueen pitää olla yhtenäinen. Tämän takia pitkään toimivilla ohjelmilla muistin pirstoutuminen (fragmentation) on epätoivottu tila sekä suorituskyvyn että muistitilan käytön kannalta (Labrosse 2002a, s. 273).

Dokumenttialusta voidaan käynnistää antamalla lähtötietoina prosessin tila. Tällöin dokumenttialustasta käynnistetään vain tietty funktio, jota voidaan suorittaa yhden tai useamman kerran. Tästä seuraa erittäin helppo, nopea ja automatisoitavissa oleva regressiotestattavuus. Erillisiä koodiin upotettuja testausohjelman pätkiä ei tarvita. Listan dynaamisuuden ja funktionaalisen ohjelmoinnin luonteen takia rivi kerrallaan ajettavan virheenkorjausohjelman (debugger) käyttö on hankalaa ja hidasta. Virheenkorjausohjelman käyttö vaatii myös oman erillisen spesifisen käännöksen suoritettavasta ohjelmasta, jota se hidastaa oleellisesti. Debuggerilla voidaan ajaa käännetyn ohjelman lähdekoodia askel kerrallaan. Tämän takia debuggeri ei näe ohjelmoijan kannalta tärkeitä funktioita, koska ne ovat listassa eivätkä käännetyn ohjelman lähdekoodissa. Näiden syiden takia debuggeri onkin osoittautunut tarpeettomaksi ja jopa vahingolliseksi, eikä sitä käytetä dokumenttialustan kehityksessä.

dokumenttialusta	n kpl, funktionaalinen data tai koodi
alustaohjelma	yksi suoritettava tiedosto ABI-rajapintaa varten
library - API	kirjastot / apufunktiot
käyttöjärjestelmä	MS, Linux, QNX, Android, iOS, MicroC/OS-II ...
tietokone	mobile, PC, serveri ...

Kuva 6. Dokumenttialustan rakenne.

Käyttäjälle näkyvä dokumenttialustasovellus käsittää kuvassa 6 näkyvät osat. Käyttöjärjestelmän kannalta dokumenttialusta on useita erillisiä prosesseja, joille on monistettu yhtä ja samaa suoritettavaa tiedostoa, mutta erilaisilla funktionaalisilla datoilla käynnistettyinä ja ajettuina. Kirjastot ja apufunktiot ladataan vasta sen jälkeen, kun alustaohjelma on tunnistanut tietokoneen käyttöjärjestelmän ja kirjastojen versiot.

Kun ohjelman kontrolli eli se, mikä lopulta käynnistetään suoritukseen ohjelmassa, on käännetyn ohjelman ulkopuolella datana, voidaan samaan ohjelmaan sisällyttää useita saman ohjelman eri versioita. Toisaalta mikään ei estä sisällyttää samaan suoritettavaan tiedostoon eri ohjelman eri versioita. Tämä puolestaan helpottaa itse suoritettavan tiedoston testausta eri ympäristössä tai käyttöjärjestelmässä kuin siinä, missä ohjelma tulisi lopulta toimimaan. Ongelmana Turing -täydellisien koneiden algoritmeissa on, ettei ohjelman päättymistä voida todistaa ja siten varmistaa mitenkään. Koskaan ei siis voida olla varmoja, loppuuko algoritmin toiminta tietyllä alkuehdolla vai ei. Algoritmin loppumisen varmistaminen edellyttäisi massiivista testaamista kaikilla mahdollisilla alkuarvoilla. Yleensä se on käytännössä mahdotonta suorittaa.

Pysähtymisongelman ratkaisuun ja pysähtymisen varmistamista varten kopioidaan tekoälyä sovellettaessa "Generative Adversial Nets" -ideaa (Goodfellow et al. 2014) algoritmin ulkopuolisesta kriittisestä tarkkailijasta, joka itsenäisesti tekee päätöksiä

algoritmin lopettamisesta tiettyjen kriteerien perusteella. Kriteerit on sille opetettu oikein lopettaneiden ajokertojen pohjalta. Tällöin tarkkailijalla eli kriitikolla on tietoa, mitkä kriteerit ja millä painoarvoilla ne ovat merkityksellisiä pysäyttämisehtoja missäkin yksittäistapauksessa. Käytännössä siis ei tarvita täysin varmaa tietoa siitä, pysähtyykö algoritmi itse vai ei. Tarvitaan vain käsitys siitä, milloin on viisasta pysäyttää algoritmi. Tavallisesti tänä kriitikkona toimii ihminen, joka toteaa jonkin tekemisen kestäneen liian kauan, tai ohjelman käyttäneen liikaa muistia tai muita tietokoneen rajallisia resursseja ja päättää keskeyttää prosessin. Ennen uudelleenkäynnistystä ihminen vielä päättää, millaisilla tilan arvoilla ja mistä algoritmin suoritus jatkuu.

Tarvitaan siis tapa pysäyttää prosessi eli algoritmi tarvittaessa ulkopuolelta ja jatkaa suoritusta hallitusti tietystä ennalta määrätystä algoritmin kannalta turvallisesta kohdasta. Jotta näin voidaan toimia, tarvitaan käyttöön tietokoneen keskeytyksistä tuttu menettelytapa (Silberschatz & Galvin 1998, s. 405). Tietokoneessa käynnissä olevilla prosesseilla on prioriteetti. Korkeampiprioriteettinen prosessi voi keskeyttää alempiprioriteettisen prosessin milloin tahansa. Tällöin alempiprioriteettisen prosessin tila laitetaan talteen ja korkeampiprioriteettinen prosessi ajetaan. Kyseisen prosessin suorituksen loputtua palataan takaisin keskeytettyyn prosessiin palauttamalla tallennettu tila. Sen jälkeen alempiprioriteettisen prosessin ajo jatkuu kuin mitään ei olisi tapahtunut.

Dokumenttialustassa ajon keskeytyksen aiheuttaa ulkopuolinen kriitikko-ohjelma. Tarkkailtava prosessi puolestaan on tallentanut oman suorituksensa aikana useita ajon aikaisia tiloja kriitikolle. Mikäli tarkkailtava prosessi loppuu normaalisti, kriitikko poistaa tallennetut tilat ja odottaa seuraavaa algoritmin ajokierrosta ja sen talletettavia tiloja. Tässä siis älykäs automaatio korvaa tarkkailevan ihmisen, vaikkakin hyvin yksinkertaisella tavalla. Algoritmin tilan tallentaminen on erityisen helppoa funktionaalisissa algoritmin ratkaisuissa, mutta se ei ole täysin mahdotonta proseduraalisissa ratkaisuissakaan, vaikkakin vaikeampaa.

6.3.3 Ohjelmointimenetelmät

Tietokoneohjelma on ajattelun tulos. Tulos on käännettävissä tai tulkittavissa tietokoneelle suoritettavaksi. Väheksymättä logiikkaa tai matematiikkaa jotkut ohjelmoijat

korostavat ohjelmallisessa ajattelussaan "kauneuden" merkitystä lukuisten ratkaisu-vaihtoehtojen karsinnassa (Oram & Wilson 2007, s. 39, 385). Siitä huolimatta niin sanottu ohjelmallinen ajattelu määritellään yleensä loogiseksi ja matemaattiseksi ajatteluksi. (Berry & Csizmadia 2016) Ohjelmallista ajattelua rajoittaa kone, joka antaa tarkasti määrittyvän rajapinnan toimivuudelle. Käytännössä toimivuus voidaan todeta joillakin, mutta ei välttämättä kaikilla lähtöarvoilla.

Ohjelmoinnilla on vahvasti empiirinen lähtökohta. Kuten muussakin empiirisessä tieteessä oleellista on havainnointi ja mittaaminen. Tässä havainnoinnin ja mittaamisen kohteena on tietokone.

Yksinkertaisena esimerkkinä siitä, mitä tietokoneen toiminnasta voidaan havainnoida, on koneessa tapahtuva reaalilukujen laskennan epätarkkuus. Tietokoneella - tai manuaalisesti - laskettaessa tärkeimmät laskutoimitukset ovat yhteenlasku ja kertolasku. Matematiikassa näillä kahdella laskutavalla on voimassa vaihdantalaki, liitäntälaki ja osittelulait. Tietokoneella laskentaa tehtäessä päädytään helposti tilaan, jossa nämä edellä esitetyt matemaattiset lait eivät päde tietokoneen käyttämillä reaaliluvuilla. Lopputulos muuttuu sen mukaan, missä järjestyksessä reaaliluvut lasketaan yhteen tai kerrotaan. Poikkeaman syy on tietokoneen laskennassa käytetyn reaaliluvun sisäisen ja ulkoisen esitysmuodon erossa: luvun sisäinen esitysmuoto on binäärinen, ulkoinen kymmenjärjestelmäinen (Petzold 1999, s. 342). Se numeroarvo, joka toisessa esitysmuodossa on tarkka, onkin toisessa esitysmuodossa epätarkka ja pyöristykset tehdään laskennan edetessä eikä vasta lopussa. Tietokone ei anna mitään varoitusta tai ilmoitusta näistä eroista tai niiden vaikutuksista. Ohjelmoija ei pysty korjaamaan tätä tietokoneen ominaisuutta, ainoastaan minimoimaan sen vaikutukset näkymättömiksi käyttäjälle.

Tässä työssä käytettäviä ohjelmointimenetelmiä tarkastellaan lähinnä siitä näkökulmasta, mitä virheitä niiden käytöstä voi aiheutua ja millä tavalla menetellen syntyneiden virheiden vaikutus voitaisiin piilottaa käyttäjältä. Virheitä ei siis pyritä poistamaan, koska se on mahdotonta, vaan virheiden vaikutus poistetaan, koska se on mahdollista.

Dokumenttialustan lähdekoodin kääntämisen aikana saadaan kaikkien sisäisten funktioiden suhteellinen osoite suoritettavan tiedoston alusta. Alustan käyttämät

ulkoiset kirjastofunktiot ja niiden osoitteet tulevat ladattua ajon aikana dynaamisesti kirjastoista, levyltä tai muiden sovellusten jo valmiiksi keskusmuistiin lataamina. Tämä edellyttää dokumenttialustan kykyä käyttää objektiorientoituneesta ohjelmoinnista tuttua polymorfismista ohjelmointitekniikkaa (Pierce 2002, s. 226).

Dokumenttialusta ei lataa käynnistyksen yhteydessä yhtään suoritettavan tiedoston ulkopuolista kirjastoa keskusmuistiin. Ainoastaan tehtävän suorittamiseen tarvittavat funktiot ladataan. Käyttöjärjestelmä tosin on koonnut funktiot useisiin satoihin kirjastoihin, jotka ovat minimiladattavia yksikköjä. Toisaalta joku toinen sovellus on jo voinut ladata API:n muistiinsa yhteiskäyttöä varten. Dokumenttialustasta täytyy kuitenkin luoda eri käännösversiot, jos ABI-rajapinnat muuttuvat.

Suoritettava tiedosto koostuu useasta osasta. Lähtökohtana on käyttöjärjestelmän spesifinen aloitusosa, jolla luodaan tarvittava koodin suoritusympäristö oletusarvoilla sekä koodi (code), alustettu data (data), alustamaton data (bss), pino (stack) ja keko (heap). Alkuosa kutsuu lähdekielisestä koodista tietyn nimistä, ympäristöstä riippuvaa pääfunktiota. Sen nimi on esimerkiksi "main", "program" tai "WinMain". Lähdekielisessä koodissa on kaikki lähdekoodin pääohjelman nimet yhtä aikaa käytössä. Ympäristöstä riippuva alkuosa päättää, mitä se milloinkin kutsuu. Tätä tietoa voidaan käyttää tunnistamaan ajossa käytetty ympäristö.

Kaikki laskenta eli käytetyt algoritmit ovat toteutettavissa Turingin koneella. Turingkone on abstraktio täysin mekaanisesta koneesta, joka pystyy toteuttamaan kaikki tunnetut algoritmit. Oikeastaan algoritmit määritellään laskennoiksi, jotka Turingin kone pystyy toteuttamaan. Tämä olettamus oletetaan oikeaksi, vaikka sitä ei ole voitu siksi todistaa. (Kripke 2013, s. 77 - 78) Ehkä vahvin todistus asialle on lambdakalkyyli (Turner 2006, s. 521), joka lähtee täysin erilaisista lähtökohdista kuin Turingin kone, mutta päätyy sen kanssa intuitiivisesti täysin yhteneviin lopputuloksiin.

6.3.4 Testaus

Testaus on oleellinen osa ohjelmistokehitystä, mutta tapa millä tavalla se suoritetaan, tulee muuttumaan. Koska mikään ohjelma ei ole virheetön, virheiden jäljitys (debugging) on myös oleellinen osa ohjelmistokehitystä. Se samoin kuin testaus tulee muuttumaan nykyisestä tavasta. Hajautetussa järjestelmässä virheiden jäljitys vaatii

tapaa, joka ei muistuta tavanomaista virheen jäljitystä käymällä läpi lähdekoodia rivi kerrallaan. Testaukseen tullaan käyttämään yhä enemmän ohjelmistoprojektiin käytetystä ajasta, joten kaikki toimenpiteet, jotka lyhentävät mitenkään testaukseen vaadittua aikaa, ovat suositeltavia.

Periaatteessa ohjelmointivirheitä ovat syntaksivirheet ja loogiset virheet (Bouras 2018). Dokumenttialustan suoritusympäristö on hajautettu ja heterogeeninen. Tällaisessa ympäristössä oleellista on kiinnittää huomioita osaan syntaksivirhetyypeistä sekä loogisiin virheisiin ja näiden virheiden korjaamiseen liittyvään testaamiseen. Dokumenttialustan suunnittelussa tavanomaiset ohjelmointikieliriippuvaiset syntaksivirheet eivät tarvitse erityistä huomiota, koska niiden havaitsemiseen ja korjaamiseen on olemassa valmiita työkaluja. Lisäksi ne eivät ole mitenkään spesifisiä hajautetulle ja heterogeenisille ympäristöille.

Ohjelmien välisten viestien virheet kuuluvat syntaksivirheisiin. Viestit voivat olla sekä koodia että dataa. Viestivirheet tosin ovat vaikeammin havaittavissa kuin vain ohjelmistokieleen liittyvät virheet. Viestien syntaksivirheet ovat tyypillisiä hajautetun ja heterogeenisen tuotantoympäristön ongelmia, koska kustannusten takia kolmannen osapuolen ohjelmia ei yleensä ole suunniteltu tai testattu riittävästi ennen käyttöönottoa (Morris 2012, s. 24). Testaus joudutaan tekemään vasta tuotantoympäristössä.

Jos ohjelmien välisen viestiliikenteen syntaksi on täysin vapaa eli määrittelemätön, sisään tulevassa viestissä ei silloin voi olla syntaksivirhettäkään. Määrittelemättömän viestin vastaanottaminen vaatii dokumenttialustalta tiettyyn tapaan rakennettua ja suunnittelua arkkitehtuuria. Siihen palataan tuonnempana mikropalvelua koskevassa alaluvussa 6.3.6. Internetissä toimivilla järjestelmillä oleellista on, että sisään tuleva tieto on mahdollisimman vapaata ja ulostuleva tieto mahdollisimman tarkkaan määriteltyä. Tätä periaatetta on kritisoitu sen epäyhtenäisen tavan vuoksi hoitaa ja käsitellä virheilmoituksia, ja siten altistaa järjestelmät hakkeroinnille. (Parker 2008; Buechler & Pingle 2009, s. 112 -113)

Loogiset ohjelmointivirheet ovat helposti todettavissa ja ratkaistavissa keskitetyissä ja homogeenisissä yhden toimijan ympäristöissä, koska niistä voidaan luoda mahdollisimman tarkasti tuotantoympäristöä matkiva kehitys- ja testiympäristö. Hajautetussa ja heterogeenisessä monen toimijan ympäristössä, jossa tapahtuu jatkuvasti

usean toimijan tekemiä muutoksia, testiympäristön luominen on käytännössä mahdotonta. Virtuaalikoneet, kontit (Slater 2015), ovat yritys homogenisoida ympäristöä sovellukselle. Dokumenttialustassa näihin ei voida tukeutua, koska ei aseteta rajoituksia dokumenttialustan käyttöympäristölle. Dokumenttialustan heterogeenisen ympäristön vuoksi testaus on tehtävä tuotantoympäristön käytön aikaisilla järjestelmillä. Testausta on luonnollisesti tehty ennen yleiseen käyttöön päästämistä, mutta testaus jatkuu tuotannossa, koska osaa testauksesta voidaan tehdä vain tuotantoympäristössä. Samoja testaustapoja, joita käytetään tuotantojärjestelmissä, käytetään myös kehitys-ympäristössä.

Testiympäristöissä voidaan käyttää virheiden jäljitysohjelmille käännettyjä ympäristöjä tavanomaiseen tapaan. Niitä ei voi käyttää tuotantoympäristöissä eikä siis hajautetun ympäristön testausjärjestelmässä. Esimerkiksi dokumenttialustassa tapahtuu outo virhe ranskalaisessa tietokoneessa, jossa ajetaan ranskankielistä toimittajan käyttöjärjestelmää. Muissa saman toimittajan käyttöjärjestelmän kieliversioissa kyseistä virhettä ei esiinny. Tavanomainen ratkaisu olisi, että luodaan täydellinen kopio ranskankielisine käyttöjärjestelmineen testausta varten, jossa yritetään testaamalla toistaa sama virhe ja korjata se. Korjauksen jälkeen toimitetaan asiakkaalle korjattu uusi versio testausta varten ja toivotaan, että virhe on poistunut. Asiakas voi itse testata uuden version omassa testiympäristössään.

Dokumenttialustassa toteuttava ratkaisu on kuitenkin toinen. Koska dokumenttialustassa testaus voidaan suorittaa tuotantojärjestelmässä, testaus suoritetaan kyseisessä ranskan-kielisessä koneessa tuotantojärjestelmässä. Näin saadaan selville se, oliko kyseessä kieliriippuvainen vai jokin muu kyseisestä koneympäristössä johtuva virhe. Syyksi voi paljastua esimerkiksi, että käyttäjälle näkymätön käyttöjärjestelmän sisäinen termin kirjoitusasu oli muutettu ranskankieliseksi.

Käyttöjärjestelmiä voidaan ja pitää päivittää jatkuvasti erinäisten ongelmien poistamiseksi. Päivitysten tekeminen on oleellista myös tuotantoympäristön tietoturvan ylläpitämiseksi.

Etätestaus asettaa erityisiä vaatimuksia dokumenttialustan rakenteelle. Ratkaisevaa tässä työssä on, että rakenteessa yhdistetään kehitys-, testaus- ja tuotantoympäristöt. Samalla yhdistetään yksikkö-, järjestelmä-, integrointi- ja hyväksymistestaus. Kaikki tehdään

samalla ympäristöllä. Testauksessa painotetaan luonnollisesti tietoturvallisuutta. Rajapintojen yhtenäinen määrittely mahdollistaa tämäntyyppisen testaamisen.

Tämä testaustapa perustuu alimmillaan funktiotasolle, joten ohjelmoijalla on oltava riittävä tietotaitotaso laatia yksittäinen funktio oikein. Yksittäisten ohjelmointirivien toimivuuteen huomion kiinnittäminen on tällöin liian "yksinkertaista" ohjelmallista ajattelua.

Samalla kun voidaan testata etänä yksittäistä dokumenttialustan osaa missä päin maailmaa tahansa, samalla tekniikalla voidaan päivittää mikä tahansa dokumenttialustan ajossa oleva ohjelma. Haluttaessa dokumenttialussa on mahdollista estää sekä etätestaus että etäpäivitys. Lähtöoletuksena on, että molemmat on kielletty. Päivitykset ja testaukset voidaan suorittaa sekä online- että offline-tapaan. Koska viesti voi sisältää sekä koodin että dataa, se voi sisältää myös uuden kohdekoneessa suoritettavan dokumenttialustan päivityksen. Vaikka päivityksiä tulee koko ajan lisää, kuitenkin dokumenttialustaohjelmasta on olemassa vain yksi perusversio, koska päivitys tehdään aina perusversioon. Tämä ominaisuus on suoraan seurausta dokumenttialustan arkkitehtuurista.

6.3.5 Tiedonsiirto

Tietokoneiden välinen tiedonsiirto on hyvin yksinkertaista, mutta erittäin vaikeaa tehdä riittävän varmaksi täysin automaattisena internetin kautta ilman ihmisen valvontaa ja ylläpitoa. Niin sanotun "kahden kenraalin paradoksi" (Gray 1978) kuvaa varman tiedonsiirron ongelmaa. Esimerkissä viestit kulkevat epävarmana pidettävän maaston yli lähettien avulla. Kenraalit eivät voi olla täysin varmoja hyökkäysajankohdan yhteisestä sopimisesta, vaikka viestejä välitettäisiin kuinka monta kertaa tahansa. Aina jää epäilys siitä, ettei viimeinen viesti mennytkään läpi toiselle kenraalille. Laajemmin ottaen "kahden kenraalin paradoksi" tarkoittaa, että tiedonsiirrossa ei riitä, että tieto siirtyy kahden tahon välillä. Tämän lisäksi molempien tahojen pitää olla yksimielisiä, että siirto on todella tapahtunut ja vain yhden kerran. Tiedonsiirto tahojen välillä sisältää epävarmuuden siitä, ettei viesti mennytkään perille. Internet on juuri kuten paradoksissa käytetty "vihollisen miehittämä" ja viestin välitykselle epävarma alusta, jossa ei voi olla koskaan varma viimeisen viestin läpimenosta.

Internet on syntynyt pikemminkin jatkuvan evoluution kuin täsmällisen suunnitelman kautta. Internetin arkkitehtuurina on heterogeeninen tietokoneiden muodostama verkko, jonka tehtävänä on kuljettaa datagrammeja (Anttila 2001, s. 114) mahdollisimman tehokkaasti ja joustavasti tietokoneesta toiseen. Datagrammi on pakettivälitteisen verkon perussiirtoyksikkö, jonka varmaa tiedonsiirtoa verkossa ei voida taata. Dokumenttialusta tarvitsee tiedostojen välitykseen minimissään datagrammivälityskerroksen toimiakseen. Datagrammivälityskerroksella tarkoitetaan OSI-mallin mukaista IP-verkkokerrosta (Network layer). Kyseessä on tilaton protokolla (stateless protocol), jossa tiedonsiirtoon osallistuvilla osapuolilla ei ole mitään tietoa toistensa tilasta käytettävän protokollan puitteissa. Tämä ei kuitenkaan estä sitä, että tiedonsiirtoon osallistuvilla osapuolilla on tietyllä todennäköisyysarvolla esitettävä tieto tiedonsiirto-osapuolten tilasta.

Käytettävä tiedonsiirto on teknisesti mahdollista toteuttaa verkkokerroksen ICMP-protokollan (ICMP 1981) IP-tason nopeilla viesteillä. Tunnetuin ICMP-protokollaa hyväksi käyttävä ohjelma on nimeltään Ping. Ping-ohjelma on työkalu, jolla voidaan kokeilla tietyn verkko-osoitteen (IP-osoitteen) olemassaoloa sekä viestin välityksen nopeutta ja luotettavuutta. Ping-ohjelma käyttää "Echo Request"-viestiä tarkistamaan, onnistutaanko saavuttamaan kohdetietokonetta, joka vastaisi "Echo Response"-viestillä. Kyseisellä työkalulla voidaan selvittää myös useita erilaisia verkko-ongelmia. Ping-ohjelma esimerkiksi antaa suoraan kyseisen tietokoneen vastausajan ilman ylimääräisiä sovelluksista ja tietokannoista johtuvia hidasteita. Se tulostaa tiedon siitä, kuinka monta eri verkkojen tai koneiden välisiä reititinhyppyjä on jouduttu suorittamaan viestin-välityksessä. Ping-ohjelmalla on myös mahdollista määrittää maksimiviestin koko välillä 32 – 65527 tavua, jonka ICMP-protokolla pystyy luotettavasti välittämään perille asti. Koska jos jokin matkalla oleva reititin pilkkoo viestin pienemmiksi osiksi kokoamatta niitä automaattisesti takaisin, niin vain osa päätyy perille. Kyseisen protokollan käyttöä rajoittavat tietoturvan vuoksi välttämättömät palomuurit, joko yritysten yleiset palomuurit tai tietokoneissa itsessään käytettävät palomuuriohjelmat. Vaikka Ping-ohjelma on tehokas ja yksinkertainen tapa selvittää ja saada selville verkko-ongelmia, se on myös tehokas ja nopea työkalu vihamieliselle hakkerille selvittää verkossa olevia tietokoneita. Tästä syystä sen käyttöä on rajoitettu tiettyihin tietokoneisiin ja verkon osiin. (Scambray et al., s. 34 - 43)

User Datagram Protocol eli UDP tai toisinaan viitattu UDP/IP (Postel 1980) on internetin yksi vanhimmista tiedonsiirtoprotokollista. Sitä käytetään muun muassa konferenssivideoimisessa, reaaliaikaisissa tietokonepeleissä ja kaikessa, missä tarvitaan tiedonsiirron käyttöön koko verkon kapasiteetti. Sitä käytettäessä hyväksytään tilanne, jossa osa viestejä voi kadota kokonaan. Esimerkiksi sovelluskerroksen NTP (Mills et al. 2010) käyttää kuljetuskerroksen UDP-protokollaa täsmällisen aikatiedon välittämiseen tietokoneiden välillä. Siirrettävän tiedon luonteen vuoksi ei ole tarpeellista saada läpi jokaista aikaleimaviestiä, ja viestin uudelleenlähettäminen on itseasiassa virhe. Koska protokolla ei sisällä minkäänlaista omaa virheenkäsittelyä tai pakettien uudelleen-lähetystä, lähettävän puolen tietopaketteja voidaan lähettää niin nopeasti kuin koneessa kytketty verkko antaa myöten. Protokolla ei odota minkäänlaista vastausta tai kuittausta vastaanottavasta päästä. Tällöin vastaanottavan pään ei tarvitse olla edes koko aikaa toiminnassa. Myöskään verkon hetkelliset katkokset tai vastaanottavan tietokoneen hetkelliset ylikuormitukset eivät vaikuta mitenkään lähettävään tietokoneeseen. UDP-protokollalla voidaan katsoa olevan periaatteena "lähetä ja unohda". Tällöin tämä protokolla on myös vähemmän riippuvainen verkon latenssista kuin esimerkiksi TCP.

Kuljetuskerroksen TCP-protokolla puolestaan tekee virheentarkistusta ja välille jääneiden pakettien eli viestin osien uudelleenlähetyksiä. Kyse on TCP-pinosta, jossa virheellisessä järjestyksessä tulleet datagrammipaketit kootaan automaattisesti oikeaan järjestykseen varmistamaan, että kaikki paketit saapuvat perille. TCP-protokolla kuvaillaan luotettavaksi ja virheenkorjaavaksi datavirran tuottavaksi palveluksi (Anttila 2001, s. 133 - 135). Useat internetissä yleisesti käytetyt sovellukset käyttävät juuri tätä protokollaa mukaan lukien World Wide Web, E-mail, File Transfer Protocol, Secure Shell ja monet muut. TCP-protokolla on myös yleisimmin edelleen reititettävä protokolla internetissä. Ongelmaksi tulee tosiasia, joka todistettiin jo vuonna 1975 ja esitettiin pankkiryöstöä suunnittelevien gangstereiden ongelmana (Akkoyunlu et al. 1975). Paremmin ongelma tunnetaan edellä mainittuna "kahden kenraalin ongelmana" tai paradoksina.

Dokumenttialustassa käytetään edelleenreitityksen vuoksi TCP/IP päällä olevaa HTTP -protokollakerrosta; se takaa laajimman valmiina ja käytössä olevan toimintaympäristön. Client-server -tyyppisen yhteydenoton sijasta käytetään peer-to-peer -tyypistä eli P2P-kommunikointia. Kommunikointitavan toi tunnetuksi Napster jo vuonna 1999 (Riehl

2001, s. 1765). P2P-kommunikointi on saanut kielteistä julkisuutta väitteistä, että kyseisellä tavalla siirretään laitonta materiaalia ja kuormitetaan siten verkkoa tarpeettomasti (Riehl 2001, s. 1769). Tämä on tietysti vastoin verkon tärkeää neutraliteettiperiaatetta. Tosin internetin neutraliteetti on muutenkin kyseenalaistettu useita kertoja (Lampila 2018). Palveluntarjoajat (ISP) ovat estäneet erinäisiä P2P -protokollia tai ainakin hidastaneet niitä. Myös yksityiset yritykset omissa verkoissaan ovat estäneet P2P-protokollien toiminnan taloudellisista ja suorituskykysyistä.

Hajautetussa järjestelmässä P2P-protokollan edut ovat kuitenkin kiistattomat suoritus-kyvyn, luotettavuuden, tietoturvallisuuden ja laajennettavuuden vuoksi (Braun 2017). Lisäksi lohkoketjuteknologian hyväksikäyttö dokumenttialustassa mahdollistaa yritysten taloudellisen arvon suojaamisen. Täysin ei kuitenkaan voida koskaan estää, etteikö joku talleta dokumenttialustaan kopiosuojattua materiaalia. Mutta jos tarkoitus on muu kuin laillinen varmuustallennus, kiinnijäämisen riski on liian iso taloudelliseen hyötyyn nähden.

Internet-liikenne on koko ajan tarkkailun alaisena ja epätavallista ja poikkeavaa voidaan hidastaa tai kokonaan estää. Tämän takia dokumenttialusta naamioituu käyttämään client-server-tyyppistä ratkaisua peittääkseen oman peer-to-peer -verkkonsa. Nämä internetissä olevat virtuaaliserverit palvelevat ainoastaan mahdollistaakseen client-server-tyyppisen yhteyden. Ne eivät osallistu millään tavalla siirrettävän tiedon sisältöön tai edes sen ohjaamiseen lähettäjän ja vastaanottajan välillä. Tämän takia ne ovat virtuaalisia kahdessakin mielessä: ne ajetaan virtuaaliympäristössä, eivätkä ne ole varsinaisia toiminnallisia servereitä. Mikäli HTTP-protokollakerrosta ei hyväksytä esimerkiksi yrityksen proxy-serverin kautta, otetaan käyttöön HTTPS-protokolla.

Olipa kyseessä mikä tahansa menetelmä tai prosessi, niin aina jokin osa siitä voi olla tapahtumatta tai tapahtua väärin. Siksi jos prosessissa on mahdollisimman vähän askelia, joiden täytyy seurata toisiaan kokonaisuuden takia, niin sitä vähemmän siinä on myös virheherkkiä kohtia.

Dokumenttialustassa siirretään tiedostoja, joiden maksimikoko on sama kuin TCP-datan maksimikoko IPv4-datagrammissa eli 65495 tavua (Bhunia 2006, s. 232). Siis kaikki siirrettävät tiedostot ovat maksimissaan noin 64 KB:n kokoisia. Talletettavan dokumentin maksimikoko on n * 64 KB, missä n:ää rajoittaa koko verkon

tallennuskapasiteetti. Dokumenttialustassa ei varsinaisesti ole mitään kokorajaa dokumentille. Dokumentin kokoa rajoittavat sitä käsittelevän alkuperäisen ohjelman rajoitukset, muun muassa Excel, Adobe Reader ja Word.

Asynkroninen tiedonsiirto ei odota saavansa mitään vastausta vastaanottavalta taholta, eli se ei jää odottamaan vastausta. Siksi on syytä kyseenalaistaa Coulourisin, Dollimoren ja Kingbergin väite, että asynkronisessa tiedonsiirrossa ei ole väliä, toimiiko vastaanottaja synkronisesti vai asynkronisesti. Ainoana kritiikkinä asynkroniselle vastaanotolle he esittävät itse ohjelman teon monimutkaisuuden, ja sen takia sitä ei ole otettu käyttöön nykyisissä tiedonsiirtojärjestelmissä. (Coulouris et al. 2005, s. 134) Käyttäjät voivat itse todeta synkronisen tiedonsiirron ongelman eli vastaanottopään virhetilanteen surffatessaan verkossa. Tällöin selain näyttää jumiutuneen, vaikka selain ei tee mitään muuta kuin odottaa (timeout) seuraavaa tiedonsiirtoprosessin osaa tapahtuvaksi.

Yleisesti ottaen asynkronisessa tiedonsiirrossa olisi selkeämpää tarkoittaa sekä lähettävän että vastaanottavan osapuolen ei-blokkaavaa eli asynkronista käyttäytymistä. Tämä sama asia tulee esille Coulourisin, Dollimoren ja Kingbergin kirjoituksessa vikasietoisista palveluista, jossa jätetään vikasietoisuuden tarkastelun ulkopuolelle itse tiedonsiirron virheet (Coulouris et al. 2005, s. 615). Voidaan tietysti väittää, että tiedonsiirtovirheet eivät kuulu palvelulle itselleen. Mutta jos palvelua ei voida käyttää asynkronisella tavalla, vaan ainoastaan synkronisella tavalla, se lienee palvelun vika.

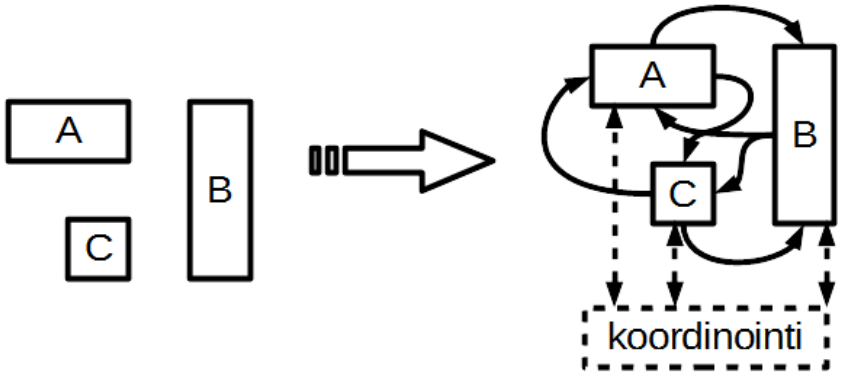
Mikäli dokumentti on suurempi kuin 64 KB, niin tiedostonsiirto tapahtuu paloittain asynkronisesti. Lähettävä pää ei välttämättä odota mitään vastausta vastaanottavasta päästä, vaan katkaisee yhteyden tai lähettää seuraavan viestin simuloiden kärsimätöntä käyttäjää, joka ei jaksa odottaa mitään kuittausviestiä toisesta päästä. Tällä tavalla sama dokumenttialustan käyttämä viestitysprotokolla toimii sekä UDP-pohjaisena että ICMP-pohjaisena. Viestitysprotokolla toimii myös normaalin sähköpostin kautta. Tekoälyä käytetään varmistamaan viestityksen riittävä hitaus eli yksittäisten viestien todennäköinen perillemeno. Tämä vaati jatkuvaa verkon tilan selvitystä ja muutoksiin sopeutumista. Tekoälyä käytetään myös "kahden kenraalin ongelman" riittävän varmaan ratkaisuun. Tietoa siirrettävistä tiedostoista eli metatietoa siirretään vain riittävän varmuuden saavuttamiseksi itse tiedostojen siirrosta. Vaikka tieto kulkee ennalta

määrättyyn yhteen suuntaan, niin metatieto tiedosta tai sen puutteesta pitää aina kulkea molempiin suuntiin.

6.3.6 Mikropalvelut

Mikropalveluperiaatteen (microservice, microservice architecture, cloud-native architecture) käyttö on tapa suunnitella sovelluksen tietoarkkitehtuuri (Syrjä 2016, s. 11: Stetson 2017, s. 1 - 5). Mikropalveluarkkitehtuuri soveltuu keskitettyä, monoliitista arkkitehtuuria (Pressman 2005, s. 267) paremmin hajautettujen järjestelmien suunnitteluun ja toteutukseen yksityisissä tai julkisissa pilvissä toimivissa sovelluksissa (Garriga 2017). Netflix:ä, yhdysvaltalaista tilausvideopalvelua kymmenille miljoonille käyttäjille tarjoavaa yritystä, pidetään mikropalvelujen johtavana edelläkävijänä (Hämäläinena 2016; Netflix OSS 2016).

Jotta voidaan helpottaa laajemman kokonaisuuden tekemistä, se paloitellaan osasysteemeiksi (Kuva 7). Paloittelua tehdään niin kauan, kunnes saavutetaan tila, jossa osasysteemin käyttäytyminen on ennakoitavissa ja järjestelmällistä. (Järvinen & Kerola 1978, s. 11) Tekotapa mahdollistaa laajojen kokonaisuuksien rakentamisen hallitusti.



Kuva 7. Joukko erillisiä osasysteemejä liitetään keskenään kommunikoivaksi laajemmaksi kokonaisuudeksi. (Mukaillen Järvinen & Kerola 1978, kuvio 1.5).

Mikropalvelu on kommunikoiva osasysteemi. Se on määritelty ja se tekee vain yhden asian; sen se tekee hyvin. Etuliitteellä "mikro" on haluttu korostaa palvelun olevan

hienosyisempi eli pilkottu pienempiin osasysteemeihin kuin mitä internet-palvelut (webservices) ovat aikaisemmin olleet. (Slater 2015)

Netflixin järjestelmäarkkitehtinä pitkään toiminut Adrian Cockcroft esittää mikropalvelujen pienentämistä edelleen vielä pienemmiksi osiksi eli lambda-funktioiksi (Cockcroft 2017). Osuvampi termi olisi ehkä nanopalvelu (nanoservice) (Rotem-Gal-Oz 2014). Mikropalvelut, nanopalvelut tai lambda-funktiot tarvitsevat erityisen toiminta-ympäristön toimiakseen.

Yhdenlaista mikropalveluympäristötyyppiä kutsutaan kontiksi (container) (Slater 2015). Kontin ideana on toimia ajoalustana, joka voidaan siirtää erilaisiin ympäristöihin, ja siten mahdollistaa esimerkiksi ohjelmiston kehitystyön ja testauksen tekemisen kokonaan omassa ympäristössään. Valmiiksi testattu kontti mikropalveluineen voidaan kopioida suoraan pilvipalveluihin. Tarkoituksena on siis lyhentää edelleen aikaa, mitä DevOps-kulttuurin mukaisesti ohjelmiston jakelukuntoon saattaminen ilman kontteja hallitusti kestää. (Lwakatare 2017. s. 40 - 43)

Tässä työssä dokumenttialusta tarjoaa alustan itsenäisille tapahtumaohjautuville funktioille, joita voidaan kutsua nanopalveluiksi. Alusta pitää automaattisesti huolen myös tietoturvasta, ohjelmaversioiden hallinnasta ja jakelusta, yleisestä toiminnan koordinoinnista, tiedonsiirrosta ja tapahtuvien virheiden tai poikkeamien käsittelystä.

6.3.7 Hajautettu tietokanta

Dokumenttialustan suunnittelussa on otettu huomioon, että pelkästään tiedon suuren määrän vuoksi se pitää hajauttaa. Tämä datavaranto tukeutuu globaaliin internet-alustaan, jossa jo nyt 99 % tiedosta siirtyy paikasta toiseen. Digitaalisen datan liiketoimintamallissa oleellinen osa on tiedon jakamista. Digitaalisen tiedon arvoon vaikuttaa se, kuinka hyvin se pystytään jakamaan. Tässä työssä on vältelty tekoäly-termin käyttämistä jakamisessa, koska se ei sinänsä tuo tarvittavaa konkreettista toteutustapaa esille. Myös termi "ohjelmistorobotiikka" tässä yhteydessä sisältää paljon tarpeetonta.

Yksinkertaisuudessaan tiedon pitää olla koko ajan saatavilla käyttäjillä. Samoin informaatio muuttuneesta tiedosta pitää kulkeutua käyttäjille läpinäkyvästi. Hajautettu

tietokanta toteuttaa nämä vaatimukset. Tietoturvan kannalta tiedonkäsittely on syytä suorittaa salattuna muuttamatta sitä selväkieliseksi missään vaiheessa. Homomorfiset salausmenetelmät mahdollistavat tiedon käsittelyn salattuna. Julkisella avaimella salattu kysely kohdistetaan samoin julkisella avaimella salattuun tietoon, jolloin kyselyn tulos on saatavissa selväkieliseksi ainoastaan kyselyn tekijän salaisella avaimella.

6.3.8 Optimointi

Knuthin mukaan liian aikainen optimointi on erittäin haitallista ("*root of all evil*"). Samalla sivulla hän kuitenkin toteaa, että niinkin pieni kuin 12 prosentin parannus menetelmään riittää perusteluksi tehdä optimointia millä tahansa tekniikan alalla. (Knuth 1974, s. 268) Tällöin jopa syrjityn hyppylauseen (go to) käyttö on sallittu tämän tavoitteen saavuttamisessa.

Ohjelmistokehityksen kannalta on hyödyllistä ensin pyrkiä toimivaan ratkaisuun ja vasta sitten ohjelman suorituskykymittausten jälkeen optimoida tarvittavat funktiot. Polyformismia hyväksi käyttäen voidaan jättää molemmat funktiot tai vaikka n-kappaletta funktiota voimaan yhtä aikaa ja tehdä päätös kulloinkin käytettävästä funktiosta käännettävän koodin ulkopuolella. Rakenne mahdollistaa myös testaamisen automatisoinnin eri funktioversioiden välillä ja niiden tuloksien vertailun ilman ohjelmamuutoksia. Samoin mahdollistuu eri funktioversioiden välinen vertailu normaalin käytön aikanakin tuotantoympäristössä.

Korkean tason kieli voi helpottaa tietokoneen ohjelmointia ja tehdä koodista siirrettävän tietokoneiden välillä, mutta se ei koskaan tee siitä tehokkaampaa kuin symbolisella konekielellä (assembler) tehtäessä (Petzold 1999, s. 353; Oram & Wilson 2007, s. 127). Symbolisella konekielellä voidaan tehdä kaikki, mitä keskusyksiköllä muutenkin voidaan tehdä. Tosin tämä ei rajoitu pelkästään keskusyksikköön, vaan symbolisella konekielellä voidaan ohjata myös yleistyviä grafiikkayksiköitä (GPU), digitaalisia signaaliprosessoreita (DSP) ja digitaalisia mikropiirejä (FPGA). Laitteiden ohjelmoitavuuden takia esimerkiksi grafiikkayksiköitä voidaan käyttää muuhunkin kuin grafiikan tuottamiseen. (Kyrnin 2018) Tehokkuuden tavoittelussa syntyy myös uusia ratkaisuja, kuten esimerkiksi WebAssembly (Wasm, WA), jossa laitteiston sijaan suoritusalustana on selain (WebAssembly 2017).

Dynaamisen muistin käyttö tai lähinnä sen käyttötavan valinta on suorituskyvyn optimoinnin kannalta erittäin tärkeää. Kriteerinä, miten ohjelma paloitellaan nano-palveluiksi, on toimintojen lisäksi tapa, miten dynaamista muistia käytetään. Hyvin eri tavalla dynaamista muistia käyttävät palvelut eriytetään omiksi ajettaviksi ohjelman-osiksi. Ääripäässä ovat ohjelmanosat, joissa ei käytetä nopeuden ja tietoturvan takia lainkaan dynaamista muistia.

6.4 Dokumenttialusta konepajanäkökulmasta

Koska tässä työssä tarkoituksena on laatia dokumenttialustaohjelman suunnitteluperi-aatteita ohjelmointiteknisestä näkökulmasta, varsinaista ohjelman käyttöliittymää ei ole esitetty; työ ei ole dokumenttialustan manuaali. Terästuotantoon kohdistuvan tapaus-tutkimuksen vuoksi tässä luvussa kuitenkin hahmotellaan lyhyt katsaus dokumenttialustan käyttötapoihin. Katsausta on pidettävä suuntaa-antavana, koska konepaja-kohtaiset ja varsinkin käyttäjäkohtaiset erot voivat vaikuttaa vaadittaviin toteutuksiin. Lisäksi luvussa arvioidaan suunnitelman mukaisesti toteutettavan dokumenttialustan hyötyä sen käyttöönottajalle terästeollisuudessa.

Dokumenttialustasta on hyötyä konepajalle nimenomaan silloin, kun dokumentteja ja niissä olevaa tietoa jaetaan eri tahojen kesken. Jos dokumenttialustaohjelma on käytössä vain yhdellä konepajalla, ohjelman funktio on yksinkertaisimmillaan ainoastaan arkistomainen tiedon varastointi. Teollisuuslaitoksen sisäverkkoon voidaan luonnollisesti asentaa ohjelma eri osastojen tai prosessien käyttöön niiden omille tietokoneilleen. Silloin nämä voivat ylläpidon lisäksi jaella tietoa keskenään.

Varsinainen hyöty dokumenttialustasta konepajalle on silloin, kun sen yhteistyö-kumppaneilla, esimerkiksi koko alihankintaketjun tai komponenttitoimittajien kaikilla toimijoilla on käytössään dokumenttialusta. Tietoa toimijoiden kesken voidaan jakaa siten, että tiedonsiirto on turvallinen. Valmistettavan tuotteen kaikki oleellinen ja mahdollisesti myöhemmin tarvittava tieto sen jostain ominaisuudesta on jäljitettävissä helposti. Kaikki ne tahot, jotka osallistuvat tuotantoon ja joilla on oikeus tietää tuotanto-prosessista, saavat muuttumattoman tiedon käyttöönsä. He voivat lisätä tietoa, eivät muuttaa olemassa olevien data-aineistojen sisältöä. Mikäli dokumenttialusta on ladattu myös esimerkiksi toimilupia myöntäville ja ammatillisia sertifikaatteja laativille

tahoille, yksittäisen konepajan kannalta tämä merkitsee tiedon tallentamisen automatisoitumista dokumenttien skannaamisen sijaan. Ajantasainen ja nopeasti päivittyvä tieto on kaikkien osapuolten edun mukaista.

Koko tuotteen elinkaaren kannalta hyvä olisi, jos myös sen ylläpitoon, huoltoon tai korjaamiseen liittyvät tahot ottavat dokumenttialustan käyttöönsä. Käytön aikana syntyneiden dokumenttien avulla ongelmatilanteet tai -kohdat hälyttävät automaattisesti parhaassa tapauksessa jo ennen ongelman aktivoitumista tai eskaloitumista.

Dokumenttialusta on terästuotantoketjun sovellus itsessään, mutta se on samalla ohjelmakomponentti konepajan käyttämille, sisällöltään kokonaan toisen tyyppisille sovelluksille, joita tarvitaan tiedon siirtämiseen alihankintaketjun tai yhteistyöyritysten välillä. Se on myös itsenäinen integrointityökalu sovellusten välillä. Lisäksi sitä voidaan käyttää integroimaan yrityksen sisäisiä tietokantoja. Hajautetun arkkitehtuurinsa ja data-ohjautuvuutensa ansiosta se soveltuu myös integroimaan useiden eri yritysten sovelluksia ja tietokantoja yhteen.

Konepaja ottaa dokumenttialustaohjelman käyttöönsä yksinkertaisesti lataamalla sen esiasennettuna koneeseensa tai koneisiinsa. Salauslauseella tunnistaudutaan ohjelman käyttäjäksi ja siten saadaan pysyvä, yksilöllinen identiteetti tietokoneelle tai sen käyttäjälle. Tiedon syöttö voi tapahtua monin eri tavoin: skannaamalla papereita, kopioimalla jo digitaalisessa muodossa olevia dokumentteja, IoT-laitteiden tallentamia tietoja kopioimalla ja tietysti käyttäjien itsensä sisään syöttäminä. Dokumentteja ja yli-päänsä tietoa voidaan syöttää ohjelman käyttöön miten paljon tahansa, ainoastaan tallennuskapasiteetti on rajoittava tekijä.

Tallennuskapasiteetin mittakaavavertailuksi esitetään seuraavassa muutamia esimerkkejä dokumenttitietokannoista. Oulun yliopiston 2013 käyttöön otettu digitaalisia opinnäytteitä ja tieteellisiä julkaisuja sisältävä tietokanta "Jultika" on koolta 51 Gt käsittäen 13555 dokumenttia (Skiftesvik 2018). Niin sanottuna Panama-papereina tunnettu tietovuoto käsitti 2,6 Tt dataa 11,5 miljoonan dokumentin muodossa (Obermaier et al. 2016). Mikäli halutaan ostaa erikseen dokumenttialustan tietojen tallennuksesta vastaava esiasennettu tietokone yrityksen sisäverkkoon, oheinen esimerkkikokoonpano voisi olla mahdollinen: keskusmuistia 32 Gt, levymuistia 24 Tt ja riittävä verkkokapasiteetti. Yksikön hinta on silloin noin 2600 euroa ALV 0%

(PCSpecialist 2018). Tällöin yhdessä esimerkkilaitteistossa on tallennuskapasiteettia 470-kertaisesti verrattuna Jultika-kantaan ja 9-kertaisesti Panama-papereihin. Vaikka kapasiteetti on konepajalle varmasti riittävä, reduktanssin takia yksiköitä kannattaa hankkia useampia. Ne on syytä sijoittaa fyysisesti erilleen, mutta yhdistettyinä toisiinsa nopealla erillisverkolla, jotta ne eivät kuormittaisi liikaa yrityksen sisäverkkoa erikois-tilanteissa. Erikoistilanne on esimerkiksi kokonaan uuden, esimerkkinä käytetyn tietokonelaitteiston liittäminen sisäverkkoon. Uusi kone päivittää itsensä ajan tasalle muista verkon tietokoneista automaattisesti. 24 Tt kestää latautua 1 Gb:n sisäverkossa noin 3 vuorokautta. Siksi tietokoneet ovat keskenään 10 Gb:n erillisverkossa. Tällöin latautuminen tapahtuu muuta verkkoa häiritsemättä vain yön yli.

Dokumenttialustaohjelma itsessään on freeware. Se on asennettavissa vapaasti internetistä ilman rajoituksia globaalisti. Tällä voi olla merkitystä, mikäli esimerkiksi komponenttitoimittajat ovat ulkomaalaisia yrityksiä.

Kuten digitalisaatiossa yleensä, myös dokumenttialustan käyttöönoton jälkeen useita manuaalisia työvaiheita dokumentaation hallinnassa muuttuu radikaalisti tai jää kokonaan pois. Toisaalta tämän tapainen ohjelmistoarkkitehtuurin hyväksikäyttö myös helpottaa ICT-alalla jatkuvasti kasvavaa työvoimapulaa automatisoimalla monia ylläpidon tehtäviä. Dokumenttialustan arkkitehtuuri on internetissä toimiessaan tunteeton myös nykyisin yleistyneille yhteen pisteeseen kohdistuville palvelunestohyökkäyksille, joita on paljon halvempaa tehdä kuin torjua.

7 JOHTOPÄÄTÖKSET

Dokumenttialusta nojaa käännettyyn pääohjelmaan, jonka toteutus on ollut epätavallinen ja sen vuoksi varsin vaativa. Ohjelman etu on siinä, että se taipuu mihin tahansa tiedon tallentamiseen ja siirtoon pelkästään kulloisenkin datan ohjaamana. Dokumenttialusta puolestaan on suunniteltu käyttäjälle helpoksi tai suorastaan näkymättömäksi, mikäli kattava automaatio halutaan toteuttaa.

Työssä on pyritty esittämään toimivia ratkaisuja yleisiin tietojärjestelmiä haittaaviin ongelmiin. Näitä ovat muun muassa sähkökatkokset, kovalevyn hajoamiset, verkkoyhteyksien katkeilu, tietokoneohjelman ennenaikainen pysähtyminen, tietokoneohjelman ikuinen luuppiin jääminen, tiedonsiirron epäonnistuminen. Ratkaisu toteutetaan automatisoimalla virhetilanteiden korjaaminen ilman ylläpitohenkilöstön tarvetta.

Ohjelmistovaatimukset, ohjelmistotuotantomallit, ohjelmointikielet, erilaiset tietoliikenneprotokollat, virheenjäljitystavat ja ohjelmointiympäristöt aiheuttavat usein eriasteisia rajoituksia valittaville ohjelmistoteknisille ratkaisuille. Siksi tässä työssä lähtökohdaksi on otettu mahdollisuus tehdä tietokoneessa ja verkossa toimiva ainoastaan puhtaasti ohjelmistotekninen ratkaisu, jota ohjelmistotuotantoprosessin muut tekijät eivät rajoita. Kyse on siis ideaalimallista.

On itsestään selvää, etteivät kaikki tallennetut dokumentit ole samanarvoisia lopputuotteen laadun kannalta. Laadun kannalta merkityksellisin dokumentti voi kuitenkin paljastua vasta tuotteen tai rakenteen pitkänkin käytön jälkeen. Tämän takia dokumenttien määrällistä tallennusta tai tallennusmuotoa ei ole rajoitettu millään tavalla.

Dokumenttien hallinnassa suurin etu saadaan, jos kaikkien kyseiseen kohteeseen liittyvien dokumenttien tuottajilla ja laatijoilla on käytössään dokumenttialusta. Silloin dokumenttien manuaalinen moninkertainen syöttö dokumenttialustaan poistuu. Tässä työssä ei ole otettu kantaa työvoima- tai talouspoliittisiin kysymyksiin, vaan tehty ainoastaan malli optimaaliseen ohjelmatekniseen toteutukseen dokumenttien hallinnassa.

Toteutusmenetelmissä kyberturvallisuuden merkitys on ollut ratkaisevaa.

8 YHTEENVETO

Työn ensisijaisena tarkoituksena on ollut kehittää ja esittää sellaisen sovellusohjelman suunnitteluperiaatteet, jolla voidaan tallentaa ja jakaa teräsrakenteiden tuotannon ja koko elinkaaren käsittäviä dokumentteja tietoturvallisesti. Aihe liittyy taustoittavana osana Lapin ammattikorkeakoulun ArcDigi-hankkeeseen.

Menetelmävalinnat ja -kehitys ovat työssä keskeisellä sijalla. Ohjelmoinnissa on käytetty lohkoketjuteknologiaa ja siihen sidoksissa olevia tietoteknisiä ratkaisuja. Näistä oleellisimpia ovat tiedon tallentamisen ja tiedonsiirron varmistamismenetelmät, hajautettu tietokanta, funktionaalinen ohjelmointi ja älykäs automaatio.

Suunniteltavaa sovellusohjelmaa kutsutaan dokumenttialustaksi. Sitä ei ole pilotoitu, mutta sen toimintavarmuus voidaan taata. Se nimittäin perustuu työn tekijän aiemmin toteuttamalle käännettävälle pääohjelmalle, jonka toimivuus on testattu useissa integrointitehtävissä. Pääohjelman monikäyttöisyys, geneerisyys, johtuu siitä, että data määrittelee, mitä ohjelma tekee. Tässä nimenomaisessa tapauksessa se tallentaa ja siirtää digitoituja dokumentteja luotettavasti internetissä.

Teräsrakenteiden dokumentointi on yhteydessä paitsi niiden laadun varmistamiseen myös koko elinkaaren aikana tapahtuneiden muutosten ja korjausten jäljitettävyyteen. Binääritiedostoina tallennettavien dokumenttien määrää ei tarvitse hajautetun tietokannan vuoksi millään lailla rajoittaa, eikä alkuperäiselle esitysmuodolle ole erityisvaatimuksia.

Tallennettavan ja siirrettävän tiedon salausmenetelminä voidaan käyttää symmetrisiä tai epäsymmetrisiä salausavaimia. Lohkoketjuteknologian ansiosta alkuperäinen dokumenttialustaan tallennettu tieto pysyy muuttumattomana, tietoa voidaan ainoastaan lisätä, ei poistaa. Asynkroninen tiedonsiirto puolestaan poistaa tiedonsiirrossa toisinaan ilmenevät pullonkaulat tai verkon häiriöistä johtuvien keskeytysten aiheuttaman haitan. Tiedonsiirron varmistamisessa on kiinnitetty huomiota niin sanottuun "kahden kenraalin ongelmaan", eli huoleen siitä, että viesti ei olekaan mennyt perille ja tallentunut järjestelmään.

Älykäs automaatio on toteutettu funktionaalisella ohjelmoinnilla ja laskentakapasiteettia kasvattavalla rinnakkaisuudella. Lisäksi useita ylläpitotehtäviä on automatisoitu.

Koska dataviesti käsitetään suoritettavana ohjelmakoodina, samaa ohjelmakoodia voidaan ajaa suorittamaan rajaton määrä eri sovelluksia. Monikäytettävyytensä lisäksi esitettyjä suunnitteluperiaatteita noudattava ohjelma on tietoturvallinen. Vikaantuneista ja yhdestä vikapisteestä toimimattomiksi eskaloituneista tietojärjestelmistä kertovissa uutisissa tietoturvan puute on se, mihin tässä työssä on kiinnitetty erityishuomiota. Työssä esitetyillä menetelmillä voidaan estää muun muassa nykyiset palvelunesto-hyökkäystavat.

Tietoturva ei ole mikään yksittäinen ihminen, laite, algoritmi tai ohjelma, joka tekee järjestelmästä turvallisen. Tietoturva on pikemminkin yksi niistä rakennuspalikoista, joista koko järjestelmä on rakennettu. Toiveena olisi päästä rakentamaan edellä mainitun kaltaista laajaa ilmaisohjelman kokonaisuutta, koska sen toteuttaminen olisi mahdollista myös käytännössä eikä vain ideaalimallissa.

LÄHDELUETTELO

Ailisto, H., Helaakoski, H., Dufva, M. & Tuikka, T., 2017. Tuottoa ja tehokkuutta Suomeen tekoälyllä. Espoo: Teknologian tutkimuskeskus VTT Oy, VTT- Policy Brief No. 1/2017. 8 s.

Akkoyunlu E. A., Ekanadham K. & Huber R.V., 1975. Some Constraints and Tradeoffs in the Design of Network Communications [verkkodokumentti]. SIGOPS Oper. Syst. Rev., 9(5):67{74, November]. Saatavissa http://dsg.tuwien.ac.at/linksites/teaching/courses/AdvancedDistributedSystems/download/1975_Akkoyunlu,%20Ekanadham,%20Huber_Some%20constraints%20and%20tradeoffs%20in%20the%20design%20of%20network%20communications.pdf [viitattu 17.8.2018].

Anttila, A., 2001. TCP/IP tekniikka. 2. painos. Helsinki: Helsinki Media Erikoislehdet. 471 s. ISBN 951-832-061-6.

Anttila, J. & Jussila, K., 2016. Mitä laatu on? [verkkodokumentti]. Suomen Standardisoimisliito SFS ry. Saatavissa https://www.sfs.fi/ajankohtaista/uutiskirjeet/uutiskirjeet_2016/mita_laatu_on_artikkeli [viitattu 13.8.2018].

Asp, R., Tuominen, T. & Hyppönen, H., 2018. 1.1 Mitä on kunnossapito? [verkkodokumentti]. Opetushallitus, Kunnossapitoyhdistys, EDU.fi. Saatavissa http://www03.edu.fi/oppimateriaalit/kunnossapito/perusteet_1-1_mita_on_kunnossapito.html [viitattu 13.8.2018].

Baker, D. & Maidukov, S., 2015. Turn paper documents into searchable PDFs [verkkodokumentti]. Adobe, tutorials. Saatavissa https://helpx.adobe.com/acrobat/how-to/scan-paper-documents-searchable-pdf.html [viitattu 13.8.2018].

Bartee, T. C., 1992. Data Communications, Networks, and Systems. 2. painos. United States of America: SAMS. 442 s. ISBN 0-672-22790-8.

Ben-Ari, M., 1982. Principles of Concurrent Programming. United States of America: Prentice-Hall International, Inc. 172 s. ISBN 0-13-701078-8.

Berry, M. & Csizmadia, A., 2016. Computational thinking and mathematical reasoning [verkkodokumentti]. Milesberry.net. Saatavissa http://milesberry.net/2016/11/computational-thinking-and-mathematical-reasoning [viitattu 13.8.2018].

Bhunia, C. T., 2006. Information Technology Network And Internet. Intia: New Delhi, New Age International 466 s. ISBN 81-224-1662-4.

Bouras, A. S., 2018. What is the Difference Between Syntax Errors and Logic Errors? [verkkodokumentti]. Bouraspage.com. Saatavissa https://www.bouraspage.com/repository/algorithmic-thinking/what-is-the-difference-between-syntax-errors-and-logic-errors [viitattu 17.8.2018].

Braun, P. J., Sipos, M., Ekler, P. & Fitzek, F. H. P., 2017. On the performance boost for peer to perr WebRTC-based video streaming with network coding [verkkodokumentti]. 2017 IEEE International Conference on Communications (ICC). Saatavissa https://ieeexplore.ieee.org/document/7996613/ [viitattu 17.8.2018].

Buechler, C. W. & Pingle, J., 2009. pfSense: The Definitive Guide - The Definitive Guide to the pfSense Open Source Firewall and Router Distribution. United States of Americ: Reed Media Services. 489 s. ISBN 978-0-9790342-8-2.

Chandrayan, P., 2018. BlockChain Princible, Type & Application & Why You Should Care About It [verkkodokumentti]. Medium.com: The Startup. Saatavissa https://medium.com/swlh/blockchain-principle-type-application-why-you-should-care-about-it-8c8a39113c7d [viitattu 13.8.2018].

Chong, S., 2018. Lambda calculus [verkkodokumentti]. Harvard School of Engineering and Applied Sciences — CS 152: Programming Languages, Lecture 7, Tuesday, February 13, 2018. Saatavissa https://www.seas.harvard.edu/courses/cs152/2018sp/lectures/lec07-lambdacalc.pdf [viitattu 13.8.2018].

Cockcroft, A., 2017. Adrian Cockcroft Presents: Shrinking Microservices to Functions [verkkodokumentti]. Inside HPC. Saatavissa https://insidehpc.com/2017/02/adrian-cockcroft-presents-shrinking-microservices-functions [viitattu 13.8.2018].

Comer, D. E., 2009. Computer Networks and Internets. 5. painos. United States of America: Pearson Education International, 600 s. ISBN 978-0-13-504583-1.

Coulouris, G., Dollimore, J. & Kindberg, T., 2005. Distributed Systems – Concepts and Design. 4. painos. United States of America: Addison-Wesley, 927 s. ISBN 0-321-26354-5.

Dear, R., 2014. Difficulties in calculating MTBF and reliability [verkkodokumentti]. Embedded Computing Design. Saatavissa http://www.embedded-computing.com/embedded-computing-design/difficulties-in-calculating-mtbf-and-reliability [viitattu 16.8.2018].

EUR-Lex, 2016. EUROOPAN PARLAMENTIN JA NEUVOSTON ASETUS (EU) 2016/679, annettu 27 päivänä huhtikuuta 2016, luonnollisten henkilöiden suojelusta henkilötietojen käsittelyssä sekä näiden tietojen vapaasta liikkuvuudesta ja direktiivin 95/46/EY kumoamisesta (yleinen tietosuoja-asetus) [verkkodokumentti]. Euroopan unionin virallinen lehti, EUR-Lex. Saatavissa https://eur-lex.europa.eu/legal-content/FI/TXT/PDF/?uri=CELEX:32016R0679&from=FI [viitattu 13.8.2018].

Feldman, J., 1999. Verkonhallinta Trainer. Jyväskylä: Gummerus Kirjapaino Oy, IT Press, 430 s. ISBN 951-826-060-5.

FIPS, 2002. Security Requirements for Cryptographic Modules. [verkkodokumentti]. nist.gocv. Saatavissa https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf [viitattu 13.8.2018].

Garriga, M., 2017. Towards a Taxonomy of Microservices Architectures [verkkodokumentti]. Italy: Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano. Saatavissa http://fmse.di.unimi.it/faacs2017/papers/ paperMSE3.pdf [viitattu 13.8.2018].

Geraud, R., 2017. Zero-Knowledge Proof: More secure than passwords? [verkkodokumentti]. Ingenico Group, Ingenico Experts /Spaping the future pf payment. Saatvissa https://blog.ingenico.com/posts/2017/07/zero-knowledge-proof-more-secure-than-passwords.html [viitattu 16.8.2018].

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y., 2014. Generative Adversarial Nets [verkkodokumentti]. Nips.cc. Saatavissa http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf [viitattu 19.8.2018].

Gray J. N., 1978. Notes on data base operating systems. In R. Bayer, R.M. Graham, and G. Seegmüller, editors, Operating Systems, volume 60 of Lecture Notes in Computer Science, pages 393{481. Springer Berlin Heidelberg.

Grindal, M., 2007. Handling Combinatorial Explosion in Software Testing [verkkodokumentti]. Linköping Studies in Science and Technology, Dissertation No. 1073. Saatavissa https://pdfs.semanticscholar.org/47e7/d56bf87fc5d1455f7fbb29e782b4814a87da.pdf [viitattu 13.8.2018].

Gupta, M., 2017. Blockchain For Dummies, IBM Limited Edition. United States of America: John Wiley & Sons, Inc. 41 s. ISBN 978-1-119-37139-7.

Haataja, J., 2000. Ohjeita Matlabin käyttöön [verkkodokumentti]. csc.fi. Saatavissa https://extras.csc.fi/math/matlab/matlabohje.pdf [viitattu 13.8.2018].

Halevi, S., 2018. Elegant, Disgusting Cryptography [verkkodokumentti]. IBM. Saatavissa https://www.ibm.com/blogs/research/2018/03/elegant-disgusting-cryptography [viitattu 16.8.2018].

Halminen, L., 2018. "Tietoturva ei ole kovin palkitsevaa" - Teknologia: Tietoturva-asiantuntijan mukaan hakkereiden hyökkäyksillä voi olla "todella vakavia seurauksia". Helsingin Sanomat, N:o 149, 4.6.2018, s. A 20.

Held, G., 1993. Top Secret Data Encryption Techniques. United States of America: SAMS Publishing, 218 s. ISBN 0-672-30293-4.

Hietanen, P., 2001. C++ ja olio-ohjelmointi. 7. painos. Porvoo: Docendo Finland Oy, 830 s. ISBN 951-846-000-0.

Hinnant, H. E., 2016. endian, Just endian [verkkodokumentti]. open-std.org. Saatavissa http://open-std.org/JTC1/SC22/WG21/docs/papers/2016/p0463r0.html [viitattu 13.8.2018].

Hughes, J., 1990. Why Functional Programming Matters. Teoksessa: Turner, D. A. Research Topics in Functional Programming. USA: Addison-Wesley, s. 17–42. ISBN 0-201-172236-4.

Hunt, A. & Thomas, D., 2011. The Prgmatic Programmer - from journeyman to master. 26. painos. United States of America: Addison-Wesley. 321 s. ISBN 0-201-61622-X.

Huovinen, A., 2017. Onko esineiden internet oikeasti palveluiden internet? [verkkodokumentti]. VTT. Saatavissa https://vttblog.com/tag/esineiden-internet [viitattu 13.8.2018].

Hurula, K. & Kauppi, T., 2015. Uusiutuva teräsrakentaminen 2014 - Lapin alueen teräsrakentaminen - haasteet, osaamispääoma ja kehittämistarpeet. Rovaniemi: Lapin AMK:n julkaisuja sarja B. Raportit ja selvitykset 15/2015. 82 s. ISBN 978-952-316-095-8.

Hämäläinena, P., 2016. Mikropalvelut nousivat hypen huipulle - mitä hyötyä niistä on? [verkkodokumentti]. Tivi-lehti. Saatavissa https://www.tivi.fi/Kaikki_uutiset/mikropalvelut-nousivat-hypen-huipulle-mita-hyotya-niista-on-6534379 [viitattu 13.8.2018].

ICMP, 1981. Internet Control Message Protocol - DARPA Internet Program Protocol Specification [verkkodokumentti]. Network Working Group, RFC 792. Saatavissa https://tools.ietf.org/html/rfc792 [viitattu 17.8.2018].

Järvinen, P. & Kerola, P., 1978. Systemointi I - Käytäntö tietosysteemin rakentamisessa. Vaasa: Oy Gaudeamus Ab. 165 s. ISBN 951-662-239-9.

Kaarnalehto, M., 2011. Salausmenetelmät: Symmetrinen, epäsymmetrinen ja tiivistealgoritmit [verkkodokumentti]. Laurea ammattikorkeakoulu. Saatavissa https://www.theseus.fi/bitstream/handle/10024/33848/Kaarnalehto_Mika.pdf?sequence=1 [viitattu] 13.8.2018].

Kaksonen, A., 2018. SmartSteel 1.0 - ensimmäinen askel kohti materiaalien internettiä [verkkodokumentti]. SSAB AB, Sweden. Saatavissa https://www.ssab.fi/ssab-konserni/uutishuone/ uutisarkisto/2018/05/23/07/00/smartsteel-10--ensimminen-askel-kohti-materiaalien-interneti [viitattu 13.8.2018].

Karsikas, N., 2013. Colour marking metal surfaces. Patent, classifications B23K26/362 Laser etching, WO2014170551A1, Assignee Cajo Technologies Oy 2013-04-18.

Karvi, T., 2012. Tiivistefunktioilta vaadittavat ominaisuudet [verkkodokumentti]. Helsingin yliopisto. Saatavissa https://www.cs.helsinki.fi/u/karvi/perusteet-luku2-bea_12.pdf [viitattu 13.8.2018].

Kauppakamari, 2011. Yleisohje kirjanpidon menetelmistä ja aineistoista 01.02.2011 [verkkodokumentti]. Helsingin seudun kauppakamari. Saatavissa https://kauppakamaritieto.fi/fi/s/t/kirjanpito/kirjanpitolautakunnan-yleisohjeet-ja-lausunnot/yleisohjeet/yleisohje-kirjanpidon-menetelmista-ja-aineistoista-01022011/ [viitattu 16.8.2018].

Kauppi, T., 2013. Korkeiden lämpötilojen teräkset - kirjallisuustutkimus. Kemi: Kemi-Tornion ammattikorkeakoulu, sarja B. Raportit ja selvitykset 12/ 2013. 67 s. ISBN 978-952-5897-70-8.

Kauppi, T. & Toppila, R., 2018. Digitalisaaation hyödyntäminen teräsrakenteiden vaatimustenmukaisuuden osoittamisessa. Ohutlevylehti 1/2018. s. 17 - 20.

Kemppi, K., 2017. Weld-Eye - The Internet of Welding: How IoT is transforming quality and safety [verkkodokumentti]. 2nd Annual Internet of Manufacturing Conference. Saatavissa https://internetofbusiness.com/wp-content/uploads/2017/02/Kari-Kemppi-14.35-15.05-Stream-A-Day-2small.pdf [viitattu 10.8.2018].

Kirjanpitolautakunta, 2011. Kirjanpitolautakunnan yleisohje kirjanpidon menetelmistä ja aineistoista 1.2.2011 [verkkodokumentti]. Edilex.fi. Saatavissa https://www.edilex.fi/kilaohje/kirjanpito [viitattu 7.8.2018].

Kirk, J., 2018. Leak of 23,000 Private Keys Triggers Security Scramble - Digital Certificate Revocation Blame Game: Trustico Swaps Blows With DigiCert

[verkkodokumentti]. bankinfosecurity.com. Saatavissa https://www.bankinfosecurity.com/private-keys-for-23000-digital-certificates-leaked-a-10689 [viitattu 7.8.2018].

Kitzelmann, E., 2009. Analytical Inductive Functional Programming [verkkodokumentti]. In Logic-Based Program Synthesis and Transformation, 18th International Symposium, LOPSTR 2008, Revised Selected Papers, Volume 5438 of LNCS, pages 87–102. Springer-Verlag. Saatavissa http://emanuel.kitzelmann.org/documents/publications/Kitzelmann2009.pdf [viitattu 9.8.2018].

Knuth, D. E., 1974. Structured Programming with go to Statements. United States of America: ACM Computing Surveys (CSUR) Volume 6 Issue 4, Dec. 1974. 261 - 301 s.

Knuth, D. E., 2000. The Art of Computer Programming – Volume 3 / Sorting and Searching. 5. painos. United States of America: Addison-Wesley, 780 s. ISBN 0-201-89685-0.

Kohonen, T., 1987. Neural networks for speech recognition. Otaniemi: Helsinki University of Technology 1987. ISBN 951-754-298-4.

Korkishko, I., 2018. Pros and Cons of Functional Programming [verkkodokumentti]. Syndicode.com. Saatavissa https://syndicode.com/2018/06/07/pros-and-cons-of-functional-programming [viitattu 7.8.2018].

Koski, M., 2015. Kadonneen provenienssin metsästys - Provenienssien jäsentyminen arkistokuvailun kansallisessa käsitemallissa [verkkodokumentti]. Tampereen yliopisto 61 s. Saatavissa https://tampub.uta.fi/bitstream/handle/10024/97506/GRADU-1435212873. pdf?sequence=1 [viitattu 13.8.2018].

Koski, O. & Husso, K., 2018. Tekoälyajan työ - Neljä näkökulmaa talouteen, työllisyyteen, osaamiseen ja etiikkaan. Helsinki: Työ- ja elinkeinoministeriön julkaisuja 19/2018. 55 s. ISBN 978-952-327-311-5.

Kripke, S. A., 2013. The Church-Turing "Thesis" as a Special Corollary of Gödel's Completeness Theorem. s. 77 - 104 Teoksessa Copeland, B., J., Posy, C., J. & Shagrir,

O. (toim.) 2013. Computability: Turing, Gödel, Church, and beyond. United States of Americ: The MIT Press. 362 s. ISBN 9781461931843, 9780262312684.

Kyrnin, M., 2018. Using Graphics Cards for More Than Just 3D Graphics - How the Graphics Processor Is Turning Into a General Processor [verkkodokumentti]. Lifewire. Saatavissa https://www.lifewire.com/graphics-cards-3d-graphics-834089 [viitattu 13.8.2018].

Labrosse, J. J., 2002a. MicroC/OS-II - The Real-Time Kernel. 2. painos. United States of America: CMP Books, 606 s. ISBN: 1-57820-103-9.

Labrosse, J. J., 2002b. Complete and Ready-to-Use Modules in C - Embedded Systems Building Blocks. 2. painos. United States of America: CMP Books, 611 s. ISBN: 0-87930-604-1.

Lahti, V., 2016. Lohkoketju muuttaa maailmaa [verkkodokumentti]. Sitra.fi. Saatavissa https://www.sitra.fi/blogit/lohkoketju-muuttaa-maailmaa [viitattu 3.8.2018].

Lampila, A., 2018. Erävoitto USA:n senaatissa! [verkkodokumentti]. Uusi suomi. Saatavissa http://artolampila.puheenvuoro.uusisuomi.fi/255453-eravoitto-usan-senaatissa [viitattu 17.8.2018].

Lehtomäki, M., 2016. Uusi kirjanpitolaki - tiivistelmä olennaisimmista muutoksista [verkkodokumentti]. Taloushallintoalan tradenomit ry. Saatavissa https://www.taloushallintoalantradenomit.fi/uutiset.html?36890 [viitattu 16.8.2018].

Leino, T., 2006. Staattisesti kuormitettujen hitsausliitosten suunnittelu - laajarunkoisten liikuntahallien rakenteellinen turvallisuus [verkkodokumentti]. VTT, tutkimusraportti, versio 19.7.2006. 74 s. Saatavissa https://www.vtt.fi/inf/julkaisut/muut/2006/HitsLiitSuunn_19-7-2006.pdf [viitattu 11.8.2018].

Lepola, P. & Makkonen, M., 2009. Hitaustekniikat ja teräsrakenteet. 4. painos. Helsinki: WSOYpro Oy, 429 s. ISBN 978-951-0-27158-2.

Lwakatare, L. E., 2017. DevOps adoption and implementation in software development practice. - Concept, practices, benefits and challenges. Oulu: Acta Universitas Ouluensis, A Scientiae Rerum Naturalium 702. 110 s. ISBN 1796-220X (Online).

Lynch, G. S., 2015. Single Point of Failure - The Ten Essential Laws of Supply Chain Risk Management. United States of America: Wiley. 320 s. ISBN 978-1-119-19835-2.

Martikainen, A., 2013. Lean, Six Sigma ja Total Welding Management hitsaavassa verkostossa. Lappeenrannan teknillinen yliopisto, Teknillinen tiedekunta, Konetekniikan koulutusohjelma, diplomityö. 99 s.

Michaelson, G., 2011. An Introduction to Functional Programming Through Lambda Calculus. United States of America: Heriot-Watt University, Dover Publications, Inc. Mineola, New York. 320 s. ISBN 978-0-4686-47883-8.

Mills, D., Delaware, U., Martin, J., Burbank, J. & Kasch, W., 2010. Network Time Protocol Version4: Protocol and Algorithms Specification [verkkodokumentti]. Internet Engineering Task Force (IETF). Saatavissa https://tools.ietf.org/html/rfc5905 [viitattu 17.8.2018].

Morris, J., 2012. Practical Data Migration. 2. painos. United Kingdom: BCS Learning & Development Ltd, 248 s. ISBN 978-1-906124-84-7.

Myers, A., 2011. Stanford's John McCarthy, seminal figure of artificial intelligence, dies at 84 [verkkodokumentti]. Stanford News. Saatavissa https://news.stanford.edu/news/2011/october/john-mccarthy-obit-102511.html [viitattu 13.8.2018].

Netflix OSS, 2016. Netflix Open Source Software Center [verkkodokumentti]. Saatavissa https://netflix.github.io/ [viitattu 7.8.2018].

Niiniluoto, I., 1983. Tieteellinen päättely ja selittäminen. Helsinki: Otava, 416 s. ISBN 951-1-73379-6.

Obermaier, F., Obermayer, B., Wormer, V. & Jaschensky, W., 2016. About the Panama Papers [verkkodokumentti]. Süddeutsche Zeitung, Panama Papers, The Secrets of dirty

money. Saatavissa https://panamapapers.sueddeutsche.de/articles/56febff0a1bb8d3c3495adf4/ [viitattu 18.8.2018].

Ojansuu, V., 2018. Asiakasarvotyökalu: koneoppimisen luoman asiakasarvon määrittäminen [verkkodokumentti]. Tampereen teknillinen yliopisto, diplomityö. 78 s. Saatavissa https://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/25690/ Ojansuu.pdf?sequence=1&isAllowed=y [viitattu 16.8.2018].

Oram, A. & Wilson, G., 2007. Beautiful Code - Leading Programmers Explain How They think. United States of America: O'Reilly Media Inc. 593 s. ISBN 978-0-596-51004-6.

Paavola, M., 2018. Murtosokasta SPoF-ilmiöön [verkkodokumentti]. Oulun yliopisto, teknillinen tiedekunta, konetekniikka, kandidaatintyö. 26 s. Saatavissa http://jultika.oulu. fi/Record/nbnfioulu-201802201246 [viitattu 13.8.2018].

Parker, D., 2008. Analyzing a Hack from A to Z (Part 2) [verkkodokumentti]. TechGenix. Saatavissa http://techgenix.com/Analyzing-Hack-Part2/ [viitattu 17.8.2018].

Partanen, A., 2003. Olio-ohjelmien testaus [verkkodokumentti]. Kuopion yliopisto: Informaatioteknologian ja kauppatieteiden tiedekunta, tietojenkäsittelytieteen koulutusohjelma, pro gradu -tutkielma. 70 s. Saatavissa http://www.cs.uef.fi/tutkimus/Teho/apartanegradu.pdf [viitattu 13.8.2018].

Patterson, D. A. & Hennessy J. L., 2005. Computer Organization and Design - The Hardware / Software Interface. United States of America: Morgan Kaufmann Publishers. 3 painos. 621 s. ISBN 1-55860-604-1.

PCSpecialist, 2018. Fractal Tower PC Configuration - Asus H310M-A, Intel i7-8700 Six Core, 32 GB DDR4, 2 x 12 TB Seagate 3.5", 1 TB M.2 PCIe NVMe, 6 GB NVIDIA GTX 1060, Asus XG-C100C, Corsair H115i Liquid Cooler [verkkodokumentti]. www.pcspecialist.co.uk. Saatavissa https://www.pcspecialist.co.uk/configuration/quote [viitattu 28.8.2018].

Pervilä, M., 2018. It-väen kannatta hypätä ajoissa mukaan iot-talkoisiin [verkkodokumentti]. TIVI, CIO, IT ja esineiden internet. Saatavissa https://www.tivi.fi/CIO/it-vaen-kannattaa-hypata-ajoissa-mukaan-iot-talkoisiin-6736501 [viitattu 16.8.2018].

Petzold, C., 1999. CODE - The Hidden Language of Computer Hardware and Software. United States of America: Microsoft Press. 393 s. ISBN 0-7356-0505-X.

Pierce, B. C., 1991. Basic Category Theory for Computer Scientists. United States of America: The MIT Press, 100 s. ISBN 978-0-262-66071-6.

Pierce, B. C., 2002. Types and Programming Languages. United States of America: The MIT Press, 623 s. ISBN 978-0-262-16209-8.

Pittman, K., 2017. The IIot in a Nutshell [verkkodokumentti]. Engineering.com, Manufacturing. Saatavissa https://www.engineering.com/AdvancedManufacturing/ArticleID/15282/The-IIoT-in-a-Nutshell.aspx [viitattu 16.8.2018].

Postel, J., 1980. User Datagram Protocol [verkkodokumentti]. Network Working Group, RFC 768. Saatavissa https://tools.ietf.org/html/rfc768 [viitattu 17.8.2018].

Pressman, R. S., 2005. Software Engineering – A Practitioner's Approach. 6. painos. New York: McGraw-Hill, 912 s. ISBN 007-123840-9.

Prowell, S. J., Trammel, C. J., Linger R. C. & Poore J. H., 1999. Cleanroom Software Engineering - Technology and Process. United States of America: Addison-Wesley, 390 s. ISBN 0-201-85480-5.

Rana, S., 2018. IPC though shared memory [verkkodokumentti]. GeeksforGeeks. A computer science portal for geeks. Saatavissa https://www.geeksforgeeks.org/ipc-shared-memory/ [viitattu 17.8.2018].

Rasheed, S., 2018. Why is the uptime of a website so important? [verkkodokumentti]. Quora.com. Saatavissa https://www.quora.com/Why-is-the-uptime-of-a-website-so-important [viitattu 16.8.2018].

Riehl, D. A., 2001. Peer-to-peer Distribution Systems: Will Napster, Gnutella, and Freenet Create a copyright Nirvana or Gehenna? [verkkodokumentti]. Mitchell Hamline School of Law. Saatavissa https://open.mitchellhamline.edu/cgi/viewcontent.cgi?article=1796&context=wmlr [viitattu 17.8.2018].

Rotem-Gal-Oz, A., 2014. Services, Microservices, Nanoservices – oh my! [verkkodokumentti]. Cirrus Minor. Saatavissa http://arnon.me/2014/03/services-microservices-nanoservices/ [viitattu 13.8.2018].

Rouse, M., 2014. PKI (public key infrastructure) [verkkodokumentti]. Techtarget.com. Saatavissa https://searchsecurity.techtarget.com/definition/PKI [viitattu 13.8.2018].

Rouse, M., 2016. Dependency hell - Definition [verkkodokumentti]. Techtarget.com. Saatavissa https://searchitoperations.techtarget.com/definition/dependency-hell [viitattu 9.8.2018].

Rudo, P., 2011. The Biggest Disk Defragmentation Mynths [verkkodokumentti]. Enterprise Features. Saatavissa http://www.enterprisefeatures.com/the-biggest-disk-defragmentation-myths/ [viitattu 13.8.2018].

Ryabitsev, K., 2014. PGP Web of Trust: Core Concepts Behind Trusted Communication [verkkodokumentti]. Linux.com: News for the Open Source Professional. Saatavissa https://www.linux.com/learn/pgp-web-trust-core-concepts-behind-trusted-communication [viitattu 16.8.2018].

Salminen, A., 2005. Metatiedot organisaatioiden sisällönhallinnassa [verkkodokumentti]. Jyväskylän yliopisto, Lainsäädäntöprosessin tiedonhallinnan kehittäminen, metatiedot suomalaisen lainsäädäntöprosessin tiedonhallinnassa, seminaari Eduskunnassa 2.6.2005. Saatavissa http://www.it.jyu.fi/raske/seminaari2005/salminen_1_20050602.pdf [viitattu 13.8.2018].

Salonen, J., Halunen, K., Korhonen, H., Lähteenmäki, J., Pussinen, P., Vallivaara, V., Väisänen, T. & Ylén, P., 2018. Lohkoketjuteknologian mahdollisuudet ja hyödyt sosiaali- ja terveydenhuollossa [verkkodokumentti]. Valtioneuvoston selvitys- ja tutkimustoiminta. Saatavissa https://tietokayttoon.fi/documents/10616/3866814/

80_Lohkoketju teknologian_mahdollisuudet_final.pdf/3dd2c141-e8b3-4825-8637-76bf7206704b? version=1.0 [viitattu 7.8.2018].

Scambray, J., McClure, S. & Kurtz, G., 2001. Hacking Exposed: Network Security Secrets & Solutions. 2. Painos. United States of America: Osborne / McGraw-Hill. 703 s. ISBN 0-07-212748-1.

Schneier, B., 1998. Security Pitfalls in Cryptography [verkkodokumentti]. *Information Management & Computer 1998.* Saatavissa https://www.schneier.com/essays/archives/1998/01/security_pitfalls_in.html [viitattu 7.8.2018].

Schneier, B., 2003. Beyond Fear - Thinking Sensibly About Security in an Uncertain World. United States of America: Springer-Verlag, 295 s. ISBN 0-387-02620-7.

Schumann, T., 2018. Consensus Mechanisms Explained: PoW vs PoS [verkkodokumentti]. hackernoon.com. Saatavissa https://hackernoon.com/consensus-mechanisms-explained-pow-vs-pos-89951c66ae10 [viitattu 16.8.2018].

Siberschatz, A. & Galvin, P. B., 1998. Operating System Concepts. 5. painos. United States of America: Addison-Wesley, 888 s. ISBN 0-201-54262-5.

Skiftesvik, K., 2018. [Request ID :##RE-811158##] : Jultika-tietokannan koko? [yksityinen sähköpostiviesti]. Vastaanottaja: Mikko Paavola. Lähetetty 16.8.2018 klo 9.31 (GMT +0300).

Slater, N., 2015. Using Containers to Build a Microservices Architecture [verkkodokumentti]. Medium.com: AWS Startup Collection. Saatavissa https://medium.com/aws-activate-startup-blog/using-containers-to-build-a-microservices-architecture-6e1b8bacb7d1 [viitattu 13.8.2018].

Stallings, W., 2006. Cryptography and Network Security - Principles and Practices. 4. painos. United States of America: Pearson Education International, 680 s. ISBN 0-13-202322-9.

Stavely, A. M., 1999. Toward Zero-Defect Programming. United States of America: Addison-Wesley, 240 s. ISBN 0-201-38595-3.

Stetson, S., 2017. Microservices - Reference Architecture [verkkodokumentti]. NGINX Inc. 53 s. Saatavissa https://www.nginx.com/people/chris-stetson [viitattu 13.8.2018].

Syrjä, J., 2016. Mikropalveluarkkitehtuurit. Vaasa: Vaasan ammattikorkeakoulu, tietotekniikka. 56 s.

Södervall, A., 2014. Teräsrakentamisen standardi SFS-EN 1090 - Ammattiopisto Lappian Kemin oppimisympäristöön [verkkodokumentti]. Lapin AMK. 53 s. Saatavissa https://www.theseus.fi/bitstream/handle/10024/83043/Sodervall_Arto.pdf.pdf?sequence=1 [viitattu 12.8.2018].

Tanenbaum, A. S., 2017. An Open Letter to Intel - addressed to Brian Matthew Krzanich CEO of Intel [verkkodokumentti]. Saatavissa https://www.cs.vu.nl/~ast/intel/ [viitattu 7.8.2018].

Technet, 2017. Avoiding Single Points of Failure [verkkodokumentti]. USA: Technet Microsoft. Saatavissa https://technet.microsoft.com/en-us/library/cc938489.aspx [viitattu 7.8.2018].

Tesler, L., 1970. CV: Adages & Coinages [verkkodokumentti]. nomodes.com. Saatavissa http://www.nomodes.com/Larry_Tesler_Consulting/Adages_and_Coinages.html [viitattu 13.8.2018].

The Unicode Standard, 2018. The Unicode® Standard - Version 11.0 - Core Specification. United States of America: The Unicode Consortium. 976 s. ISBN 978-1-936213-19-1.

Tseng, M., Edmunds, T. & Canaran, L., 2018. Introduction to Edge Computing in IIoT - An Industrial Internet Consortium White Paper [verkkodokumentti]. Industrial Internet Consortium. Saatavissa https://www.iiconsortium.org/pdf/Introduction_to_Edge_Computing_in_IIoT_2018-06-18.pdf [viitattu 19.8.2018].

Turner, D., 2006. Church's Thesis and Functional Programming s. 518 - 544. Teoksessa: Adam Olszewski, Jan Wole`nski, Robert Janusz. (toim.) Church's Thesis After 70 Years. Germany: Ontos Verlag, ontos mathematical logic 2006. s. 514 -554. ISBN 3-938793-09-0.

Unicode, 2018. Unicode® 11.0.0 [verkkodokumentti]. The Unode Consortium. Saatavissa http://www.unicode.org/versions/Unicode11.0.0/ [viitattu 12.8.2018].

Verronen, P., Kaartinen, H. & Nokela, S., 2016. Tulevaisuuden Internet of Things (IoT) mittausympäristöt. Kokkola: Centria-ammattikorkeakoulu. Raportteja ja selvityksiä, 6. 36 s. ISBN 978-952-7173-00-8.

Väestörekisterikeskus, 2018. Saamelaisnimien oikeinkirjoitus vaatii muutoksia useisiin viranomaisten tietojärjestelmiin [verkkodokumentti]. Väestörekisterikeskus. Saatavissa https://vrk.fi/artikkeli_e/-/asset_publisher/saamelaisnimien-oikeinkirjoitus-vaatii-muutoksia-useisiin-viranomaisten-tietojarjestelmiin [viitattu 7.8.2018].

WebAssembly, 2017. WebAsssembly [verkkodokumentti]. webassembly.org. Saatavissa https://webassembly.org [viitattu 13.8.2018].

Wolter, J., 2005. Unix Incompatibility Notes: Byte Order [verkkodokumentti]. unixpapa.com. Saatavissa https://unixpapa.com/incnote/byteorder.html [viitattu 7.8.2018].