TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

TUOMAS KAITTOLA

WEB-TEKNOLOGIOILLA TOTEUTETUN DATATUOTTEEN ARKKITEHTUURI

Diplomityö

Tarkastajat: Prof. Petri Ihantola ja TkT Jukka Huhtamäki

Tarkastajat ja aihe hyväksytty 9. elokuuta 2017

TIIVISTELMÄ

TUOMAS KAITTOLA: Web-teknologioilla toteutetun datatuotteen arkkitehtuuri
Tampereen teknillinen yliopisto
Diplomityö, 52 sivua
Joulukuu 2017
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastajat: Prof. Petri Ihantola ja TkT Jukka Huhtamäki

Avainsanat: datatuote, ohjelmistoarkkitehtuurit, verkostoanalyysi, visuaalinen analytiikka, visuaalinen verkostoanalyysi, web-teknologiat

Big data, suuret datamäärät ja niihin liittyvä analytiikka ovat olleet viime vuosina tietotekniikan alan suurimpia trendejä. Kaikki haluavat kerätä dataa, mutta harva osaa hyödyntää sen potentiaalia. Pilvipalvelut ja viime aikoina huimasti kehittyneet web-sovellukset ovat johtaneet käyttöliittymien siirtymiseen web-selaimiin, ja suurten datamäärien käsittely on lähes pakko tehdä riittävästi kapasiteettia tarjoavissa pilvipalveluissa. Tästä syystä data-analytiikan sovellukset ovat siirtyneet luonnostaan internetiin.

Suuria datamääriä niiden käsittelyyn sekä analytiikkaan yhdistäviä palveluita voidaan kutsua esimerkiksi *datatuotteiksi*, mutta käsitteistö ei ole vakiintunutta ja aihepiiriin liittyvää tutkimusta on tehty erittäin rajallisesti. Tässä työssä selkeytetään tilannetta ja esitetään määritelmä "datatuote"-käsitteelle. Kirjallisuuskatsauksen avulla todettiin datatuotteen olevan tietotekninen järjestelmä, joka käsittelee ja jalostaa syötteenään saamaansa dataa vuorovaikutteisesti ja tuottaa näin lisäarvoa loppukäyttäjälleen.

Tämän lisäksi työssä tutkittiin ja ehdotettiin web-pohjaiselle datatuotteelle soveltuvaa arkkitehtuurimallia sekä toteutettiin ehdotuksen mukaisen datatuotteen prototyyppi Tampereen teknillisellä yliopistolla käynnissä olevan *Cobweb*-tutkimusprojektin tarpeisiin. Tämä osuus toteutettiin *Action Design Research*-tutkimusmenetelmän mukaisesti. Datatuotteiden todettiin käsittelevän lähdedataansa tyypillisesti usean toisistaan erillisen vaiheen kautta. Tähän pohjautuen web-teknologioilla toteutetulle datatuotteelle ehdotettiin microservices-arkkitehtuurimallia, jossa datan käsittelyn yksittäiset vaiheet on toteutettu omissa palveluissaan.

ABSTRACT

TUOMAS KAITTOLA: Architecture of a Web-Based Data Product
Tampere University of Technology
Master of Science thesis, 52 pages
December 2017
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiners: Prof. Petri Ihantola and Doc. Jukka Huhtamäki

Keywords: Data Product, Software Architectures, Network Analysis, Visual Analytics, Visual Network Analytics, Web Technologies

Big data, large amounts of data and analytics related to those have been one of the biggest trends in the information technology industry for the past few years. Everyone wants to collect data, but only a few is able to exploit the potential which lies in that data. Cloud services and web applications, which have been greatly evolved lately, have enabled current trend where user interfaces are moving from desktop to the browsers, and crunching of big amounts of data practically has to be done in the cloud. This has organically moved data analytics and related applications to the Internet.

Services that combine big amounts of data to interactive analytics can be called as *data products*, as an example, but the terminology is not yet fully established and amount of research around those is still quite limited. In this work we want to make the terminology more clear and find a definition for the "data product" term. By doing literature review we found that data products are information technology systems which process and refine their input data and combine it with interactive analytics and therefore produce value to their users.

Among that, we wanted to compare architectural patterns, which could be used with web-based data products and then implement a prototype software needed in *Cobweb* research project in Tampere University of Technology. This part of the research was done according to *Action Design Research* method. Typically data products process their input data through several, loosely coupled phases. Based on that we propose microservices architectural style for web-based data products.

ALKUSANAT

Tämä diplomityö syntyi lähes kokonaisuudessaan vuoden 2017 aikana työskennellessäni osa-aikaisena Tampereen teknillisen yliopiston matematiikan laitoksella tutkimusapulaisena *Cobweb*-tutkimusprojektiin liittyen. Työn valmistumista edesauttoi itseni lisäksi useampi taho, joille haluan esittää tässä kiitokseni. Ensimmäiset kiitokset ansaitsevat työni ohjaajat Jukka Huhtamäki sekä Petri Ihantola, joilta sain arvokkaita työtä koskevia kommentteja sekä kehitysideoita. Kiitän myös kollegaani Timo Kalliomäkeä tässä työssä käytetyn LaTeX-pohjan toteuttamisesta sekä Samuli Rahkosta työn oikolukemisesta. Tämän lisäksi haluan kiittää työnantajaani Vincitiä, joka mahdollisti työn tekemisen joustavasti töiden ohessa.

Tampereella, 18. marraskuuta 2017

Tuomas Kaittola

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

API	Application Programming Interface
ADR	Action Design Research
AJAX	Asynchronous JavaScript And XML
BaaS	Backend as a Service
CRIPS-DM	CRoss Industry Standard Process for Data Mining
DOM	Document Object Model
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MVC	Model-View-Controller
MVP	Minimum Viable Product
REST	Representational State Transfer
SPA	Single Page Application
XML	Extensible Markup Language

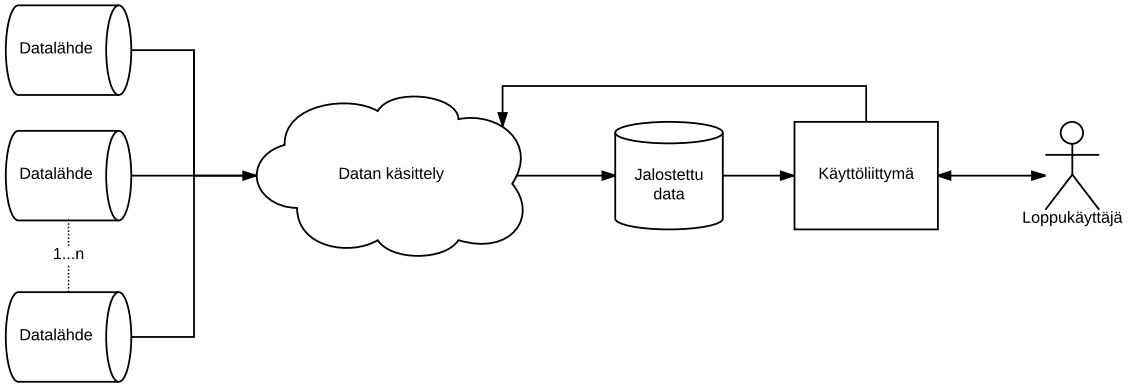
1. JOHDANTO

Big data on ja sen merkitys on ollut viime vuosina tekniikan alan kuumimpia puheenaiheita (Labrinidis & Jagadish, 2012, s. 2032). Tallennusmedioiden hintojen putoaminen on mahdollistanut valtavien datamäärien keräämisen lähes kaikkialta, josta se vain on automatisoidusti mahdollista. Datan keräämiselle ei läheskään aina ole edes olemassa olevaa tarvetta tai ennalta tiedettyä käyttötapausta, vaan motiivina toimii pelaaminen varman päälle – entä *jos* käyttötarkoitus kyseiselle datalle keksitäänkin vielä tulevaisuudessa (Tallon, 2013, s. 34).

Epäilemättä suurin osa big datasta on ainoastaan sen keränneiden yritysten hallussa, mutta myös saatavilla olevan avoimen datan määrä kasvaa jatkuvasti. Eri aihealueista kerätyssä datassa piilee valtava potentiaali, mutta datan hyödyntäminen ja analyysi on kuitenkin jäänyt monella vain pintaraapaisun tasolle (Abbott, 2001; Jagadish et al., 2014). Suurten datamäärien hahmottaminen ja käsittely on luonteeltaan tehtävä, jossa ihmiset ovat huonoja (Basole, 2009, s. 148). Siksi datamäärien räjähdysmäinen kasvu on entisestään kasvattanut tarvetta koneelliselle tiedonkäsittelylle sekä visuaaliselle analytiikalle, joka yhdistää datan pohjalta koneellisesti luodut vuorovaikutteiset visualisoinnit ihmisen tekemään analyyttiseen päättelyyn (Thomas & Cook, 2005).

Edellä kuvatusta johtuen tässä työssä keskitytään etenkin *datatuote*-käsitteeseen. Datatuote (tai englanniksi “data product”) ei ainakaan toistaiseksi ole yleisesti vakiintunut käsite. Sillä voidaan tarkoittaa ainakin tietotekniikkaa hyödyntävää järjestelmää, jonka käyttäjilleen tai tarkastelijoilleen tuoman lisäarvon voidaan katsoa perustuvan vahvasti sen tavalla tai toisella käyttämän lähdedatan *jalostamiseen*, joka tapahtuu etenkin koneellisen tiedonkäsittelyn sekä analytiikan keinoin. Kuvauksen mukaisen datatuotteen korkean tason rakenne on esitetty kuvassa 1.1. Kuvasta voidaan nähdä datatuotteen tärkeimmät ominaispiirteet: järjestelmä saa sisäänsä dataa yhdestä tai useammasta ulkoisesta lähteestä, ja ajaa datan käsittelyalgoritmin tai vastaavan läpi. Lähdedataa käsittelemällä saadaan jollain tapaa jalostettua dataa, joka esitetään loppukäyttäjälle käyttöliittymässä. Käyttöliittymä on interaktiivinen, ja loppukäyttäjä voi sen avulla esimerkiksi säätää järjestelmän parametreja siten, että hän saa nähtäväkseen useita versioita alkuperäisen lähdedatan pohjalta tuotetusta esityksestä.

Koska internet ja tietoliikenneyhteydet ovat olleet ratkaisevana tekijänä big datan



Kuva 1.1. *Datatuotteen perusrakenne*

keräämisen mahdollistajana sekä modernien järjestelmien sekä palvelujen toteutuskanavana, tässä työssä keskitytään web-pohjaisiin datatuotteisiin. Web-teknologioiden ja pilvipalveluiden hyödyntäminen on monesti myös välttämätön edellytys big datan käsittelyn mahdollistamiseksi, sillä big data on usein jo määritelmällisesti liian suurta perinteisillä menetelmillä käsiteltäväksi (Jacobs, 2009; Ward & Barker, 2013).

1.1 Tutkimusongelma ja menetelmä

Työ toteutettiin osana *Cobweb*-projektia. *Cobweb* on Tampereen teknillisellä yliopistolla vuonna 2016 alkanut ja tätä kirjoitettaessa yhä jatkuva tutkimusprojekti. Projektin päämääränä on tutkia ja kehittää laskennallisia metodeja henkilöiden älykkään törmäyttämisen sekä sosiaalisten suosittelujärjestelmien toteuttamisen mahdollistamiseksi tietotyön sekä akateemisen tutkimuksen kontekstissa. Työn puitteissa kehitettiin prototyyppejä *Cobweb*in tarpeisiin, kuten tutkijoille potentiaalisia yhteistyökumppaneita aiempien julkaisujen perusteella suositteleva *"MatchUs"*-suosittelujärjestelmä. *Cobweb*-projektia rahoittaa Suomen Akatemia.

Tässä työssä keskitytään kahteen tutkimuskysymykseen:

1. Mitä datatuotteella tarkoitetaan?
2. Millainen arkkitehtuuri sopii web-pohjaiselle datatuotteelle?

Ensimmäiseen kysymykseen vastataan aiheeseen liittyvän kirjallisuuskatsauksen kautta. Toiseen kysymykseen vastataan *Action Design Research* (ADR) -menetelmän avulla. ADR on ketterä tutkimusmenetelmä, jolla pyritään ratkaisemaan tutkimusongelma käytännössä toteutettavan tietojärjestelmän sekä toteutuksen arvioinnin avulla (Sein, Henfridsson, Purao, Rossi, & Lindgren, 2011, s. 40). Tässä tapauksessa tämä tarkoittaa, että työssä ehdotetaan teoriaan pohjautuen web-pohjaiselle datatuot-

teelle soveltuvaa arkkitehtuuria sekä testataan arkkitehtuuria toteuttamalla prototyyppi ehdotuksen mukaisesta datatuotteesta. Toteutusta arvioidaan jo työn ohessa ja muutoksia tehdään tarvittaessa, iteratiivisesti sekä ketterän ohjelmistokehityksen periaatteiden mukaisesti.

ADR-menetelmää sovellettaessa tarkoituksena ei pidä olla ainoastaan tietyn, tarkasti rajatun ongelman ratkaiseminen, vaan ennemminkin tiedon kartoitus ja kerääminen siten, että sen avulla voidaan jatkossa ratkaista myös muita samankaltaisia ongelmia. Tässä tapauksessa tämä tarkoittaa, että toteutuksessa opittuja asioita pyritään yleistämään. ADR-menetelmän vaiheet sekä periaatteet on esitetty kuvassa 1.2. ADR-menetelmän mukainen tutkimus muodostuu kuvan mukaisesti neljästä päävaiheesta: ensimmäisessä vaiheessa keskitytään ongelman määrittelemiseen, toisessa vaiheessa ratkaisun toteuttamiseen sekä arviointiin, kolmannessa vaiheessa tehdystä oppimiseen sekä pohtimiseen ja neljännessä vaiheessa opitun yleistämiseen mahdollisuuksien mukaan. Kolme ensimmäistä vaihetta tapahtuvat iteratiivisesti ja rinnakkain, samaan tapaan kuin ketterässä ohjelmistokehityksessä. (Sein et al., 2011, s. 40, s. 44)

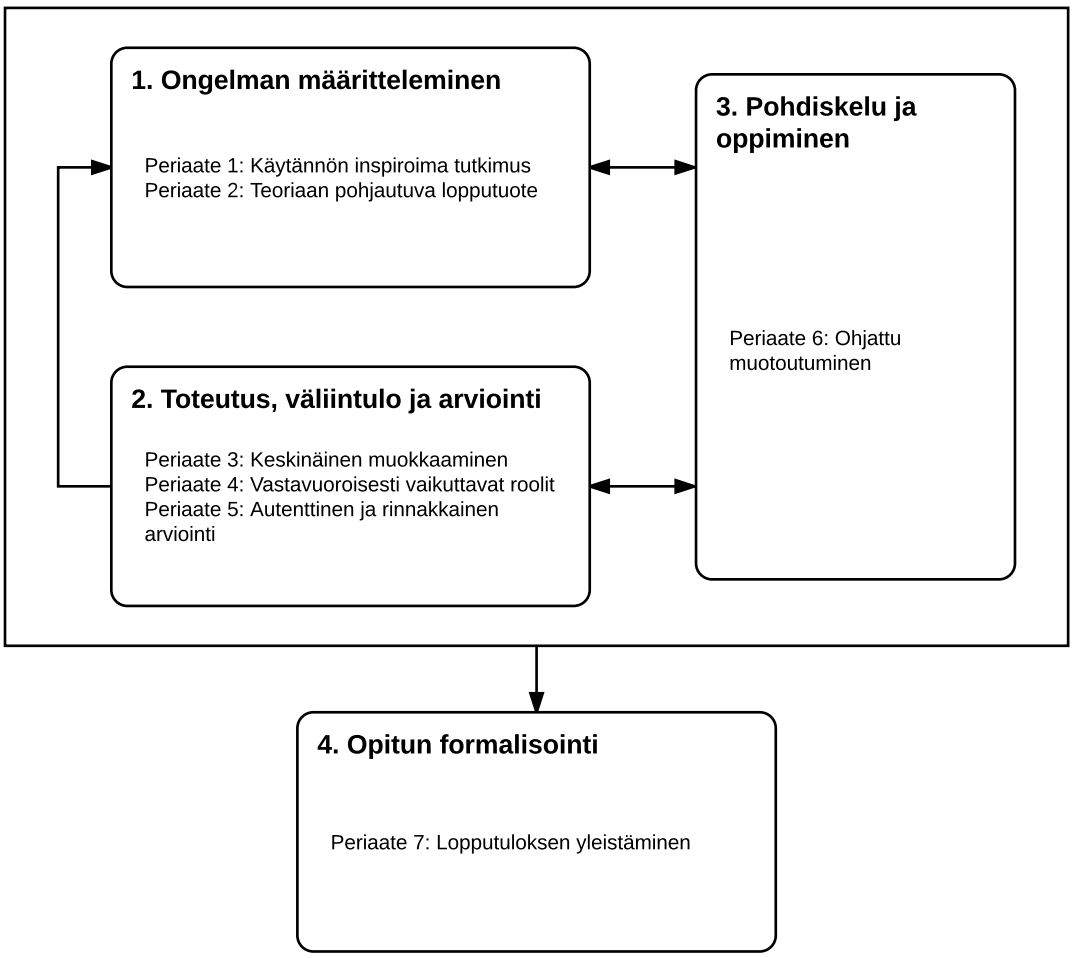
ADR-menetelmän ensimmäisen päävaiheen mukainen ongelman määritteleminen jakautuu tässä työssä kahteen osaan. Ensimmäisessä osuudessa esitetään määritelmä “datatuote”-termille ja vastataan samalla työn ensimmäiseen tutkimuskysymykseen. Kun datatuotteen määritelmä on selvillä, määritellään tutkimusongelma eli datatuotteelle soveltuvan arkkitehtuurin vaatimukset tutustumalla aiheeseen liittyvään kirjallisuuteen. Näiden pohjalta tehdään konkreettinen ehdotus datatuotteen arkkitehtuurista. Tämän pohjalta siirrytään ADR:n toisen vaiheen mukaiseen osuuteen ja toteutetaan ehdotusta vastaavan järjestelmän prototyyppi. Toteutusta arvioidaan ADR-menetelmän kolmannen vaiheen mukaisesti ja muokataan toteutusta tarvittaessa. Lopulta työssä arvioidaan prototyypin toteutuksen onnistumista arkkitehtuurin näkökulmasta sekä esitetään arvio aiemmin ehdotetun arkkitehtuurimallin soveltuvuudesta yleisempiin tapauksiin.

Kuvan 1.2 mukaisten ADR-menetelmän periaatteiden tulkitaan tarkoittavan tässä työssä seuraavaa:

Periaate 1 - käytännön inspiroima tutkimus: työ tehdään *Cobweb*-projektin tarpeisiin, ja tavoitteena on toteuttaa työn ohessa konkreettinen datatuote.

Periaate 2 - teoriaan pohjautuva lopputuote: datatuote määritellään kirjallisuuteen pohjautuen, ja sen arkkitehtuuria ehdotetaan ohjelmistoarkkitehtuurien teorian pohjalta.

Periaate 3 - keskinäinen muokkaaminen: tutkimus toteutetaan iteratiivisesti.



Kuva 1.2. *ADR-menetelmän vaiheet, pohjautuen Sein et al. (2011)*

Periaate 4 - vastavuoroisesti vaikuttavat roolit: *Cobweb*-projekti vaikuttaa toteutetun datatuotteen suunnitteluun ja ominaisuuksiin.

Periaate 5 - autenttinen ja rinnakkainen arviointi: toteutusta arvioidaan työn ohessa.

Periaate 6 - ohjattu muotoutuminen: työn aikana saadut kokemukset vaikuttavat toteutukseen.

Periaate 7 - lopputuloksen yleistäminen: lopputulosta arvioidaan myös projektikontekstin ulkopuolella.

1.2 Aiheen rajaus

Cobweb-projekti keskittyy sosiaalisia suhteita kuvaavan datan tutkimiseen, ja siksi myös tässä työssä keskitytään nimenomaan sitä hyödyntäviin datatuotteisiin. Sosiaaliset suhteet ovat luonnostaan dataa, johon verkostomainen esitystapa sopii hyvin.

Tästä johtuen työssä painotetaan verkostoanalyysiä sekä visuaalista analytiikkaa datatuotteissa käytettävänä datan jalostamisen keinona.

MatchUs-järjestelmän käyttämä suosittelualgoritmi on rajattu työn ulkopuolelle. Työssä ei käsitellä myöskään suosittelujärjestelmien loogiseen toteutukseen liittyvää teoriaa. Järjestelmän web-arkkitehtuuria kuvataan tarkoituksellisesti ainoastaan yleisellä tasolla. Toteutusyksityiskohtiin syventymistä vältetään, koska *ADR*-tutkimusmenetelmän hengen mukaisesti työn tavoitteena on löytää mahdollisimman geneerisiä ratkaisuja, joita voidaan yleistää web-pohjaisten datatuotteiden kontekstissa. Datatuotteelle ominaiseen data-analytiikkaan sekä visuaaliseen analytiikkaan tutustumiseksi työssä analysoidaan Twitteristä kerätyn datan pohjalta generoitua sosiaalista verkostoa. Analyysin tulosten tarkempi käsittely on rajattu työn ulkopuolelle.

1.3 Työn rakenne

Työ on jäsennetty seuraavasti: luvussa 2 keskitytään datatuotteen määritelmään ja vastataan ensimmäiseen tutkimuskysymykseen kirjallisuuskatsauksen avulla. Tämän lisäksi luvussa käydään läpi visuaalisen analytiikan ja verkostoanalyysin perusteita. Luvussa 3 tutustutaan kirjallisuuteen pohjautuen web-pohjaisen datatuotteen arkkitehtuuriin vaikuttaviin tekijöihin sekä potentiaalisiin arkkitehtuurimalleihin. Arkkitehtuurin vaatimusten kartoituksen katsotaan olevan ADR-tutkimusmenetelmän ensimmäiseen vaiheeseen kuuluvaa ongelman määrittelyä. Näiden pohjalta tehdään konkreettinen ehdotus datatuotteen arkkitehtuurista, eli vastataan alustavasti työn toiseen tutkimuskysymykseen. Luvussa 4 siirrytään käytäntöön ja esitellään case-esimerkkeinä tässä tutkimuksessa toteutetut prototyypit. Prototyypit toimivat ADR-menetelmän toisen vaiheen mukaisina käytännön toteutuksina ehdotetusta arkkitehtuurimallista. Arkkitehtuurin onnistumista arvioidaan case-esimerkkien lomassa. Samalla pyritään oppimaan toteutustyöstä eli toimitaan ADR-menetelmän kolmannen vaiheen mukaisesti. Lopuksi arvioidaan edellisessä luvussa ehdotetun arkkitehtuurin yleistettävyyttä case-töistä saatujen käytännön kokemusten pohjalta. Luvussa 5 tehdään yhteenveto työn tuloksista sekä arvioidaan tutkimuskysymyksiin vastaamisen onnistumista.

2. DATATUOTE

Kuten johdannossa mainittiin, datatuote ei ole ainakaan toistaiseksi laajalti tunnettu termi, mutta sen englanninkielinen vastine "data product" on kuitenkin alkanut esiintymään viimeaikaisissa julkaisuissa, kuten Davenport ja Kudyba (2016), Davenport (2013) ja Fox ja Hendler (2011). Näissä datatuotetta on kuvattu järjestelmänä, joka **yhdistää** olemassa olevaa dataa sekä **analytiikkaa** ja siten **jalostaa** dataa luoden siten **lisäarvoa** käyttäjilleen.

Datan keräämiseen ja siihen liittyvään analytiikkaan keskittyvät tuotteet yhdistetään nykyään usein web-teknologioihin, mutta vastaavia järjestelmiä on ollut olemassa jo ennen nykyistä web-aikakautta. Esimerkiksi Meyer ja Zack (1996) ovat kirjoittaneet aiheesta jo vuonna 1996. Vaikka he eivät suoraan mainitse termiä "data product", niin julkaisun voidaan tulkita käsitelleen järjestelmiä, jotka täyttävät oleellisilta osin datatuotteen tunnusmerkit ja sopivat siten saman käsitteen alle.

2.1 Datatuotteen määritelmä

Eräs harvoista aiemmista julkaisuista, joka eksplisiittisesti mainitsee käsittelevänsä nimenomaan datatuotteita ja esittää termille jonkinlaisen määritelmän, on Davenportin ja Kudyban julkaisu vuodelta 2016. Siinä kuvataan datatuotetta seuraavan sitaatin mukaisesti (Davenport & Kudyba, 2016, s.83):

> ...we will focus on the combination of new analytical capabilities and burgeoning data assets that together form value-added information product offerings. In common parlance, these offerings are often called "data products."

Tämän lisäksi Meyer ja Zack, jotka toimivat Davenportin ja Kudyban esikuvina mutta eivät vielä vuonna 1996 käyttäneet käsitettä "datatuote", mainitsivat samaa tarkoittavan "informaatiotuotteen" ominaispiirteeksi myös **interaktiivisuuden**, jonka he määrittelivät seuraavasti (Meyer & Zack, 1996, s.49):

> Interactivity is the user's ability to dynamically select, manipulate, integrate, and format the information to suit particular and changing needs.

Käytännössä interaktiivisuudella voidaan tarkoittaa esimerkiksi sitä, että käyttäjä voi valita tai kustomoida datan esitystavan, tai mahdollisesti jopa muokata järjestelmän parametreja ja tutustua siten lähdedataan paremmin.

Meyer ja Zack (1996, s. 48) nostivat esiin myös arvon luomisen, joka pohjautuu ensisijaisesti datan **jalostamiseen** (engl. refine), joka voidaan jakaa kahteen lajiin: fyysiseen sekä loogiseen jalostamiseen. Esimerkki fyysisestä jalostamisesta on datan konvertoiminen muodosta toiseen, kun taas loogista jalostamista voi olla esimerkiksi datan luokittelu tai yhdistäminen toiseen tietolähteeseen. Datan jalostaminen saattaa myös sisältää esimerkiksi datan siistimistä (engl. data cleaning), jolla tarkoitetaan mm. datassa esiintyvien virheiden ja epäjohdonmukaisuuksien löytämistä ja poistamista datan laadun parantamiseksi (Rahm & Do, 2000, s. 3). Voidaan olettaa, että lähtökohtaisesti datatuotteen on tarkoitus jalostaa dataa siten, että siitä on välitöntä arvoa järjestelmän hyödyntäjälle. Tilanteesta riippuen datan jalostaminen voi kuitenkin luoda arvoa myös välillisesti, sillä esimerkiksi datan siistiminen tai joustavampaan formaattiin tallentaminen helpottaa sen uudelleenkäyttöä ja toimii siten arvon luomisen pohjana (Meyer & Zack, 1996, s. 48).

Näihin pohjautuen esitämme datatuotteelle seuraavanlaista määritelmää, jota käytämme myös tässä työssä:

> Datatuote on tietotekninen järjestelmä tai kokonaisuus, joka saa syötteekseen dataa yhdestä tai useammasta lähteestä ja **jalostaa** kyseistä dataa **vuorovaikutteisesti** tuoden siten **lisäarvoa** loppukäyttäjälleen.

Seuraavissa kappaleissa avataan termiä tarkemmin ja tuodaan määritelmän web-pohjaisten palveluiden kontekstiin.

Tässä työssä keskitytään web-pohjaisiin datatuotteisiin, jotka hyödyntävät vahvasti internetiä, web-teknologioita ja pilvipalveluita datan keräämiseen, käsittelyyn ja esittämiseen. Web-pohjaisuus ei kuitenkaan ole datatuotteelle määritelmällisesti pakollista, ja samoja periaatteita voi suurimmaksi osaksi yleistää myös ilman internetyhteyttä toimiviin datatuotteisiin.

Datatuotteen lähdedata voi olla peräisin mielivaltaisesta lähteestä, esimerkiksi palvelun käyttäjiltä tai toisesta web-palvelusta raavittuna (engl. scraping). Datalähteitä voi olla yksi tai useampia. Useampaa kuin yhtä datalähdettä käyttäviä palveluilta voidaan nimittää myös mashup- eli koostepalveluiksi: erään määritelmän mukaan mashup-palvelu on sovellus, joka yhdistää kolmansien osapuolien tarjoamaa dataa avoimien API-rajapintojen (*Application Programming Interface*) kautta käyttäjän omaan dataan (Weiss & Gangadharan, 2010, s.40). Toisin päin tarkasteltuna koostepalvelut voidaan luokitella datatuotteiksi, jos ne jalostavat tai analysoivat lähdedataansa datatuotteelle ominaisin tavoin, mutta koostepalvelu ei automaattisesti ole datatuote.

Datatuotteelle ominaisen arvon luomisen voidaan ajatella olevan käytännössä synonyymi etenkin markkinoihin liittyvässä tutkimuksessa käytetylle termille *arvon luominen* (engl. value creation), johon puolestaan liittyy oleellisesti termi *arvon kaappaaminen* (engl. value appropriation). Esimerkiksi Mizik ja Jacobson (2003) mainitsevat molemmat termit. Arvon luomisella tarkoitetaan prosesseja, jotka tuottavat hyödynnettävissä olevaa *arvoa*, kuten esimerkiksi innovaatioita tai tuotteita. Arvon kaappaamisella puolestaan tarkoitetaan luodun arvon muuttamista liikevoitoksi, esimerkiksi myymällä valmistettuja tuotteita tai kaupallistamalla innovaationsa. Toisin sanoen, jos datatuotteen halutaan olevan varsinainen tuote myös liiketoiminnan näkökulmasta, niin on huomioitava, että arvon luomisen lisäksi datatuotteen tai siihen liittyvän liiketoiminnan on mahdollistettava myös arvon kaappaaminen.

Datatuotteen hyödyllisyyden sekä arvon luomisen kannalta erittäin tärkeänä ominaisuutena voidaan pitää myös tuotteen käyttöliittymää tai prosessoidun datan pohjalta tuotettua esitystä (Meyer & Zack, 1996, s. 48). Tähän liittyy olennaisena osana visuaalinen analytiikka, jota käsitellään tarkemmin seuraavassa luvussa.

2.2 Visuaalinen analytiikka

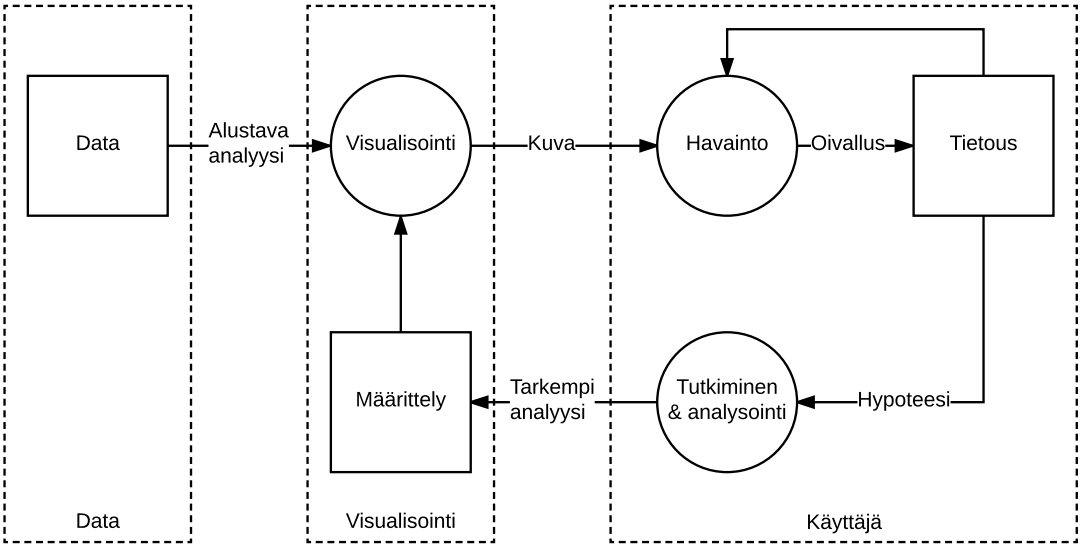
Datamäärien kasvaessa tiedon paikallistaminen muuttuu jatkuvasti haastavammaksi (Shneiderman, 1996, s. 1). Tämän lisäksi suurivolyyminen ja tekstipohjainen data on käytännössä aina ihmisnäkökulmasta vaikeasti hahmotettavaa, joten sen hyödyntäminen ei onnistu ilman koneellisesti tapahtuvaa datan käsittelyä ja jalostamista. Visuaalinen analytiikka vastaan tähän haasteeseen (Basole, 2009, s. 148). Näkökyky on ihmisen tehokkain aisti informaation vastaanottamiseen (Shneiderman, 1996, s. 2), joten visualisoinneilla on mahdollista helpottaa tiedon ymmärrettävyyttä olennaisesti.

Schreck ja Keim (2013, s. 68) määrittelevät visuaalisen analytiikan koneellista tiedonkäsittelyä (engl. computational knowledge discovery) sekä interaktiivista visualisaatiota yhdisteleväksi yhdistäväksi alaksi. Heidän mukaansa koneellisen tiedonkäsittelyn avulla datasta saadaan eroteltua mielenkiintoisimmat rakenteet, joiden interaktiivinen visualisointi herättelee analysoijan luovuutta sekä auttaa yhdistämään datan aiempiin pohjatietoihin.

Interaktiivisuuden tarpeellisuudelle on olemassa myös käytännönläheisempi perustelu. Visuaalisen analytiikan ensimmäinen haaste on löytää ja kehittää paras mahdollinen algoritmi halutun ongelman ratkaisemiseksi. Paraskaan algoritmi ei kuitenkaan käytännössä pysty ratkaisemaan käyttäjän puolesta haluttua tiedon esitystapaa tai analyysissä käytettäviä parametreja. Siksi käytetty algoritmi halutaan integroida interaktiiviseen visualisaatioon. (Keim et al., 2008, s. 158)

Koneellinen tiedonkäsittely voi tarkoittaa esimerkiksi datan jaottelua ryhmiin (engl. cluster) tai datan automaattista luokittelua (engl. predicting class labels). Visualisoinnissa data muokataan kognitiivisesti hyödylliseen visuaaliseen muotoon, kuten kuvaajaksi tai graafiksi. Tavoitteena on mahdollistaa datan hahmottaminen ja sen merkityksen arviointi näköaistin avulla. (Schreck & Keim, 2013, s. 69) Vastaavaan määrittelyyn on päätynyt myös Keim et al. (2008, s.157). Myös heidän mukaansa visuaalinen analytiikka yhdistelee automaattisia analysointimenetelmiä interaktiivisiin visualisointeihin, joiden tavoitteena on auttaa ymmärtämään suuria datamääriä ja helpottaa niihin pohjautuvien päätösten tekemistä.

Visuaalisen analytiikan etememisen tyypilliset vaiheet on esitetty kuvassa 2.1. Kuva korostaa datalähtöisyyttä sekä iteratiivista etenemistä: aluksi lähdetään liikkeelle datasta, jonka pohjalta luodaan alustava visualisointi. Sen perusteella tehdään havaintoja datasta sekä luodaan hypoteesi. Visualisointia päivitetään tarpeen mukaan ja päivitetystä visualisoinnista pyritään jälleen tekemään uusia havaintoja, joiden perusteella pyritään vahvistamaan tai hylkäämään aiemmin tehty hypoteesi.



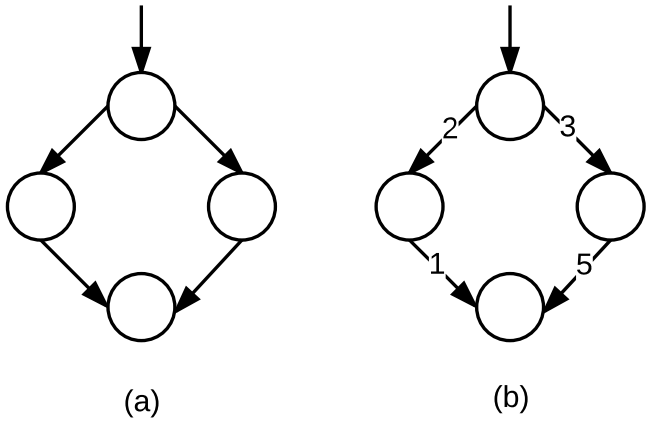
Kuva 2.1. *Visuaalisen analytiikan iteratiivinen eteneminen, pohjautuen Keim et al. (2008, s. 165)*

Visuaalisen analytiikan tutkimus on ehkä hieman yllättäen saanut vauhtia myös syyskuun 11. päivän jälkeisestä aikakaudesta, joka laukaisi etenkin Yhdysvalloissa valtavan panostuksen mahdollisten kansallisten turvallisuusuhkien ennakointiin ja torjuntaan. *Illuminating the Path*-julkaisussaan James J. Thomas ja Kristin A. Cook loivat laajalti tunnustetun pohjan visuaalisen analytiikan tutkimukselle sekä kehittämiselle samalla kuin sitoivat aiheen vahvasti Yhdysvaltoihin kohdistuvan terrorismiuhan torjuntaan. (Thomas & Cook, 2005) Seuraavissa aliluvuissa esitellään visuaalisen analytiikan taustalla olevat alueet: verkostoanalyysi, visuaalinen verkos-

toanalyysi, tiedon visualisointi sekä data-analytiikka.

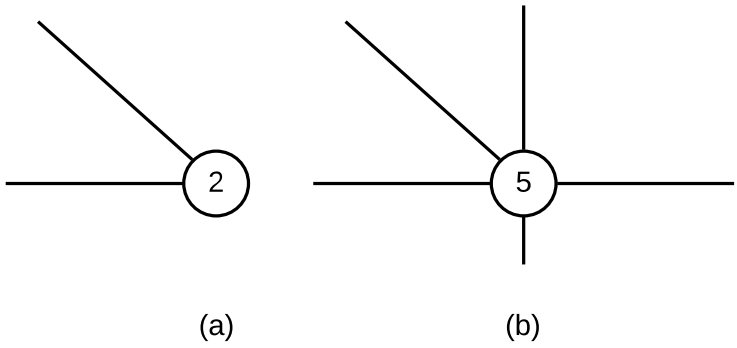
2.3 Verkostoanalyysi

Verkostoanalyysi on menetelmä, jossa tutkittava ilmiö mallinnetaan solmujen ja kaarien avulla verkostoina eli graafeina. Aiheen perusteos on esimerkiksi Wasserman ja Faust (1994) julkaisema *Social Network Analysis*. Verkostoissa solmut edustavat yksiköitä, ja kaaret niiden välisiä suhteita. Yksiköiden välisillä suhteilla voi olla suunta, jolloin muodostuu suunnattu verkosto, ja/tai yhteyden vahvuutta kuvaava painoarvo, jolloin muodostuu painotettu verkosto. Verkosto voi olla samaan aikaan sekä suunnattu että painotettu. Esimerkki suunnatusta verkostosta (a) sekä suunnatusta ja painotetusta verkostosta (b) on esitetty kuvassa 2.2. Verkostoista voidaan laskea tiettyjä tunnuslukuja, joista yksinkertaisin on solmun asteluku (engl. node degree). Solmun asteluvulla tarkoitetaan siihen johtavien kaarien lukumäärää eli solmun, johon johtaa yksi kaari, asteluku on 1. Solmujen astelukuja on havainnollistettu myös kuvassa 2.3.



Kuva 2.2. *Suunnattu verkosto (a) sekä suunnattu ja painotettu verkosto (b)*

Verkostoanalyysin juuret ovat sosiaalisessa verkostoanalyysissä (Wasserman & Faust, 1994), ja vaikka se ei ole ideana uusi, niin sitä on toistaiseksi hyödynnetty melko vähän muiden sovellusalueiden parissa (Huhtamäki, 2016, s. 14). Tyypillisesti verkostoanalyysiä käytetään sosiaalisia suhteita tutkittaessa tai mallinnettaessa. Tätä voidaan havainnollistaa yrityselämän esimerkillä: oletetaan verkosto, joka kuvaa yrityksen henkilöstön välisiä sosiaalisia suhteita. Verkoston solmut kuvaavat työntekijöitä, ja kaaret heidän välisiä suhteitaan. Yrityksen työntekijät tuntevat toimitusjohtajan ainakin nimeltä lähes poikkeuksetta, joten verkoston useimmista solmuista lähtee kaari toimitusjohtajaan. Toimitusjohtaja taas ei välttämättä tunne



Kuva 2.3. *Solmun asteluku. (a):n asteluku on 2, koska siihen johtaa kaksi (2) kaarta, ja (b):n asteluku on 5, koska siihen johtaa viisi (5) kaarta.*

kuin pienen osan työntekijöistään, joten toimitusjohtajasta työntekijöihin suuntautuvia kaaria on vähemmän. Vastaavasti yhteyden eli kaaren painoarvo voisi kuvata tässä esimerkissä suhteiden välistä voimakkuutta. Työntekijöiden suhde on tyypillisesti vahvin niiden henkilöiden kanssa, joiden kanssa he työskentelevät päivittäin. Vastaavasto toimitusjohtajan tunteminen ainoastaan nimeltä on heikko suhde.

2.4 Tiedon visualisointi

Tiedon visualisoinnilla pyritään esittämään dataa visuaalisessa muodossa, jotta ihmiset voivat hyödyntää sen tulkitsemisessa näköaistiaan (Gershon & Eick, 1998, s. 199). Se keskittyy dataan, jolle ei löydy luontaista esitystapaa 2- tai 3-ulotteisessa avaruudessa (Keim, 2002, s. 101). Visualisointien kehittyminen on mahdollistanut suurten datamäärien ymmärtämisen ja käyttämisen päätöksenteon pohjana aiempaa tehokkaammin (Gershon & Eick, 1998, s. 199). Yleensä tiedon visualisoiminen vaatii kuitenkin luovaa tai taiteellista panostamista, joka ei onnistu yksinomaan koneellisesti (Fox & Hendler, 2011, s. 707). Tästä johtuen ainakaan toistaiseksi ei ole olemassa valmiita työkaluja, jotka pystyisivät generoimaan laadukkaita visualisointeja mielivaltaisesta datasta ilman ihmisen apua.

Fox ja Hendler (2011) mukaan esimerkiksi tieteellisissä analyyseissä visualisointi toteutetaan usein vasta viimeisenä, mutta he painottavat julkaisussaan nopeatahtisen visualisoinnin etuja sekä tärkeyttä. Etenkin modernit web-teknologiat nähdään nopeammin generoitavien sekä interaktiivisten visualisointien mahdollistajina. Useat tutkijoiden käyttämät visualisointityökalut eivät kuitenkaan mahdollista web-pohjaisten ratkaisujen tapaan visualisoinnin sitomista sen käyttämään lähdedataan, jolloin visualisoinnit luonnollisesti vanhenevat heti kuin niiden esittämä data muuttuu tai päivittyy. Tilanteen parantamiseksi Fox ja Hendler (2011) kannusta-

vat tutkijoita ottamaan visualisointityökalut käyttöön jo aiempaa aikaisemmin sekä dokumentoimaan datan sekä visualisoinnin välisen yhteyden.

Tiedon visualisointi ja siihen liittyvät haasteet eivät tietenkään rajoitu pelkästään tiedepiireihin ja tieteelliseen dataan. Tyypillisiä esimerkkejä käytännön tilanteista, joissa visualisoinnit ovat tarpeen, ovat muun muassa elinkeinoelämän data, väestötilastot sekä verkostoluontoiset kokonaisuudet (Keim et al., 2008, s. 160).

2.5 Data-analytiikka

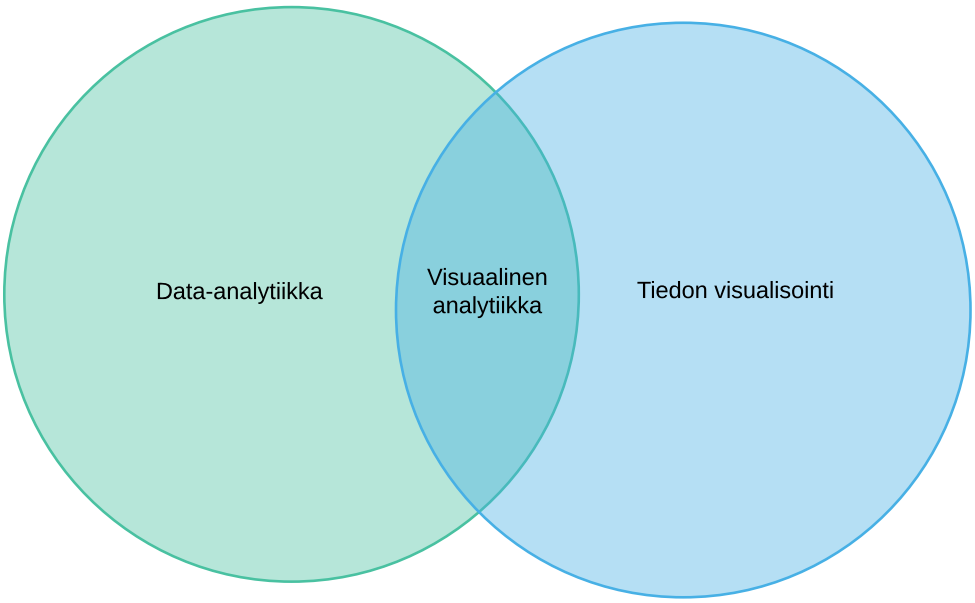
Data-analytiikka on poikkitieteellinen ala, jossa analysoidaan suuria datamääriä esimerkiksi tilastotieteen ja koneoppimisen avulla. Sen tavoitteena on päätöksenteon tukeminen. (Runkler, 2012, s. 2) Kirjallisuudessa data-analytiikka on jaettu usein kolmeen tai neljään osa-alueeseen: *deskriptiiviseen*, *prediktiiviseen*, *preskriptiiviseen* sekä *eksploratiiviseen* analytiikkaan (Davenport, 2013; Bendoly, 2016; Olshannikova, Olsson, Huhtamäki, & Kärkkäinen, 2017).

Deskriptiivinen eli *kuvaileva* analytiikka kuvaa mennyttä (Davenport, 2013) eli jo olemassa olevaa dataa (Olshannikova et al., 2017, s. 10). *Prediktiivinen* eli *ennustava* analytiikka mallintaa tulevia tapahtumia olemassa olevan datan pohjalta (Davenport, 2013). *Preskriptiivinen* eli *ohjaava* analytiikka tähtää suosituksiin (Davenport & Kudyba, 2016) ja *eksploratiivinen* eli *tutkiva* analytiikka etsii säännönmukaisuuksia tai toistuvia rakenteita (engl. patterns), suhteita sekä trendejä (Andrienko & Gennady, 2006) tai ylipäätään ymmärrystä olemassa olevasta datasta (Tukey, 1977).

Visuaalisen analytiikan yhteydessä keskitytään usein etenkin data-analytiikan deskriptiiviseen sekä eksploratiiviseen osa-alueeseen. Eksploratiivisen analytiikan avulla pyritään ymmärtämään tutkimuksen kohteena olevaa dataa sekä selvitetään mahdollisuuksien mukaan, että löytyykö datan joukosta viitteitä siitä, että lisätutkimuksilla voitaisiin löytää jotakin kiinnostavaa (Tukey, 1977, s. 1-3). Visuaalisen analytiikan kontekstissa etenkin vaihe, jossa pyritään tulkitsemaan luotua visualisointia sitä interaktiivisesti säätämällä on eksploratiivisen analytiikan toteuttamista käytännössä. Jos eksploratiivinen analytiikka johtaa mielenkiintoisiin löydöksiin, niin niille pyritään löytämään selitykset deskriptiivisen analytiikan avulla.

2.6 Käsitteiden väliset suhteet

Edellä mainituilla käsitteillä on paljon yhteistä, joten on hyvä vielä palata niiden välisiin suhteisiin ja eroihin. Käsitteiden väliset suhteet on esitetty graafisesti kuvassa 2.4. *Data-analytiikka* on pääkäsite, joka sisältää kaikenlaiset menetelmät ja prosessit, joiden päämääränä on datan analysoiminen. *Tiedon visualisointi* puolestaan tähtää datan esittämiseen havainnollisesti visuaalisessa muodossa. Se ei kuitenkaan välttämättä liity analytiikkaan tai hyödynnä kehittyneitä data-analytiikan algoritmeja (Keim et al., 2008, s. 158). *Visuaalinen analytiikka* yhdistää edellä mainitut ja pyrkii mahdollistamaan datan tutkimisen ja analysoimisen sen pohjalta tuotettujen interaktiivisten visualisointien pohjalta.



Kuva 2.4. *Venn-diagrammi data-analytiikan, tiedon visualisoinnin ja visuaalisen analytiikan välisistä suhteista*

Visualisoinnit ovat siis keskeisessä osassa niin tiedon visualisoinnissa kuin visuaalisessa analytiikassakin, mutta ensin mainittu ei painota niiden interaktiivisuutta. Visuaalinen analytiikka sisältää siis aina tiedon visualisointia, mutta tiedon visualisointi ei ole automaattisesti visuaalista analytiikkaa. Vastaavasti visuaalinen analytiikka on aina data-analytiikkaa, mutta data-analytiikka ei ole visuaalista analytiikkaa, jos se ei perustu visualisointien hyödyntämiseen.

2.7 Visuaalisen analytiikan tyypillisimmät tekniset haasteet

Kuten jo työn johdannossa todettiin, datan analysoiminen on usein haastavaa, eikä visuaalinen analytiikka ole sen suhteen poikkeus. Fox ja Hendler (2011, s. 707) mainitsivat visualisointien vaativan tavallisesti luovaa tai jopa taiteellista ponnistelua, joka ei onnistu yksinomaa koneellisesti. Visuaaliseen analytiikkaan liittyy kuitenkin myös teknisiä haasteita, joita Keim et al. (2008, s. 166-168) ovat koonneet julkaisussaan yhteen. Seuraavaksi käymme läpi heidän listansa läpi ja lopuksi arvioimme sitä nykypäivän kontekstissa.

1. **Skaalautuvuus datan määrän ja ulottuvuuksien suhteen:** Visuaalisen analytiikan tekniikoiden tulisi skaalautua sekä suurten datamäärien että jatkuvien, suuren kaistanleveyden datavirtojen suhteen ja kyetä esittämään niitä graafisesti. Tämä asettaa vaatimuksia visualisointialgoritmien suorituskyvylle.
2. **Datan ja graafisen esityksen laatu:** Keskeinen ongelma visuaalisessa analytiikassa on virheiden mahdollisuus tulosten tulkitsemisessa. Ongelmia saattaa seurata esimerkiksi lähdedatan puutteellisuudesta tai virheellisyydestä johtuen tai analyysissä käytetyn algoritmin epäsopivuudesta kyseiseen ongelmaan. Siksi lähdedatan laatua ja valitun algoritmin soveltuvuutta tulisi arvioida visuaalisen analytiikan avulla saatujen tulosten arvioinnin yhteydessä.
3. **Visuaalinen esitystapa sekä sen yksityiskohtien määrä:** Visualisoinnissa pitäisi pyrkiä löytämään tasapaino yksityiskohtien sekä helpon yleiskuvan esittämisen välille. Useampi visualisointi eri abstraktiotasoilla voi olla yksi ratkaisu.
4. **Käyttöliittymät sekä vertauskuvat:** Visuaalisen analytiikan sovellusten käyttöliittymien tulisi olla niin yksinkertaisia ja intuitiivisia, että tulosten analysoijan ei tarvitse erikseen keskittyä niihin. Monimutkaiset käyttöliittymät voivat vastaavasti häiritä käyttäjää ja viedä huomiota pois olennaisesta.
5. **Päätelaitteet:** Visualisointien tulisi tukea erilaisia päätelaitteita suurinäyttöisistä tietikoneista mobiililaitteisiin asti.
6. **Arviointi:** Koska visualisella analytiikalla tutkitaan vaihtelevia sekä usein kompleksisia ongelmia, myös saatujen tulosten arvioiminen ja vertaaminen vaatimuksiin on vaikeaa.
7. **Infrastruktuuri:** Suurten datamäärien ja visualisointien hallinta asettaa vaatimuksia niin keskusmuistin kuin tallennustilan suhteen. Interaktiivisuus edellyttää reaaliaikaista näkymän päivittämistä, kun taas datan analysointi ja käsittely voi itsessään viedä tunteja. Sulavien interaktioiden mahdollistaminen analysoijalle vaatii asynkronisten laskentaoperaatioiden yhdistämistä hybridiratkaisuihin, joissa laskenta-aikaa saadaan lyhennettyä tulosten laadun kustannuksella.

(Keim et al., 2008, s. 166-168)

Kohta 3 voidaan ratkaista nykyään myös interaktiivisen visualisoinnin avulla, joten haaste ei välttämättä ole nykyään yhtä suuri kuin vuonna 2008. Web-pohjaisuus sekä nykypäivän huomattavasti vuotta 2008 kehittyneemmät mobiilipäätelaitteet helpottavat myös huomattavasti ongelmaa 5, sillä web-teknologioilla toteutettu visualisointi voidaan usein toteuttaa myös mobiililaitteella toimivaksi. Toisaalta tämä on syytä ottaa huomioon jo käyttöliittymäsuunnittelussa. 1 sekä 7 ovat puolestaan edelleen erittäin relevantteja. Myös nykyään ratkaisu suorituskykyhaasteisiin on tyypillisesti raskaan data-analyysiosuuden ajaminen jo ennakkoon valmiiksi, jolloin visualisoinnin ei tarvitse optimitapauksessa muuta kuin esittää valmista dataa. Kohdat 2 ja 6, eli datan vaihteleva laatu ja saatujen tulosten arvioimisen hankaluus, ovat myös edelleen ajankohtaisia. Yleistä ratkaisua näihin ei ole näkyvissä ainakaan lähitulevaisuudessa, joten molempia joutuu aina arvioimaan tapauskohtaisesti.

3. DATATUOTTEEN ARKKITEHTUURI

Ohjelmistoarkkitehtuuri on abstrakti käsite, jolla tarkoitetaan pääasiassa ohjelmistotuotteen korkean tason suunnitteluratkaisuja sekä niiden toteutusta. Ohjelmiston arkkitehtuuri vaikuttaa yleisesti ohjelmiston laatuun, kuten ylläpidettävyyteen, luotettavuuteen, käytettävyyteen, suorituskykyyn sekä joustavuuteen, ja ohjelmiston koon kasvaessa arkkitehtuurin merkitys vain korostuu (Babar, Zhu, & Jeffery, 2004). Kuitenkin myös ohjelmistoarkkitehtuuri on termi, joka on määritelty monin eri tavoin. Kirjallisuudessa sen on määritelty liittyvän mm. ohjelmiston komponentteihin sekä niiden välisiin rajapintoihin (Boehm & Scherlis, 1992, s. 64), ohjelmiston abstraktioon sekä tyyliin (Perry & Wolf, 1992, s. 40), sekä ohjelmiston rakenteeseen, toiminnallisuuden jaotteluun, fyysiseen jaotteluun sekä käytettyjen elementtien yhdistämiseen (Shaw & Garlan, 1994, s. 2).

On myös esitetty, että ohjelmiston koon kasvaessa sen suunnitteluhaasteet eivät enää liity yksittäisiin algoritmeihin tai tietorakenteisiin, vaan nimenomaan korkean tason arkkitehtuuriin (Shaw & Garlan, 1994). Tätä perustellaan sillä, että mahdollisuus rakentaa ohjelmistot useista komponenteista helpottaa yksityiskohtien suunnittelua ja toteuttamista, mutta näiden väliseen kommunikaatioon ja jaotteluun ei ole olemassa yksiselitteisiä malleja. Tämä on ongelma, johon erilaiset ohjelmistoarkkitehtuurimallit pyrkivät tarjoamaan ratkaisuja.

Tässä luvussa vastataan alustavasti työn toiseen tutkimuskysymykseen ja ehdotetaan datatuotteelle soveltuvaa arkkitehtuuria. Koska yleiskäyttöisen arkkitehtuurin ehdottaminen on laaja kysymys, niin vastausta haetaan usean näkökulman avulla. Ensimmäisenä käydään läpi yleisiä tavoitteita, joihin arkkitehtuurin valinnalla kannattaa pyrkiä. Kuten Shaw ja Garlan (1994, s. 2) esittivät, ohjelmiston arkkitehtuuriin liittyy muun muassa toiminnallisuuden jaottelu ohjelmiston eri osasten välille. Jotta jaotteluun vaikuttavia tekijöitä voitaisiin ymmärtää, aluksi perehdytään datatuotteen kehittämiseen yleisellä tasolla muun muassa tähän soveltuvien prosessimallien avulla. Prosessimalleista pyritään tunnistamaan kokonaisuuksia, jotka voidaan siirtää datatuotteen kontekstiin. Tämän jälkeen luodaan katsaus web-sovellusten kehittymiseen ja nykytilaan. Näihin pohjautuen ehdotetaan web-pohjaiselle datatuotteelle soveltuvaa toiminnallisuuksien jaottelua eri ohjelmistokomponenttien näkökulmasta. Tämän jälkeen perehdytään yleisesti käytettyihin ohjelmistiarkkitehtuurimalleihin, joita voisi potentiaalisesti hyödyntää web-pohjaista datatuotetta kehitettäessä. Lopuksi yhdistetään aiemmin läpikäydyt asiat ja ehdotetaan web-pohjaiselle

datatuotteelle soveltuvaa arkkitehtuuria.

3.1 Arkkitehtuurin tavoitteet

Yleisesti ottaen hyvä lähtökohta ohjelmiston arkkitehtuurille on olemassa olevien kirjastojen ja työkalujen hyödyntäminen, koska sillä voidaan nopeuttaa tuotekehitystä. Riittävän nopea tuotekehitys on usein jo itsessään markkinoilla selviämisen edellytys: muun muassa Meyer ja Zack (1996, s. 49) totesivat, että useimmiten uuden markkinaraon ilmaantuessa sille pyrkijöitä ilmestyy nopeasti, ja jos oma tuotekehitys on liian hidasta, markkinat menetetään helposti kilpailijoille. He toteavat myös, että käyttämällä jo olemassa olevia komponentteja uudelleen on usein mahdollista vähentää tuotekehityksen kustannuksia huomattavasti.

Olemassa olevan koodin uudellenkäyttämisen lisäksi nopeaa tuotekehitystä voidaan tavoitella toteuttamalla alussa ainoastaan niin sanottu MVP *(Minimum Viable Product)* eli minimiversio tuotteesta. MVP-versiota toteutettaessa tarkoituksena on karsia tuotteesta tai uudesta ominaisuudesta pois kaikki ylimääräinen ja toteuttaa minimiajalla ja -resursseilla yksinkertaistettu versio, joka riittää tuoteidean testaamiseen. Ideana on, että tuotteen tai ominaisuuden hyödyllisyys testataan jo ennen kuin sen viimeistelyyn ja hiomiseen on käytetty resursseja. Tällöin se on helpommin hylättävissä, jos tuoteidea todetaan turhaksi tai muutoin epäonnistuneeksi. MVP-ideologian puolesta puhuu etenkin Ries (2011) kirjassaan *The Lean Startup*.

Toisaalta arkkitehtuurista sekä omasta koodista on hyvä pyrkiä tekemään uudelleenkäytettävää, vaikka lyhyellä aikataululla se ei useinkaan palvele mahdollisimman nopeaa tuotekehitystä. Knight ja Dunn (1998, s. 293-294) mainitsevat koodin uudelleenkäytön tavoittelemisen tyypillisimmiksi syiksi tuottavuudeen kasvun ja siitä seuraavan tuotekehityskustannusten takaisinmaksuajan lyhenemisen: uusia tuotteita on nopeampi kehittää, jos niiden pohjana voidaan käyttää aiempia toteutuksia. Toisaalta julkaisussaan he korostavat, että tämä ei ole ainut syy tavoitella aiempien ohjelmistokomponenttien uudelleenkäyttöä, vaan sen avulla voidaan parantaa myös tuotteiden laatua: jos olemassa olevat ratkaisut ovat jo valmiiksi testattuja ja hyväksi todettuja, on oletettavaa, että niiden käyttö johtaa parempaan lopputulokseen kuin vastaavan toiminnallisuuden toteuttaminen uudelleen. Luonnollisesti tämä edellyttää sitä, että alkuperäinen toteutus on tehty alun perin riittävän laadukkaasti.

Tuotekehityksessä joudutaan siis tasapainoilemaan osittain vastakkaisten tavoitteiden välillä. Toisaalta ideologiat voidaan jossain määrin myös yhdistää; ensin toteutetaan nopeasti MVP ja unohdetaan uudelleenkäyttö, ja kun idea on saatu validoitua,

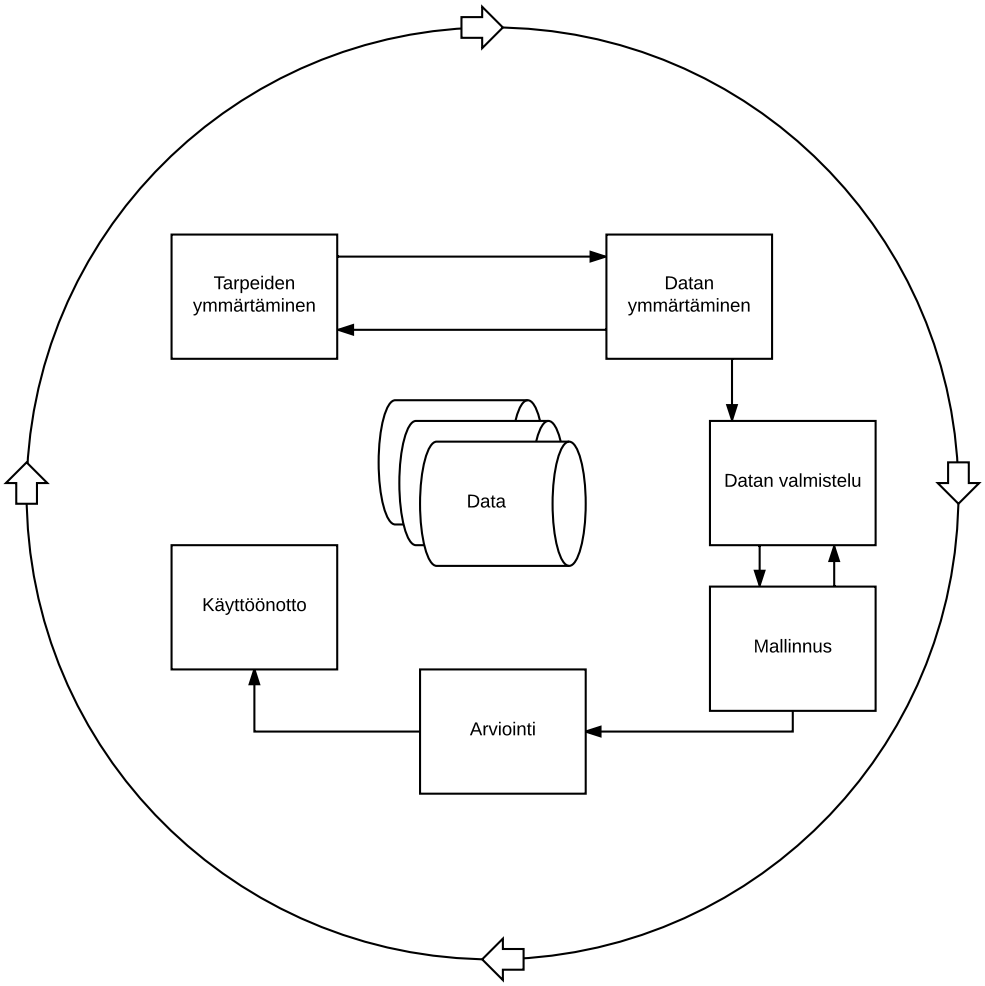
muokataan toteutusta tai aloitetaan jopa täysin puhtaalta pöydältä ja lähdetään tavoittelemaan hienostuneempaa arkkitehtuuria.

3.2 Datatuotteen kehittäminen

Datatuotteet voidaan usein nähdä data-analytiikan työkaluina, ja niiden käyttämän lähdedatan jalostaminen perustuu usein nimeomaan data-analytiikkaan tai visuaaliseen analytiikkaan. Siksi voidaan perustellusti olettaa, että data-analytiikan tutkimuksessa hyväksi havaittuja periaatteita voi yleistää myös datatuotteiden kontekstiin. Wirth ja Hipp esittivät vuonna 2000 **CRISP-DM**-prosessimallin *(CRoss Industry Standard Process for Data Mining)*, joka on tarkoitettu datalouhinnan projekteille (Wirth & Hipp, 2000). CRISP-DM jakaa datalouhinnan kuuteen vaiheeseen, jotka on esitetty myös kuvassa 3.1:

1. Tarpeiden ymmärtäminen
2. Datan ymmärtäminen
3. Datan valmistelu
4. Mallinnus
5. Arviointi
6. Käyttöönotto

Tarpeiden ymmärtäminen viittaa projektin bisnestarpeiden sekä -tavoitteiden määrittelyyn, ja sen pitäisi olla suunnittelun lähtökohtana. *Datan ymmärtäminen* alkaa alustavalla datan keräämisellä, ja pyrkii identifioimaan mahdolliset datan laatuun liittyvät ongelmat sekä parhaassa tapauksessa johtaa jo ensimmäisiin löydöksiin. Tämä vaihe on sidoksissa tarpeiden ymmärtämiseen, koska saatavilla oleva data vaikuttaa vahvasti projektin mahdollisuuksiin ja siten jo niiden suunnitteluun. *Datan valmistelu* sisältää vaiheet, joita tarvitaan lopullisen (tutkimus)datan tuottamiseen raakadatan pohjalta. Tyypillisesti tämä tarkittaa mm. datan keräämistä, siistimistä sekä muuta alustavaa käsittelyä. *Mallinnus* on vaihe, jossa varsinainen tiedon louhinta alkaa. Mallinnus sisältää tyypillisesti useiden menetelmien kokeilemista. Vaihe on sidoksissa datan valmisteluun, sillä eri menetelmät voivat vaatia datan konvertoimista erilaiseen muotoon, ja toimivimpia menetelmiä ei yleensä löydetä ennen kuin useampaa menetelmää on kokeiltu. *Arviointiin* siirryttäessä projektissa on jo saatu mallinnettua sekä analysoitua dataa. On kuitenkin tärkeää pysähtyä arvioimaan, että ovatko tulokset kelvollisia ja vastaavatko ne projektin tavoitteita. *Käyttöönotossa* projektissa mallinnettu data viedään käytäntöön. Usein tähän mennessä mallinnetut tulokset ovat vielä käsittelemättömässä, lähinnä data-analyytikoille soveltuvassa muodossa. Jotta niitä voidaan oikeasti hyödyntää, ne on saatava muotoon, jonka



Kuva 3.1. *CRISP-DM-prosessimallin vaiheet, pohjautuen Wirth ja Hipp (2000, s. 5)*

voi esittää suoraan myös loppukäyttäjille. Yksinkertaisimmillaan tämä voi tarkoittaa esimerkiksi tulosten generoimista raporteiksi. (Wirth & Hipp, 2000, s. 5-7)

CRISP-DM-prosessimalli on suunniteltu ensisijaisesti geneeriseksi malliksi datan louhintaan keskittyville projekteille, ja siksi sitä sitä täytyy soveltaa kun se tuodaan datatuotteiden kontekstiin. Toisaalta koska ainakin tässä työssä käsiteltävät datatuotteet liittyvät vahvasti myös visuaaliseen analytiikkaan sekä visuaaliseen verkostoanalyysiin, niin ideoita niiden toteuttamiseksi voidaan hakea myös Huhtamäen vuonna 2016 esittämästä **Ostinato**-prosessimallista, joka on suunnattu datalähtöiselle visuaaliselle verkostoanalytiikalle (Huhtamäki, 2016). Ostinato-mallin mukaan kehitysprosessi koostuu karkeasti kahdesta päävaiheesta: Ensimmäinen vaihe sisältää lähdedatan keräämisen ja puhdistamisen ja toinen vaihe verkoston sekä visualisoinnin rakentamisen. Ostinato-mallissa nostetaan myös esille aiheeseen oleellisesti liittyviä sääntöjä ja ohjenuoria, jotka tulisi huomioida järjestelmän suunnittelussa ja toteuttamisessa. Näitä ovat *jatkuva datan kerääminen, tutkiminen, läpinäky-*

vyys, löyhä sidonta, toistettavuus, automatisointi, manuaalisten vaiheiden mahdollistaminen, matala oppimiskynnys sekä *yhteensopivuus*. Seuraavaksi käydään näistä jokainen tarkemmin läpi Huhtamäen julkaisuun pohjautuen (Huhtamäki, Russell, Rubens, & Still, 2015).

Jatkuvalla datan keräämisellä korostetaan, että datan kerääminen on usein erinäisistä rajoituksista johtuen hidasta, sekä data luonteeltaan jatkuvasti päivittyvää. Näin on esimerkiksi sosiaalisen median kontekstissa. Siksi datan keräämisen tulisi olla automatisoitua ja tapahtua mahdollisuuksien mukaan jatkuvasti.

Tutkimisella tarkoitetaan sitä, että prosessin on oltava niin joustava, että kaikkien siihen osallistuvien tulisi voida suorittaa prosessin mikä tahansa yksittäinen vaihe ainakin jollain tapaa, vaikka heillä ei olisikaan teknistä osaamista koko prosessin tehokkaaseen toteuttamiseen. Tätä voidaan helpottaa esimerkiksi tallentamalla data jaettuihin hakemistoihin, joihin kaikilla asianosaisilla on pääsy.

Läpinäkyvyydellä tarkoitetaan, että prosessissa käytetty data sekä välitulokset tulisi saattaa saataville riittävän avoimesti, jotta kaikki tutkimukseen osallistuvat henkilöt voivat tutkia niitä myös omatoimisesti. Tämän varmistamiseksi kannattaa huomioida mm. datan tallentaminen riittävän laajalti tuetuissa formaateissa.

Löyhällä sidonnalla viitataan siihen, että prosessointijärjestelmä koostuu käytännössä aina useista erillisistä komponenteista, ja niiden väliset sidokset tulisi toteuttaa niin löyhästi, että myös yksittäisen komponentin vaihtaminen onnistuu muihin koskematta. Tämä mahdollistaa esimerkiksi sen, että käytetyt kirjastot tai visualisointityökalut voidaan vaihtaa toisiin, jos parempia vaihtoehtoja tulee saataville tai tähän on muuten tarvetta.

Toistettavuudella halutaan varmistaa, että myös tutkimusryhmän ulkopuolinen taho voi toistaa tutkimuksen sen tulosten varmistamiseksi. Toisaalta tämä liittyy myös itse järjestelmän toteuttamiseen: prosessin tulee olla toistettavissa myös päivitetyllä lähdedatalla.

Manuaalisten vaiheiden mahdollistaminen perustuu huomioon siitä, että vaikka optimitilanteessa koko prosessi olisikin automatisoitu, niin käytännössä tätä ei useinkaan saavuteta, etenkään järjestelmän ollessa yhä keskeneräinen. On myös mahdollista, että yksittäisiä vaiheita suoritetaan myöhemmin käsin, jotta päästään kokeilemaan uusia menetelmiä. Siksi prosessin yksittäisten vaiheiden tulisi olla tehtävissä myös manuaalisesti.

Automatisointi tähtää siihen, että koko prosessi saataisiin automatisoitua ja siten esimerkiksi visualisoitu lopputuote muuttuisi automaattisesti lähdedatan päivittyessä. Eksploratiivisen analytiikan yhteydessä tämä ei kuitenkaan ole aina saavutettavissa.

Matalalla oppimiskynnyksellä halutaan helpottaa analytiikkaprosesseille tyypillistä tilannetta, jossa analyysin kohde vaatii niin vahvaa alueosaamista myös analytiikan kohdealalta, että sitä ei löydy yksinomaan analytiikkajärjestelmän toteuttajilta. Siksi kohdealueen asiantuntijoiden, joilta puolestaan puuttuu usein syvempi tekninen ymmärrys analytiikasta, ottaminen mukaan analytiikkaprosessiin on usein tärkeää. Tätä voidaan helpottaa kiinnittämällä huomiota prosessin ymmärtämisen helppouteen sen toteutusvaiheessa.

Yhteensopivuuden tavoittelulla halutaan varmistaa, että tulosten analysoinnissa ja jatkokäytössä voidaan käyttää valmiita, hyväksi todettuja ja yleisesti käytettyjä työkaluja, kuten Gephiä (Bastian, Heymann, & Jacomy, 2009) tai D3.js-kirjastoa (Bostock, Ogievetsky, & Heer, 2011).

Sekä CRISP-DM että Ostinato-mallit nostivat datan keräämisen sekä valmistelun ja puhdistamisen selvästi yksittäiseksi kokonaisuudeksi. Tämä on yleistettävissä suoraan myös datatuotteen kehittämisen yksittäiseksi vaiheeksi. CRISP-DM-malli puhui datan mallinnuksesta ja Ostinato verkoston rakentamisesta, mutta datatuotteiden tapauksessa nämä tarkoittavat yksinkertaisesti datan käsittelyä sekä analysointia. CRISP-DM mallin viimeistä vaihetta nimitettiin käyttöönotoksi ja Ostinato tähtäsi visualisoinnin luomiseen. Käytännössä molempien perimmäisenä tarkoituksena oli kuitenkin saada aiemmin käsitelty data selkeästi käyttäjille esitettävään muotoon.

Näihin perustuen datatuotteen teknisen toteuttamisen voidaan ajatella koostuvan ainakin kolmesta kokonaisuudesta:

1. Datan kerääminen, puhdistaminen ja tallentaminen
2. Datan analysoiminen ja tulosten tallentaminen
3. Käyttöliittymän tai visualisoinnin toteuttaminen

Etenkin Ostinato-malli korostaa vaiheiden automatisointia, ja sekä Ostinato että CRISP-DM nostavat esiin kehitysprosessin iteratiivisuuden sekä vaiheiden keskinäiset riippuvuudet: käyttöliittymää ei voi toteuttaa, ennen kuin dataa ja sen mahdollisuuksia on tutkittu. Toisaalta varsinkin Ostinato korostaa, että prosessin vaiheiden välistä riippumattomuutta kannattaa kuitenkin tavoitella; tällöin esimerkiksi datalähteen formaatin muuttuminen XML[1]:stä JSON[2]:iin ei välttämättä edellytä muutoksia muualle kuin datan keräämiseen.

[1]Extensible Markup Language

[2]JavaScript Object Notation, joka ei nimestään huolimatta ole suunnattu ainoastaan JavaScriptillä käytettäväksi

Loppukäyttäjien näkökulmasta datatuotteen tärkein kokonaisuus on käyttöliittymä, koska se on järjestelmän ainut osa, joka näkyy suoraan käyttäjille. Siksi saattaa olla mielekästä aloittaa datatuotteen kehittäminen käyttöliittymästä, jotta sen hyödyllisyyttä päästäisiin testaamaan MVP-hengessä mahdollisimman aikaisin. Tämä voi edellyttää datan käsittelyyn liittyvien vaiheiden ohittamisen ja korvaamisen synteettisellä datalla. Toisaalta datatuotteen hyödyllisyyttä loppukäyttäjille ei useinkaan voi arvioida ilman autenttista dataa. Autenttisen datan luominen ennen data-analyysiosuuden valmistumista voi olla vaikeaa, mutta tilanteesta riippuen myös datan hankkimista voidaan lähestyä ketterän ohjelmistokehityksen keinoin. Esimerkiksi hitaan data-analyysiosuuden valmistumista voidaan nopeuttaa ajamalla järjestelmään aidosta datasta koostuva, mutta määrältään rajoitettu aineisto.

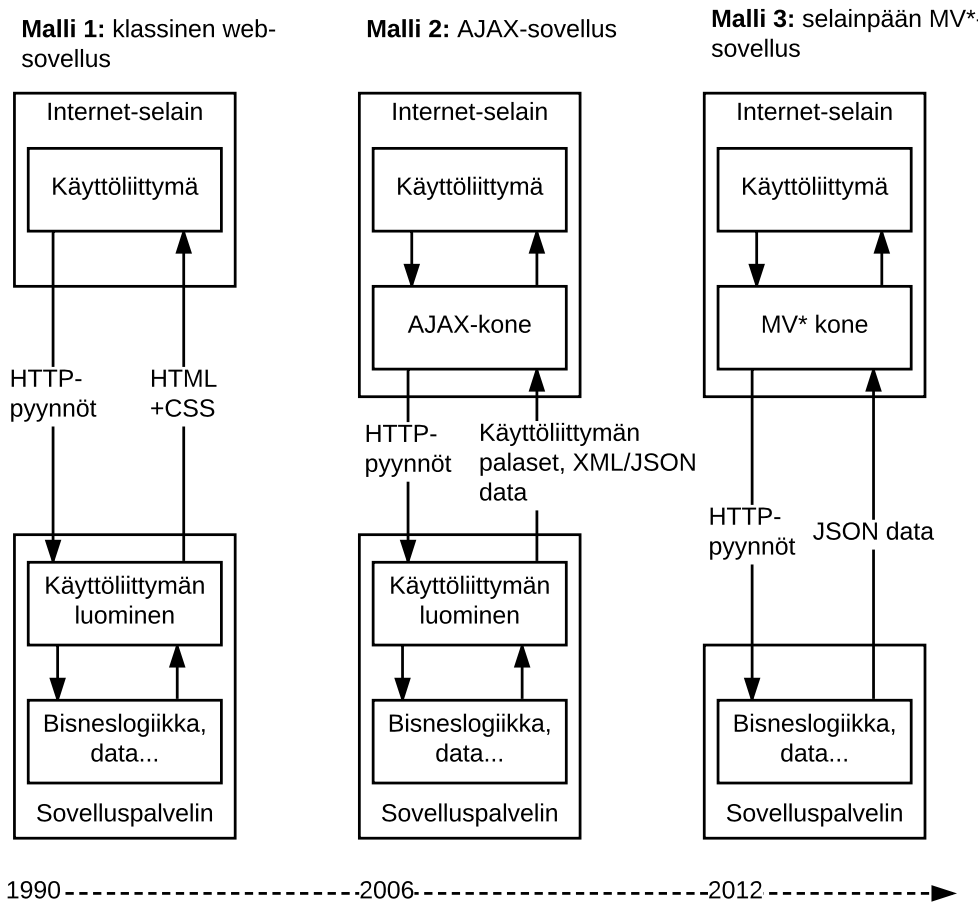
3.3 Web-sovellusten kehittyminen

Web-sovellukset ja niiden arkkitehtuurit ovat muuttuneet ja kehittyneet pitkälle nykymuotoisen internetin alkuaikoihin nähden. Kehityksen voidaan nähdä tapahtuneen esimerkiksi kolmen päävaiheen kautta, kuten Petitit ja Tricot (2014) esittivät:

Ensimmäisessä vaiheessa, 1990-luvun alkupuolella, elettiin klassisten web-sovellusten aikaa. Tuolloin web-sovellukset olivat lähinnä staattista HTML:ää, CSS:ää sekä mahdollisesti yksinkertaista JavaScriptiä. Sivut eivät olleet dynaamisia, ja aina kun käyttäjän toiminta edellytti uuden datan esittämistä, koko HTML-sivu jouduttiin hakemaan palvelimelta. Aikakauden perusarkkitehtuuri on esitetty kuvan 3.2 ensimmäisessä sarakkeessa. (Petitit & Tricot, 2014)

Toisessa vaiheessa, 2000-luvun ensimmäisen vuosikymmenen puolivälin tienoilla, siirryttiin AJAX (*Asynchronous JavaScript And XML*)-sovellusten aikakaudelle. Tuolloin selainten kehittyminen mahdollisti asynkronisten HTTP-kutsujen *(Hypertext Transfer Protocol)* tekemisen selaimesta palvelimelle, kun taas JavaScript mahdollisti web-sivujen sisällön manipuloinnin. Näiden ansiosta näkymästä toiseen siirtyminen voitiin tehdä lataamalla ainoastaan muuttunut näkymä AJAX-kutsun avulla palvelimelta käyttäjän selaimeen ja vaihtamalla se alkuperäisen näkymän tilalle JavaScriptin avulla. Tällöin koko sivua ei tarvinnut enää ladata uudelleen. Aikakauden perusarkkitehtuuri on esitetty kuvan 3.2 keskimmäisessä sarakkeessa. AJAX-sovellukset mahdollistivat paremmat vasteajat sekä monipuolisempien sovelusten kehittämisen. Haasteena oli kuitenkin sovellusohjelmoinnin huomattavasti kasvanut kompleksisuus. Esimerkiksi jQueryllä toteutettujen käyttöliittymien ylläpito oli usein erittäin haastavaa, ja aikakaudelle tyypilliset Javalla toteutetut backend-ratkaisut olivat usein myös raskaita ja monimutkaisia. (Petitit & Tricot, 2014)

Kolmannessa vaiheessa siirryttiin nykyisten web-sovelusten aikakaudelle. Moderneissa sovelluksissa käyttöliittymälogiikka on viety käytännössä kokonaan selainpäähän ja AJAX-kutsuilla haetaan palvelimelta ainoastaan esitystapaan sitomatonta dataa, jonka pohjalta sovellus pystyy generoimaan uutta sisältöä selainpäässä. Mallin mukainen sovellus on esitetty kuvan 3.2 kolmannessa sarakkeessa. (Petitit & Tricot, 2014)



Kuva 3.2. *Web-arkkitehtuurien kehitys, pohjautuen Petitit ja Tricot (2014)*

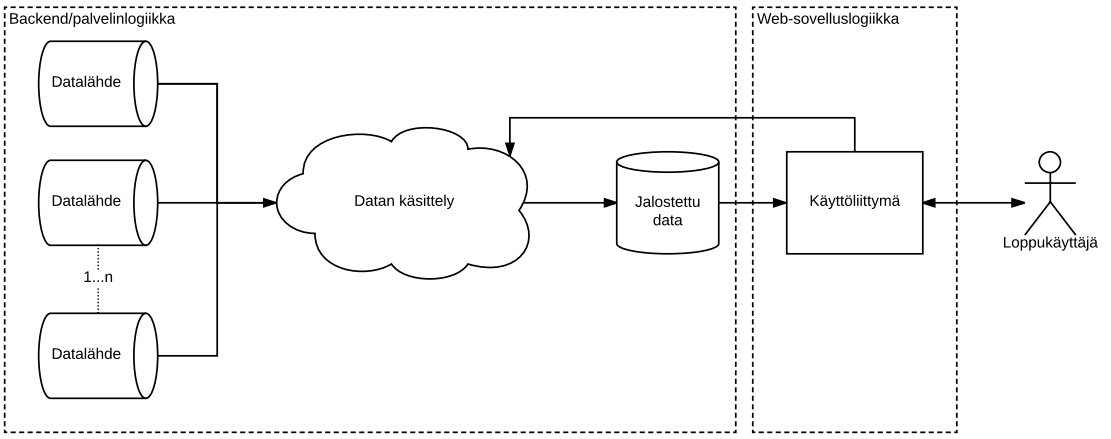
Näitä kolmannen vaiheen web-sovelluksia, joissa käyttöliittymälogiikka on siirtynyt selaimiin ja itse käyttöliittymä on haettu jo ensimmäisellä sivunlatauksella, kutsutaan yleisesti termillä SPA eli *Single Page Application*, yhden sivun sovellus. Mikowski ja Powell (2013) mukaan SPA-sovelluksen eduksi voidaan laskea muun muassa vasteajat: kun koko sovellus on ladattu selaimeen jo ensimmäisellä sivulatauksella, sen sisällä navigoiminen onnistuu täysin ilman verkkoviivettä, samaan tapaan kuin työpöytäsovelluksissa. Samalla vältetään myös selainruudun välähtäminen, joka ilmenee sivulta toiselle siirryttäessä perinteisemmissä web-sovelluksissa, joissa koko sivun sisältö haetaan aina uudelleen.

SPA-sovellukset kommunikoivat palvelimien kanssa usein REST (*Representational*

State Transfer)-rajapinnan välityksellä. Tämä on vienyt palvelinohjelmointia yleiskäyttöisempään suuntaan, sillä REST-rajapinnan ja SPA-sovelluksen välinen sidonta on löyhä, ja yksi palvelin voi palvella useampaa SPA-sovellusta. Kehitys on tuonut markkinoille *Backend as a Service*- eli BaaS-palveluita, jotka tarjoavat yleiskäyttöisiä ratkaisuja esimerkiksi käyttäjien hallintaan sekä tiedon tallentamiseen. Jos nämä riittävät sovelluksen tarpeisiin, omaa palvelinta ei tarvita välttämättä lainkaan. BaaS-palveluihin vahvasti pohjautuvien sovelluksien voidaan sanoa noudattavan *serverless*-arkkitehtuuria (Roberts, 2016).

3.4 Toiminnallisuuksien jaottelu

Luvussa 3.2 todettiin datatuotteen teknisen toteutuksen koostuvan kolmesta erillisestä vaiheesta. Näistä ensimmäinen ja toinen vaihe sisälsivät pääasiallisesti datan keräämisen, puhdistamisen, käsittelyn sekä analysoinnin. Tyypillisesti nämä ovat kaikki operaatioita, joita ei ole mielekästä tai ylipäätään mahdollista tehdä käyttäjän selaimessa. Kolmas vaihe, käyttöliittymän toteuttaminen, puolestaan on loogisesti vaihe, joka halutaan luvun 3.3 mukaisesti toteuttaa ensisijaisesti selainpäässä. Tämä on esitetty kuvassa 3.3.



Kuva 3.3. *Datatuotteen toiminnallisuuksien jaottelu palvelin- ja selainpään välillä*

Kuvan 3.3 mukaisesti datatuotteen toiminnallisuuden ja siten myös arkkitehtuurin voidaan ajatella jakautuvan karkeasti jaoteltuna kahdeksi erilliseksi kokonaisuudeksi. Potentiaalisesti raskas datan käsittely sekä analysointi on ajettava palvelimella, jotta tarjolla on tarpeeksi laskentatehoa ja sopiva tietovarasto datan sekä tulosten ja välitulosten tallentamiseksi. Web-sovellusten käyttöliittymät halutaan kuitenkin ajaa pääasiallisesti selainpäässä, SPA-tyyppisenä sovelluksena. Siksi datatuotteen arkkitehtuuria on jakautuu karkeasti palvelin- ja selainpään arkkitehtuuriin.

Merkittävä haaste etenkin visuaaliseen analytiikkaan pohjautuvien datatuotteiden toteutuksessa voi kuitenkin olla visualisointien interaktiivisuuden varmistaminen, jos niitä on tarkoitus piirtää ainoastaan selaimessa. Yksinkertaisia visualisointeja, joissa datapisteitä tai verkoston solmuja on esimerkiksi satoja tai muutamia tuhansia, voidaan yleensä ladata helposti käyttäjän selaimeen ja muokata, kuten esimerkiksi suodattaa, reaaliajassa tai ainakin riittävän nopeasti käyttäjän interaktioiden pohjalta. Monimutkaisemmat visualisoinnit, joissa datapisteiden määrä kasvaa vaikkapa kymmeniin tai satoihin tuhansiin, eivät kuitenkaan vielä nykypäivän teknologioilla toimi sulavasti selainpäässä. On myös mahdollista, että lähdedataan pohjautuvaa sisältöä halutaan kustomoida käyttäjäkohtaisesti tai käyttäjän interaktioiden perusteella siten, että haluttuun lopputulokseen johtavan data-analyysiprosessin suorittaminen vie jopa palvelinpäässä pidempään kuin on käyttäjän kannalta kohtuullista. Karkeasti ottaen haasteisiin voidaan pureutua esimerkiksi generoimalla data-analyysin tuloksia palvelinpäässä jo ennakkoon ja esittämällä yhdessä näkymässä ainoastaan rajattu osajoukko käsiteltävästä datasta. Konkreettisempia tai yleiskäyttöisiä ratkaisuja näihin erittäin suurista datamääristä johtuviin ongelmiin ei kuitenkaan käsitellä tässä työssä sen laajuuden rajaamiseksi.

3.5 Mahdolliset arkkitehtuurimallit

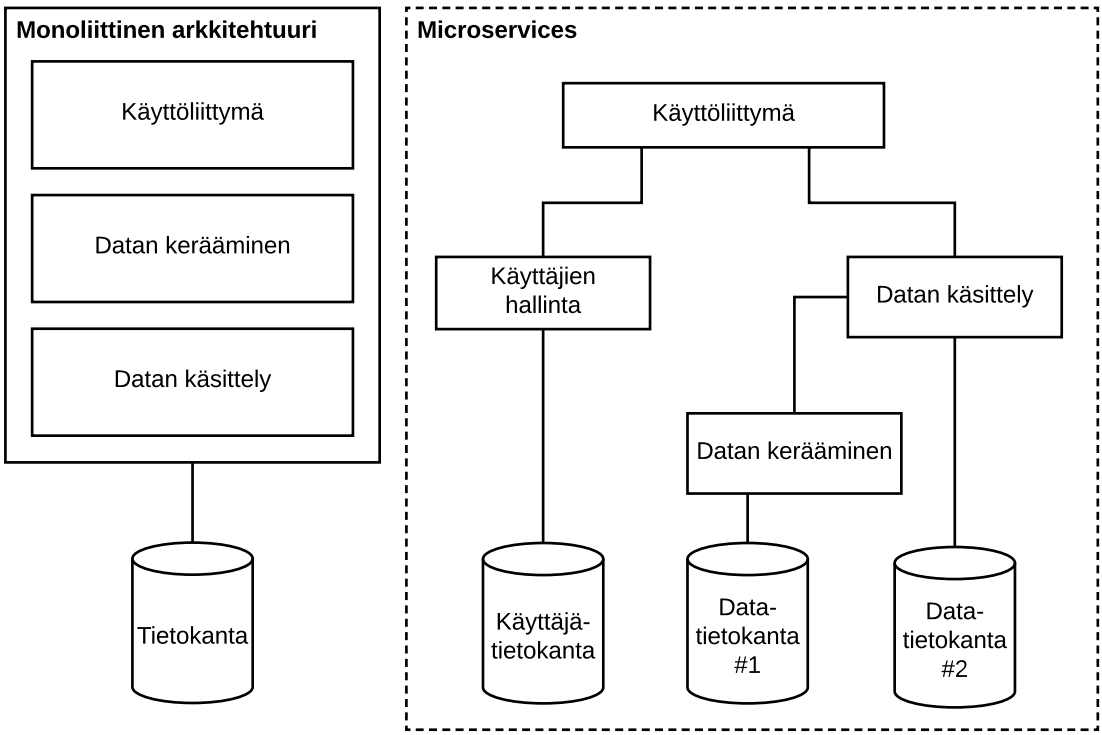
Luvussa 3.3 käsiteltyjen web-sovellusten aikakauden ensimmäisen ja toisen vaiheen aikaan käytetyt arkkitehtuurimallit eivät ole enää nykypäivänä relevantteja vaihtoehtoja, joten tässä työssä keskitytään ainoastaan nykyisiin eli kolmannen vaiheen arkkitehtuurimalleihin. Erilaisia web-arkkitehtuureja on tästä rajauksesta huolimatta olemassa paljon enemmän kuin mitä on mielekästä ottaa samaan vertailuun. Siksi tähän lukuun on pyritty valitsemaan yleiskäyttöisiä ja suosittuja arkkitehtuurimalleja, joita hyödynnetään myös nykypäivän web-kehityksessä. Vertailuun on otettu mukaan sekä selain- että palvelinpäähän sopivia arkkitehtuureja.

3.5.1 Microservices vs. monoliittinen arkkitehtuuri

Microservices on suhteellisen tuore arkkitehtuurimalli, jossa ohjelman toiminnallisuus on jaettu itsenäisiin, pienikokoisiin *mikropalveluihin*, joita ajetaan omissa prosesseissaan. Mikropalvelut kommunikoivat keskenään esimerkiksi HTTP-protokollan yli, ja niitä on siten helppo jakaa useammalle palvelimelle. (Namiot & Sneps-sneppe, 2014, s. 24). Yksittäisten palveluiden toteutustavat- tai teknologiat voidaan valita vapaasti siten, että kussakin palvelussa käytetään sen kannalta sopivimpaa ratkaisua (Villamizar et al., 2015, s. 585).

Monoliittinen arkkitehtuuri on microservicesille vastakkainen lähestymistapa. Sii-

nä ohjelman koko toiminnallisuus on toteutettu yksittäiseen järjestelmään. (Namiot & Sneps-sneppe, 2014, s. 24) Monoliittisen arkkitehtuurin ja microservices-mallin eroavaisuuksia on havainnollistettu kuvassa 3.4. Oleellisin ero mallien välillä on ohjelmistokomponenttien fyysinen jaottelu, joka on microservices-mallissa hajautettu usealle palvelimelle.



Kuva 3.4. *Monoliittisen ja microservices-arkkitehtuurin erot. Monoliittisessa arkkitehtuurissa sovelluksen koko toiminallisuus on toteutettu samaan järjestelmään, kun taas microservices-arkkitehtuurissa järjestelmä koostuu useasta mikropalvelusta.*

Järjestelmän koon kasvaessa monoliittinen arkkitehtuuri johtaa helposti ylläpidollisiin haasteisiin. Esimerkiksi palvelinresurssien skaalaaminen on vaikeaa: koska kaikki toiminnallisuus on sijoitettu yhteen järjestelmään, lisäresursseja ei voi kohdistaa ainoastaan siihen palvelun osaan, missä niitä tarvitaan. (Villamizar et al., 2015; Namiot & Sneps-sneppe, 2014) Etenkin suurten järjestelmien tapauksessa tämä voi lisätä palvelinkuluja. Myös päivitysten julkaiseminen on ongelmallista, koska pienikin muutos edellyttää koko järjestelmän uudelleenkäynnistämistä (Namiot & Sneps-sneppe, 2014, s. 24). Tästä voi seurata tarpeettomia käyttökatkoja.

Microservices-malli tarjoaa ratkaisun edellä kuvattuihin ongelmiin. Sitä käytettäessä järjestelmän skaalaaminen suuremmille käyttäjämäärille helpottuu, koska lisäresursseja voidaan kohdentaa ainoastaan suurimman kuorman alla oleville palveluille. Vastaavasti myös ohjelman päivittäminen helpottuu, sillä päivityksiä voidaan tehdä yksittäisiin palveluihin ja ottaa käyttöön muista palveluista välittämättä. (Namiot & Sneps-sneppe, 2014, s. 24) Se voi helpottaa myös useammasta kehittäjästä koostu-

vien tiimien työskentelyä, koska tällöin he voivat työskennellä eri palveluiden parissa toisistaan riippumatta.

Toisaalta myös moneen palaseen jaettuun arkkitehtuuriin liittyy omat haasteensa. Koska microservices-malli on luonteeltaan hajautettu järjestelmä, sen kompleksisuus monoliittiseen malliin verrattuna kasvaa. Myös sen testaaminen on vaikeampaa, koska testauksessa on huomioitava järjestelmän hajautuksesta seuraavat haasteet. (Namiot & Sneps-sneppe, 2014, s. 24) Näitä ovat esimerkiksi vaihteleva verkkoviive ja sen vaikutus järjestelmän toimintaan.

Monoliittinen arkkitehtuurimalli olla parempi vaihtoehto etenkin pienemmissä järjestelmissä, koska silloin ei tarvitse huolehtia palveluiden välisen kommunikaation toteuttamisesta. Kokonaisuus voi tällöin olla helpommin ymmärrettävissä. Tämän lisäksi monoliittisen järjestelmän käyttöönotto ja ylläpito on usein helpompaa palvelinten hallinnan näkökulmasta. Monoliittista järjestelmää voidaan ajaa yksittäisellä palvelimella, yhdessä ainoassa prosessissa. Microservices-mallissa hallittavia palvelimia on useampia (Namiot & Sneps-sneppe, 2014, s. 24).

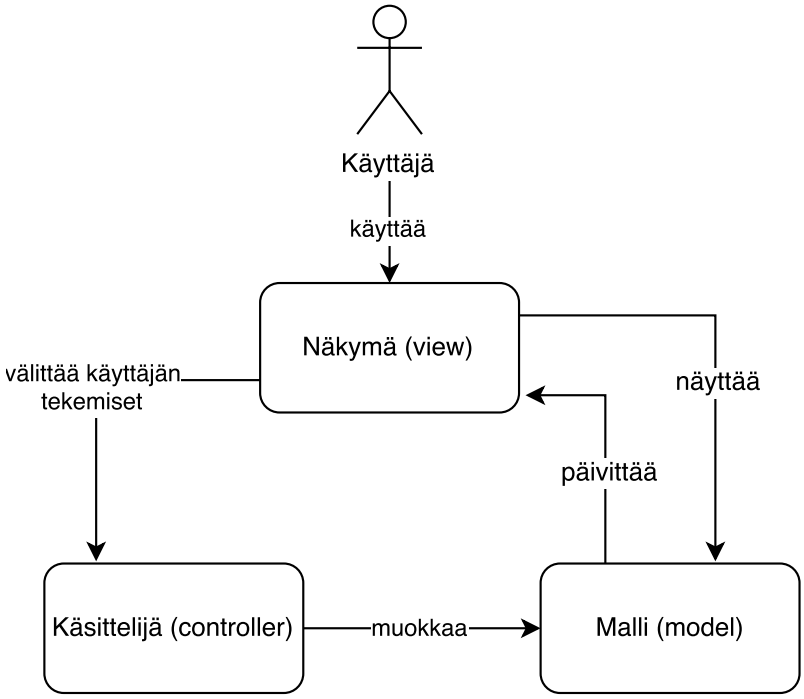
Järjestelmän kehittämisen alkuvaiheessa ei useinkaan ole täysin selvää, että millaisiin palveluihin järjestelmä kannattaa lopulta jakaa ja kuinka suuri järjestelmä tulee käytännössä olemaan. Siksi voi olla suositeltavaa aloittaa monoliittisesta arkkitehtuurista jaken sitä pienempiin järjestelmiin sitä mukaa kun kehitystyö etenee ja järjestelmästä aletaan havaitsemaan loogisesti erillisiä kokonaisuuksia.

Microservices-mallin toimivuuden puolesta puhuu sen suosio todella suurten webpalveluiden toteutuksissa. Microservices-toteutuksiin luottaa niin Google ja Facebook (Miller, 2016) kuin myös Amazon, Netflix ja LinkedIn (Villamizar et al., 2015, s. 583). Suuret yritykset hyödyntävät microservices-mallia myös suuressa mittakaavassa: esimerkiksi yksittäinen Google-haku kutsuu noin 70 erillistä mikropalvelua (Miller, 2016).

3.5.2 Model-View-Controller eli MVC-malli

Model-View-Controller eli MVC on perinteinen ja yleisesti tunnettu ohjelmistoarkkitehtuurimalli, jossa järjestelmä on jaettu kolmeen osaan: malliin (model), näkymään (view) ja käsittelijään (controller). Toisin sanoen datan esittäminen, käsittely ja muokkaaminen on eroteltu toisistaan. Mallin tavoitteena on ohjelman rakenteen selkeyttäminen sekä ylläpidettävyyden parantaminen. MVC:n mukainen jaottelun tarkoitus on mahdollistaa esimerkiksi ohjelmiston käyttöliittymän päivittäminen muuttamalla näkymäkoodia ilman, että dataa käsittelevään mallikomponenttiin tarvitsee koskea. (Worrall & Chaussalet, 2011, s. 1) Nimensä mukaisesti MVC on

tarkoitettu lähinnä käyttöliittymien toteuttamiseen. MVC-arkkitehtuurin perusrakenne on esitetty kuvassa 3.5.



Kuva 3.5. *Perinteinen esitystapa MVC-arkkitehtuurista*

MVC-jaottelu toimii suhteellisen selkeästi perinteisissä työpöytäsovelluksissa, koska silloin sen kaikki komponentit toimivat paikallisesti samalla laitteella. Sen sijaan web-sovelluksissa MVC on enemmän tulkinnanvarainen kokonaisuus.

Web-pohjaisissa ratkaisuissa selvää on lähinnä, että näkymää ajetaan lopulta loppukäyttäjän päätelaitteelta, mutta malli ja/tai käsittelijä voivat sijaita myös palvelimella. Sovelluslogiikka täytyy siis *jaotella* palvelimen ja päätelaitteen välille, ja jaottelu on käytännössä tehtävä jo sovelluksen suunnitteluvaiheessa. Käytännössä onnistuneen jaottelun tekeminen projektin alkuvaiheessa on kuitenkin usein vaikeaa, tai jopa mahdotonta, koska siihen vaikuttavat mm. projektin vaatimukset (jotka usein muuttuvat tai tarkentuvat projektin kuluessa) sekä ympäristötekijät (verkkoviive, päätelaitteiden ja palvelimien suorituskyky). Tästä johtuen MVC-arkkitehtuurin käyttäminen jaottelun kannalta ei-triviaaleissa tapauksissa (kuten esimerkiksi web-sovelluksissa) on usein haastava tehtävä. (Leff & Rayfield, 2001, s.119)

Muun muassa näistä syistä johtuen perinteisen, kuvan 3.5 mukaisen MVC-arkkitehtuurin käyttäminen web-kehityksessä ei ole etenkään nykyään suosiossa. Esimerkiksi yleisesti käytetyt interaktiiviset SPA-sovellukset tarvitsevat usein oman käsittelijän (controller) sekä mallin (model) jo selainpäässä. Toisaalta, jos sovellus haluaa tallentaa dataa myös muualle kuin käyttäjän päätelaitteelle, myös palvelinpäähän tarvitaan malli sekä mahdollisesti myös käsittelijä. MVC-arkkitehtuurimallia ei siis

välttämättä kannata tai edes voi käyttää sellaisenaan modernissa web-kehityksessä. Siinä esitelty periaate käyttöliittymän, sovelluslogiikan sekä tiedon tallentamisen erottelemisesta on kuitenkin hyvä lähtökohta useimpien sovelluksien arkkitehtuuriin. Sen oppeja voi soveltaa myös tilanteissa, joissa perinteinen ja tiukasti MVC:n mukainen toteutus ei ole mielekäs tai edes mahdollinen ratkaisu.

3.5.3 Reaktiivinen ohjelmointi

Reaktiivisella ohjelmoinnilla tarkoitetaan ohjelmointiparadigmaa, jossa sovelluksen komponenttien käyttämä data sidotaan toisiin komponentteihin siten, että datan päivittyminen yhdessä paikassa saa myös muut siihen sidotut komponentit päivittymään (Patel, 2016). Tällöin esimerkiksi datalähteen päivittyminen voi aktivoida automaattisesti myös näkymän päivittymisen.

Reaktiivisella ohjelmoinnilla voidaan helpottaa etenkin käyttöliittymien toteuttamista. Esimerkiksi MVC-henkisen sovelluksen toteuttaminen suoraviivaistuu, kun näkymä (view) saadaan sidottua suoraan malliin (model), jolloin käsittelijän (controller) tekemät mallin muokkaukset päivittyvät automaattisesti näkymään.

3.5.4 Komponenttipohjainen arkkitehtuurimalli

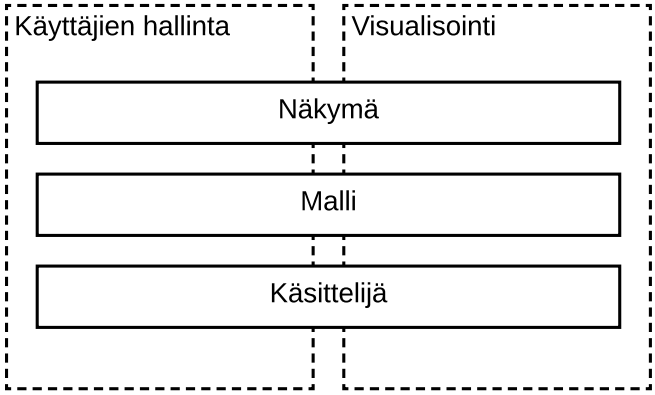
Komponenttipohjainen ohjelmistokehitys on nykyisin melko suosittu termi, jolla on kuitenkin useampia merkityksiä. Yhden määritelmän mukaan kyseessä on malli, jossa ideana on rakentaa uusia sovelluksia yhdistämällä olemassa olevia ohjelmistokomponentteja yhdeksi kokonaisuudeksi (Aoyama & others, 1998, s. 1). Sen tavoitteena on myös vastaavasti päätyä arkkitehtuuriin, joka mahdollistaa syntyneiden komponenttien uudelleenkäytön (Aoyama & others, 1998, s. 1).

Toisen määritelmän mukaan kyseessä on arkkitehtuurityyli, jossa (käyttöliittymä) toteutetaan pieninä, itsenäisinä ja autonomisina komponentteina, joilla on selkeät vastuualueet ja joita on helppo uudelleenkäyttää. Malliesimerkkinä komponenttipohjaiseen arkkitehtuuriin ohjaavasta kirjastosta voidaan pitää esimerkiksi Facebookin kehittämää React-sovelluskehystä. Komponenttipohjaisessa mallissa ideana on sisällyttää kaikki samaan ominaisuuteen liittyvä sovelluslogiikka sekä käyttöliittymä samaan komponenttiin. Mallia noudattamalla komponenteista tulee luonnostaan toisistaan riippumattomia, joka helpottaa niiden uudelleenkäyttöä. Komponenttipohjaisuus selkeyttää ohjelman rakennetta ja helpottaa ylläpitoa, koska tällöin kaikki samaan toiminnallisuuteen liittyvät ominaisuudet ovat yhdessä. (Shapiro, 2016) Web-sovelluskehysten, kuten esimerkiksi Reactin tapauksessa yksittäiset komponentit muodostavat käytännössä HTML:n DOM (*Document Object Model*)-puun palasia.

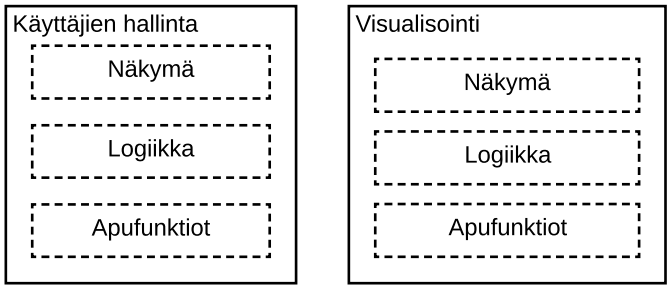
Ensimmäisenä mainittu määritelmä on niin laaja, että se kattaa kaikki sovellukset, jotka hyödyntävät ulkoisia kirjastoja. Etenkin web-kehityksessä ulkoisten kirjastojen hyödyntäminen on niin yleistä, että lähes minkä tahansa toiminnallisuutta sisältävän sivun voisi tulkita tällöin olevan komponenttipohjainen. Jälkimmäinen määritelmä puolestaan on huomattavasti rajaavampi, ja tässä työssä komponenttipohjaisella arkkitehtuurilla tarkoitetaan jatkossa ainoastaan sen mukaisia toteutuksia.

3.5.5 Komponenttipohjainen arkkitehtuuri vs. MVC

Komponenttipohjaisen arkkitehtuurimallin ja MVC:n suurin ero on niiden tavassa jakaa yksiköiden vastuualueet: MVC:ssä jako tehdään *horisontaalisesti*, komponenttipohjaisessa mallissa *vertikaalisesti* (Shapiro, 2016). Tätä havainnollistetaan kuvassa 3.6.



(a) *Esimerkiksi MVC-arkkitehtuurin käyttämässä vastuualueiden horisontaalisessa jaottelussa yksittäinen toiminto koostuu mallin, näkymän ja käsittelijän yhdistelmästä. Etenkin malli ja käsittelijä voivat olla jaettuja usean näkymän kesken.*



(b) *Esimerkiksi komponenttipohjaisen arkkitehtuurin käyttämässä vertikalisessa jaottelussa yksittäinen komponentti sisältää itsessään kaikki näkymän tarvitsemat palvelut.*

Kuva 3.6. *Vastuaalueiden horisontaalinen ja vertikaalinen jaottelu*

Horisontaalisessa jaottelussa yksittäinen toiminto koostuu usean löyhästi sidotun

kerroksen yhdistelmästä. Esimerkiksi MVC-mallin mukaista arkkitehtuuria pidetään horisontaalisena (Shapiro, 2016). Sen mukaisessa toteutuksessa (kuva 3.6a) yksittäiseen näkymään liityvä toiminnallisuus jakautuu sovelluksessa kolmeen kerrokseen: malliin, näkymään ja käsittelijään. Kerrokset ovat itsenäisiä ja toisitaan riippumattomia, eivätkä niiden palvelut ole välttämättä tarkoitettu vain yksittäisen näkymän käyttöön. Vertikaaliseksi laskettavan komponenttipohjaisen arkkitehtuurin tapauksessa (kuva 3.6b) kaikki yhteen näkymään liittyvä toiminnallisuus on puolestaan paketoitu samaan komponenttiin. Tällöin ne on tarkoitettu ainoastaan kyseisen näkymän käyttöön, eikä niitä voi hyödyntää kyseisen komponentin ulkopuolella.

Komponenttipohjainen malli rikkoo MVC:n mukaista periaatetta arkkitehtuurin eri osuuksien vastuualueiden jakamisesta, koska sen koko idea on pakata kaikki komponenttiin liittyvä toiminnallisuus samaan paikkaan. Tämä johtaa helposti näkymäkerroksen paisumiseen. (Shapiro, 2016) Toisaalta myös komponenttipohjaisessa mallissa halutaan pitää eri komponenttien vastuualueet mahdollisimman tiukasti erillään, mutta jaottelu on filosofisesti selvästi erilainen kuin MVC:n näkymämalli-käsittelijä-jako. React[3]-sovelluskehyksen julkaisun myötä komponenttipohjainen arkkitehtuurimalli on tullut erittäin suosituksi vaihtoehdoksi web-sovelluskehityksessä. Sovelluskehysten suosiota on vertailtu tarkemmin luvussa 4.1.

Komponenttipohjaisuus ohjaa käyttöliittymäkoodin jakamiseen kohtuullisen kokosiin paloihin eli komponentteihin. Se auttaa siten välttämään valtavien ja vaikeasti ylläpidettävien yksiköiden syntymistä. Komponenttien yleisesti ottaen helppo uudelleenkäytettävyys voidaan myös laskea mallin eduksi. Toisaalta myös MVC-arkkitehtuurilla on mahdollista luoda uudelleenkäytettävää koodia, joten malleja ei voida asettaa yksiselitteiseen paremmuusjärjestykseen.

3.6 Web-pohjaisen datatuotteen arkkitehtuuri

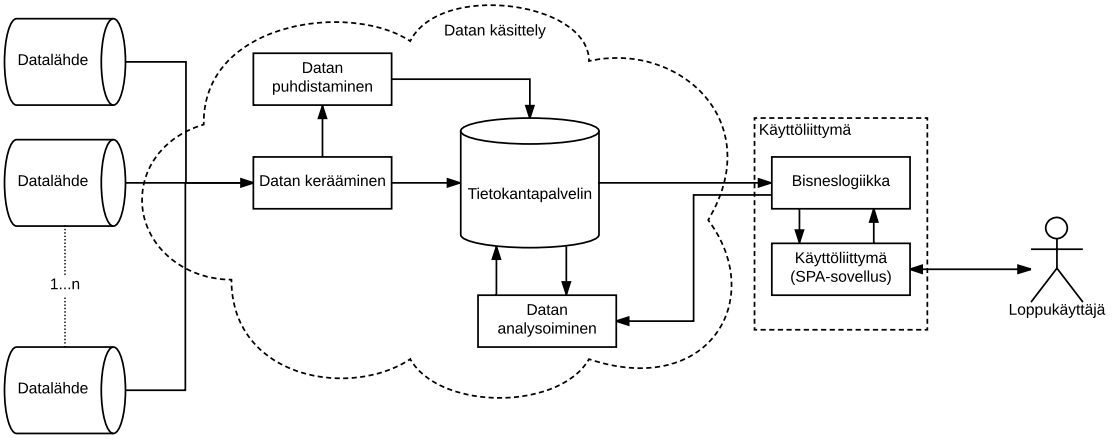
Kuten kuvassa 3.3 esitettiin, web-pohjaisen datatuotteen toimintalogiikka voidaan jakaa karkeasti palvelin- ja selainpään logiikkaan. Tarkempaa jaottelua on mahdollista tehdä esimerkiksi luvussa 3.2 esitettyjen, CRISP-DM ja Ostinato-malleihin pohjautuvien analytiikkaprosessin vaiheiden perusteella. Koska data-analytiikkaprosessi voidaan jakaa selvästi erillisiksi vaiheiksi, löyhästi sidottu microservices-arkkitehtuuri on luonteva valinta sen toteuttamiseksi. Jako mikropalveluihin on aina tapauskohtaista. Lähtökohtana voidaan kuitenkin pitää data-analytiikaprosessin vaiheita: *datan kerääminen* on oma palvelunsa, mahdollinen *datan puhdistaminen* omansa, *datan käsittely* omansa sekä *datan tallentaminen* omansa. Ehdotettu jaottelu sopii

[3]https://facebook.github.io/react/

Ostinato-mallin henkeen, sillä sekä microservices että Ostinato painottavat yksittäisten palveluiden, eli tässä tapauksessa mikropalveluiden löyhän sidonnan ja toisistaan riippumattomuuden tärkeyttä. Microservices-toteutuksessa yksittäisiä vaiheita voidaan kehittää toisistaan riippumatta. Tämä palvelee myös Ostinato-mallin tavoittelemaa prosessin manuaalisten vaiheiden mahdollistamista: löyhästi sidottuja palveluita voi kutsua myös manuaalisesti sekä tarvittaessa korvata yksittäisiä palveluita toisilla.

Luvussa 3.5.2 käsitellyn MVC-mallin henkisesti myös käyttöliittymäosuus halutaan jakaa, jotta sitä ei tarvitse sitoa tiukasti datan käsittelyyn tai tallentamiseen. Kyseisessä luvussa mainituista syistä johtuen tarkasti MVC:n mukainen toteutus ei kuitenkaan välttämättä ole mielekäs. Tyypillisesti käyttöliittymä jaetaan ainakin kahteen osaan: palvelin- ja selaintoteutukseen. Palvelimelle toteutetaan niin sanottu bisneslogiikka, joka voi sisältää esimerkiksi käyttäjien hallinnan tai maksupalveluintegraation. Selainpäässä ajettava käyttöliitymä puolestaan kannattaa toteuttaa luvun 3.3 mukaisesti yhden sivun SPA-sovelluksena. Tällöin varsinainen käyttöliittymälogiikka voidaan ajaa käyttäjän selaimessa, joka tyypillisesti parantaa vasteaikoja ja tarjoaa siten sulavamman käyttökokemuksen. Jos palvelinpään vastuulla on useita toisistaan riippumattomia vastuualueita, sen voi jakaa useampiin osiin.

Ehdotuksen mukainen arkkitehtuuri on esitetty kuvassa 3.7. Kuvassa katkoviivojen sisältä löytyvät yksiköt edustavat kokonaisuuksia, sopivat omiksi mikropalveluikseen. Ehdotuksen mukaisen arkkitehtuurin on tarkoitus toimia lähtökohtana useimpiin käytännön toteutuksiin, mutta tapauskohtainen mukauttaminen on suositeltavaa. Esimerkiksi palvelinpään mikropalvelut voidaan toteuttaa aluksi monoliittiseen palveluun, jota voidaan jakaa pienempiin mikropalveluihin sovelluksen laajuuden kasvaessa.



Kuva 3.7. *Ehdotus web-teknologioilla toteutetun datatuotteen arkkitehtuurista. Yksittäiset laatikot sekä tietokantapalvelin kuvaavat omia palveluitaan. Nuolet kuvaavat datavirran suuntaa palvelusta toiseen.*

Varsinaisen käyttöliittymän toteuttamisessa luvussa 3.5.4 esitelty komponenttipohjainen arkkitehtuurimalli olisi perusteltu ratkaisu, sillä se helpottaisi muun muassa luvussa 3.1 toivottua koodin uudelleenkäyttöä. Käyttöliittymän toteuttamisessa voisi hyödyntää myös luvussa 3.5.3 käsiteltyä reaktiivista ohjelmointia, jolla voidaan helpottaa automaattisesti sisältöään päivittävien näkymien toteuttamista.

4. CASE-TUTKIMUKSET

Datatuotteisiin perehdyttiin työssä käytetyn ADR-tutkimusmenetelmän (luku 1.1) mukaisesti toteuttamalla kahden datatuotteen prototyypit, jotka esitellään tässä luvussa omina case-tutkimuksinaan. Ensimmäisessä case-tutkimuksessa toteutettiin sosiaalisen suosittelulistan käyttöliittymä ja sovelluslogiikka *Cobweb*-projektin tarpeisiin. Tämä tutkimus on arkkitehtuurin kannalta laajempi kokonaisuus, ja toimii siten kokeena luvussa 3 suunnitellun web-pohjaisen datatuotteen arkkitehtuurin käytännön toteutuksesta.

Toisessa case-tutkimuksessa keskityttiin luvussa 2.2 käsiteltyyn visuaaliseen verkostoanalyysiin ja luotiin verkosto Twitteristä kerätyn datan pohjalta. Tutkimus on myös jatkoa datatuotteen arkkitehtuuriin tutustumiselle – kuten luvussa 2.1 todettiin, datatuotteen tulisi olla vuorovaikutteinen järjestelmä, mutta vuorovaikutteisuus jäi ensimmäisessä case-tutkimuksessa sivuosaan. Case 2 paikkaa tilannetta sen yhteydessä toteutetun interaktiivisen visualisoinnin ansiosta. Työn johdannossa (luku 1.2) todettiin, että tässä työssä ollaan kiinnostuneita sosiaalisesta datasta. Siksi tässä yhteydessä käsitellään myös sosiaalisen median datan keräämiseen liittyviä haasteita ja huomioita Twitterin kontekstissa.

4.1 Case 1: Cobweb/Suosittelulista

Tässä luvussa kuvataan *Cobweb*-projektin yhteydessä toteutettua prototyyppiä järjestelmästä, jonka tarkoituksena oli suositella käyttäjilleen ammatillisesti hyödyllisiä kontakteja, joita henkilö ei entuudestaan tunne. Protytyypin toteuttaminen palveli ADR-tutkimusmenetelmän toista vaihetta, sillä sen avulla voitiin kokeilla aiemmin ehdotettua web-pohjaisen datatuotteen arkkitehtuuria käytännössä.

Suositusten lähdedatana käytettiin dblp-palvelusta ladattua aineistoa. Dblp on avoimesti saatavilla oleva kirjallisuustietokanta, jonka omistaa saksalainen Trierin yliopisto. Palvelusta ladattu aineisto sisälsi XML-muotoon tallennettua metadataa miljoonista akateemisista julkaisuista[1]. Se on lajissaan harvinaisuus; vastaavia aineistoja samassa laajuudessa ei ole avoimesti saatavilla.

[1] `http://dblp.uni-trier.de/faq/What+do+I+find+in+dblp+xml`

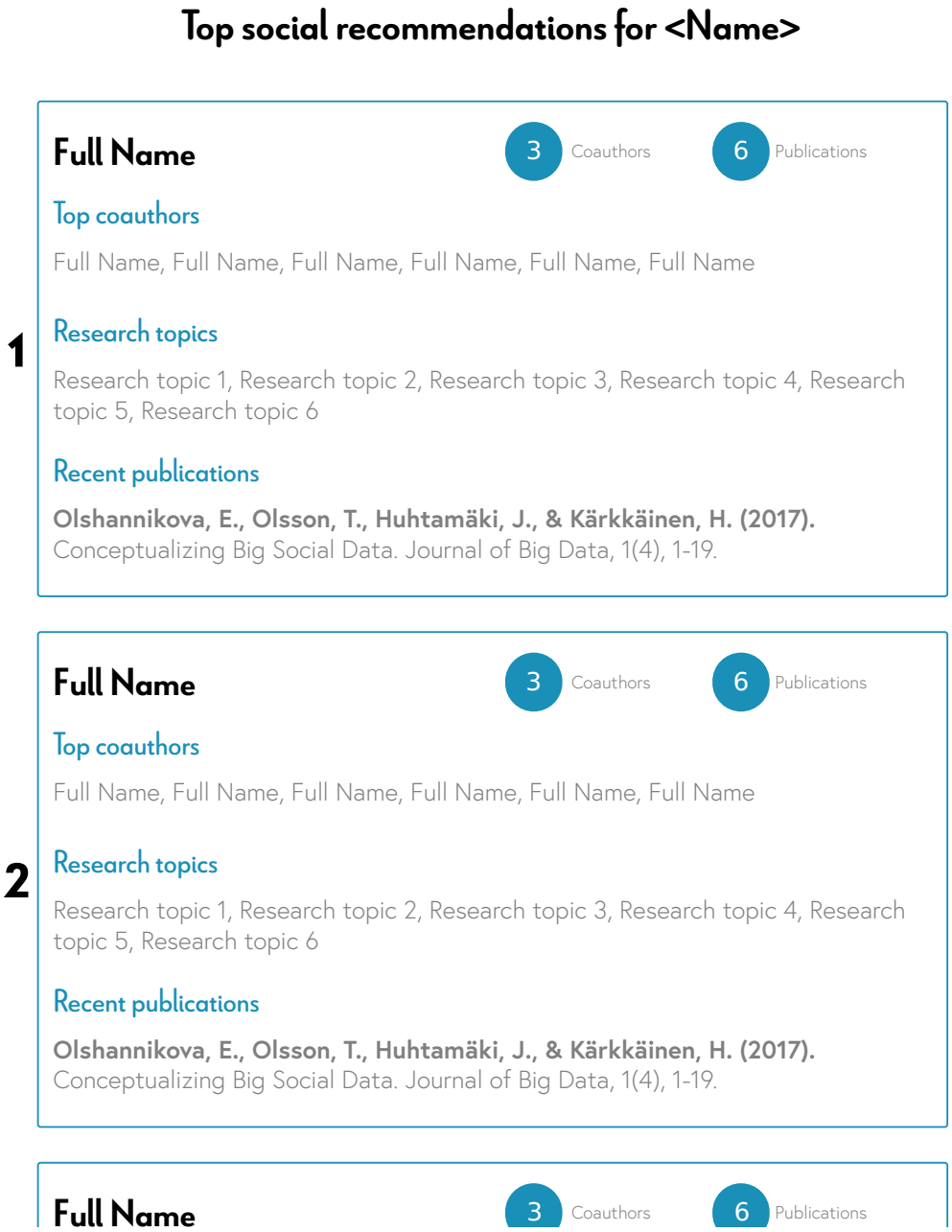
Suosittelujärjestelmän logiikan kannalta aineiston oleellisinta metadataa olivat julkaisun otsikko, kirjoittajat, julkaisupaikka sekä julkaisuaika. Järjestelmä pyrki löytämään niiden pohjalta henkilöitä, jotka ovat linkittyneet kohdehenkilöön nykyisten yhteistyökumppaineiden tai tutkimusaiheiden kautta ja jotka voisivat siten olla hänen kannaltaan relevantteja kontakteja. Jotta suosituksia voitiin esittää myös loppukäyttäjille ja arvioida niiden hyödyllisyyttä heiltä kerätyn palautteen perusteella, niiden esittämiseksi toteutettiin web-pohjainen käyttöliittymä. Järjestelmä sai nimekseen *MatchUs*.

Ensimmäinen suunnitelma käyttöliittymän ulkoasusta on esitetty kuvassa 4.1. Käyttöliittymäsuunnittelu saatiin *Cobweb*-projektilta. Ensimmäinen versio käyttöliittymästä toteutettiin melko tarkkaan tämän suunnitelman mukaisesti. Käyttöliittymää kuitenkin paranneltiin iteratiivisesti projektin aikana, sillä vaatimukset ja toiveet tarkentuivat matkan varrella. Esimerkiksi listamuotoinen esitystapa vaihdettiin karuselliin, jotta suositukset eivät näyttäisi olevan paremmuusjärjestyksessä ja vaikuttaisi siten käyttäjän ennakko-odotuksiin. Kuvaruutukaappaus lopullisesta käyttöliittymästä on esitetty kuvassa 4.2.

Käyttöliittymää koskevat vaatimukset tulivat *Cobweb*-projektilta. Tärkein vaatimus oli web-pohjaisus: käyttöliittymän tuli toimia ajantasaisilla web-selaimilla. Suositukset tuli tallentaa tietokantaan, jotta vältyttäisiin käyttöliittymän sitomisesta vain tiettyyn sisältöön. Ne oli myös voitava rajata ainoastaan asianomaisten saataville, eli esimerkiksi sidottava käyttäjätileihin. Projektilla ei ollut omaa www-palvelinta sivuston hostaukseen, joten myös sopivan hosting-palvelun valinta kuului vaatimuksiin. Järjestelmän prototyyppi haluttiin saada testattavaksi mahdollisimman nopeasti, koska työaikaa sen toteuttamiseksi oli käytössä vain rajallisesti.

Minimiratkaisun käyttäjien hallintaan, tietokannan tarjoamiseen sekä sivuston hostaukseen tarjosi Googlen omistama Firebase[2]-pilvipalvelu. Firebase on web-pohjainen sovelluskehitysalusta, joka on suunnattu perustoimintojen tarjoamiseen etenkin web- ja mobiilisovelluksille (iOS/Android). Sen tarjonnasta löytyy tietokannan, käyttäjien hallinnan ja hosting-palveluiden lisäksi myös muun muassa analytiikkaratkaisuja, mainostuspalveluja sekä push-notifikaatio ja pikaviestirajapinnat - eli kyseessä on eräänlainen BaaS-alusta. Suurin osa Firebasen tarjoamista palveluista on maksuttomia. *MatchUs*-järjestelmän kannalta hyödyllisimpiä ominaisuuksia olivat reaaliaikainen tietokanta (realtime database), käyttäjien hallinta ja autentikointi, sivuston hostaus sekä aiempien käyttöliittymäjulkaisujen versiointi. Firebase tarjosi myös valmiin, HTML- ja JavaScript-pohjaisen ohjelmistokomponentin rekisteröitymisen ja sisäänkirjautumisen integroimiseksi olemassa olevalle web-sivustolle (Firebase UI) sekä web-pohjaisen käyttöliittymän (Firebase console) käyttäjien hallintaan

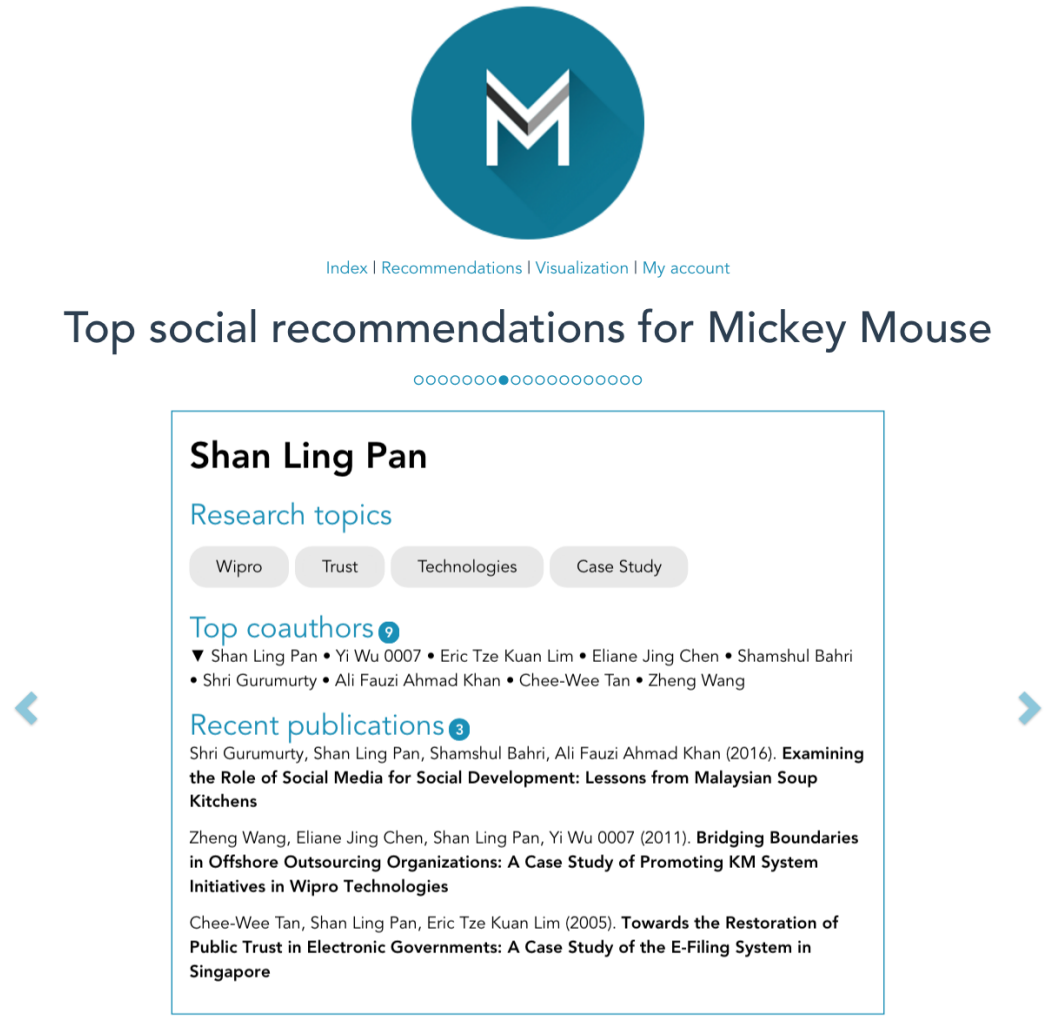
[2] `https://firebase.google.com`



Kuva 4.1. *Alkuperäinen suunnitelma MatchUs-järjestelmän käyttöliittymästä. Henkilösuositukset on esitetty käyttöliittymässä allekkain listamuodossa.*

sekä tietokannan sisällön selaamiseen ja muokkaamiseen. *MatchUs*-järjestelmän arkkitehtuurin yleiskuva on esitetty kuvassa 4.3.

Kuvan 4.3 mukaisesti loppukäyttäjä on vuorovaikutuksessa ainoastaan käyttöliittymän (frontend) kanssa, joka taas käyttää Firebasen tarjoamia palveluita. Toteutus on jaettu palveluihin luvussa 3.2 esitettyjen datatuotteen logiikan kannalta erillisten vaiheiden eli kokonaisuuksien mukaan. Palvelun käyttämän dblp- eli lähdedatan puhdistaminen ja analysoiminen toteutettiin omiin, Pythonilla kirjoitettuihin palveluihin. Palvelulle annettiin syötteenä halutun tutkijan nimi, ja ulostulona saatiin

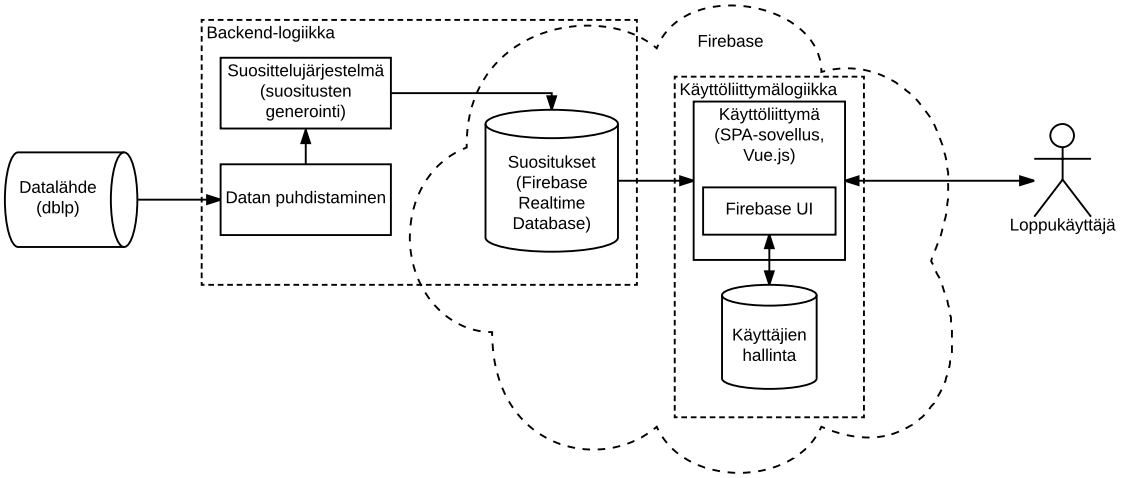


Kuva 4.2. *Kuvaruutukaappaus MatchUs-järjestelmän toteutetusta käyttöliittymästä. Henkilösuositukset on esitetty karusellissa, ja niiden yhteydessä esitetään dblp-datasta saatuja perustietoja suositellusta henkilöstä. Käyttöliittymää käytettiin Cobweb-projektissa suositusten esittämiseen loppukäyttäjille.*

JSON-muotoinen lista kyseiselle tutkijalle suositeltavista kontakteista. Suositukset tallennettiin käyttäjäkohtaisesti Firebasen Realtime Databaseen, ja haettiin sieltä käyttäjän selaimessa ajettavaan käyttöliittymään. Sovellukseen rekisteröityminen, kirjautuminen ja käyttäjien hallinta onnistui lähes yksinomaan Firebasen tarjoamia valmiita palveluita hyödyntämällä. Loogiselta jaottelultaan toteutus vastasi mikropalveluarkkitehtuuria, mutta palveluiden välinen kommunikaatio nojasi prototyypissä suurimmaksi osaksi manuaaliseen työhön.

MatchUs-järjestelmän käyttöliittymä toteutettiin JavaScriptiin perustuvalla Vue.js[3]-

[3] `https://vuejs.org`



Kuva 4.3. *MatchUs-järjestelmän arkkitehtuurin yleiskuva. Yhtenäisten viivojen ympyröimät yksiköt kuvaavat yksittäisiä kokonaisuuksia tai palveluita ja nuolet kuvaavat datan kulkua niiden välillä. Katkoviivoilla ympyröidyt laatikot kuvaavat ominaisuuden jaottelua palvelin- ja käyttöliittymäpään välillä kuvan 3.3 mukaisesti. Pilven muotoisella katkoviivalla ympyröity alue sisältää kaikki Firebasen avulla toteutetut toiminnallisuudet.*

sovelluskehyksellä. Vue.js on ilmainen ja avoimeen lähdekoodiin perustuva sovelluskehys, jota voidaan käyttää SPA-tyyppisten web-sovellusten luomiseen. Vue.js ohjaa komponenttipohjaiseen arkkitehtuurimalliin sekä reaktiiviseen ohjelmointiin ja vastaa siltä osin luvussa 3.6 tehtyä ehdotusta.

Vuen lisäksi käyttöliittymän sovelluskehykseksi harkittiin myös muita suosittuja ja yleisesti käytettyjä JavaScript-sovelluskehyksiä. GitHubissa[4] käyttäjien antamien tähtien mukaan järjestettynä vuoden 2017 alussa suosituimmat sovelluskehykset olivat taulukon 4.1 mukaisesti ReactJS, AngularJS sekä Vue.js. Näistä AngularJS voitiin laskea jo vanhentuneeksi, koska sitä seuraava Angular 2 -kehys oli julkaistu jo vuonna 2016.

Työn tekijällä oli jo aiemmin kokemusta ReactJS- ja AngularJS-kehyksistä. Aiempiin kokemusten pohjalta arvioiden *Cobwebin* tarpeisiin riittävän sovelluksen olisi varmasti voinut toteuttaa näistä kummalla tahansa. Tässä yhteydessä haluttiin kuitenkin myös hankkia kokemusta uudesta teknologiasta, ja ReactJS:n tavoin reaktiivinen ja komponenttipohjainen Vue.js vaikutti mielenkiintoiselta vaihtoehdolta.

ReactJS:iin verrattuna Vuen suurin heikkous oli sen suosion vähäisyys. Reactin päivittäiset latausmäärät ovat Vueen verrattuna yli seitsenkertaiset[5]. Tästä johtuen verkosta löytyy huomattavasti enemmän Reactiin liittyvää keskustelua, aineistoa

[4] `https://github.com`

[5] `https://npmcompare.com/compare/react,vue` 15.10.2017 tarkasteltuna Reactilla oli 101,757 päivittäistä latausta ja Vuella 14,157.

Taulukko 4.1. *JavaScript-sovelluskehysten ja -kirjastojen vertailua, pohjautuen* 5 Best JavaScript Frameworks in 2017 *(2017)*

Kehys	Tyyppi	Julkaistu	Kotisivu	Githubissa kehittäjiä	Githubissa tähtiä
AngularJS	MVW sovelluskehys	2009	`angularjs.org`	1 562	54 402
Angular 2	MVC sovelluskehys	2016	`angular.io`	392	19 832
ReactJS	JavaScript-kirjasto	2013	`reactjs.net`	912	57 878
Vue.js	MVC sovelluskehys	2014	`vuejs.org`	62	39 933
Ember.js	MVC sovelluskehys	2011	`emberjs.com`	636	17 420
Meteor.js	JavaScript-sovellusalusta	2012	`meteor.com`	328	36 496

sekä valmiita kirjastoja. Vuen kanssa on siis olemassa suurempi riski, että ongelmiin ei löydy valmiita ratkaisuja. Vuelle suunttattua materiaalia oli kuitenkin tarjolla verkossa riittävästi projektin tarpeisiin.

4.2 Case 2: Twitter-verkostoanalyysi

Toisena case-työnä toteutettiin Twitter-verkostoanalyysi. Verkostoanalyysin toteuttaminen jakautui luvussa 3.2 esitettyjen datatuotteen kehittämisen vaiheiden mukaisesti kolmeen kokonaisuuteen: työ alkoi datan keräämisellä ja puhdistamisella, jonka jälkeen sen pohjalta generoitiin verkosto, joka lopulta visualisoitiin interaktiivisena tutkimusta varten. Tässä luvussa esitellään edellä mainitut vaiheet sekä niihin liittyviä haasteita ja huomioita.

Twitter[6] on sosiaalinen mikroblogipalvelu, jonne käyttäjät voivat lähettää maksimissaan 140 merkkiä pitkiä viestejä.[7] Viesteihin on mahdollista liittää aihetunnisteita, jotka merkitään hashtagilla eli risuaitamerkillä `#`. Aihetunnisteen perusteella on mahdollista hakea samalla tunnisteella merkittyjä viestejä. Viesteissä on myös mahdollista viitata muihin käyttäjiin liittämällä halutun käyttäjänimen eteen `@`-merkin.

Twitter-datan käyttäminen sosiaalisten verkostojen pohjana on suhteellisen helppoa ja siksi sitä käytetään usein aiheeseen liittyvissä tutkimuksissa. Syitä Twitter-datan helppokäyttöisyyteen on useita. Ensinnäkin Twitter tarjoaa datan hakemiseen valmiin REST-rajapinnan[8], jonka avulla etenkin pienten datamäärien hakeminen on

[6] `https://twitter.com`

[7] Raja nostettiin osalle käyttäjistä 240 merkkiin tutkimuksen suorittamisen jälkeen. `https://blog.twitter.com/official/en_us/topics/product/2017/Giving-you-more-characters-to-express-yourself.html`

[8] `https://dev.twitter.com/rest/public/`

yksinkertaista. Rajapinta tarjoaa myös huomattavan määrän yksittäisiin viesteihin eli twiitteihin liittyvää metadataa. Verkostoanalyysien kannalta hyödyllistä on etenkin jokaiseen twiittiin liitetty kirjoittajan nimi sekä luettelo twiitissä mainituista aihetunnisteista ja käyttäjistä. Data on valmiiksi parsittua sekä JSON-muotoista, eikä sitä tarvitse puhdistaa käsin.

Tämän lisäksi verkostojen tekeminen Twitter-datasta on helppoa, koska käyttäjien väliset viittaukset ovat yksiselitteisiä – tai ainakin sellaisiksi oletettuja. Käytännössä tämä tarkoittaa esimerkiksi sitä, että `@mikko`-nimiselle käyttäjälle suunnatut viestit sisältävät aina `@mikko`-viittauksen tai muuten niitä ei tulkita hänelle suunnatuiksi. Vastaava oletus tehdään yleensä myös aihetunnisteisiin liittyen eli esimerkiksi `#tampere3`-tunnisteella Tampere3-hankkeesta käytyyn keskusteluun lasketaan mukaan vain `#tampere3`-tunnisteen sisältämät viestit. Näillä oletuksilla verkostojen kuvaaminen ja analysoiminen ohjelmallisesti on erittäin suoraviivaista.

Toisaalta esimerkiksi Tampere3-hankkeeseen liittyvää keskustelua tutkittaessa ainoastaan `#tampere3`-tunnisteeseen keskittyminen ei todennäköisesti sisällä kaikkea aiheeseen liittyvää keskustelua. Tilannetta voidaan parantaa hakemalla aineistoon mukaan myös esimerkiksi `#tre3` tai `#uusiyliopisto`-tunnisteilla merkityt viestit. Täydellisen kattavuuden saavuttaminen ei käytännössä ole mahdollista, koska keskusteluun voi sisältyä myös viestejä, joista aihetunniste puuttuu.

Twitterin REST-rajapinnassa on kuitenkin rajoituksia, jotka heikentävät sen käyttökelpoisuutta. Sallittujen HTTP-kutsujen määrä riippuu käytetystä resurssista, mutta pääsääntöisesti se on 15 tai 180 kutsua jokaista 15 minuutin aikaikkunaa kohti. Tämän lisäksi pyyntöjen kokoa on rajoitettu: esimerkiksi yhdellä pyynnöllä haettavien twiittien määrä voi olla maksimissaan 100 kappaletta. Rajapinnassa on myös muita rajoituksia, kuten `Search`-rajapinnan kyky hakea maksimissaan 7 päivää vanhoja twiittejä. Tulosten täydellisyys ei ole tällöinkään taattua. Täydellisyydellä tarkoitetaan tässä yhteydessä viestikattavuutta. Twitter ei lupaa, että haku palauttaisi kaikki aiheeseen liittyvät viestit, vaan se suosii relevanttiutta (engl. relevance), jota ei kuitenkaan määrittele tarkemmin.[9] Yhdessä nämä rajoitukset johtavat siihen, että Twitter-datan kerääminen on pääsääntöisesti hidasta sekä ennakointia vaativaa: datan kerääminen jälkikäteen ei onnistu. `Streaming`-rajapinnan kautta on mahdollista hakea *kaikki* aiheeseen liittyvät viestit, mutta se edellyttää rajapinnan *reaaliaikaista* käyttöä.

Tässä työssä toteutettiin verkosto Tampereen teknillisen yliopiston Twitter-tilin `@TampereUniTech` seuraajista. Verkoston pohjana käytetty data haettiin Twitterin REST-rajapinnan kautta. Rajapinnan rakenteesta johtuen se tehtiin muuta-

[9] `https://dev.twitter.com/rest/public/search` (luettu 29.8.2017)

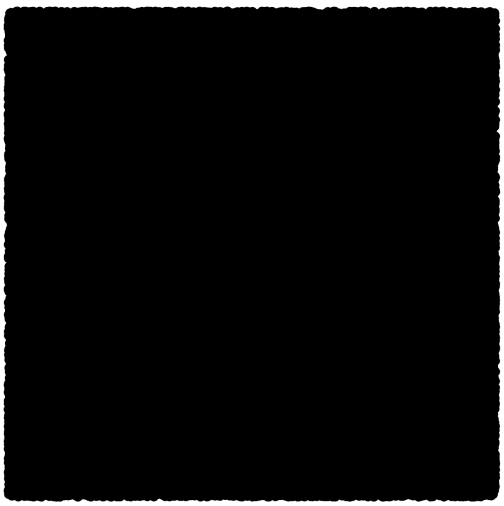
massa osassa. Ensimmäisenä haettiin `@TampereUniTech`-tilin seuraajien käyttäjä-id:t. Seuraajadatan keräämisen aikaan tilillä oli 4446 uniikkia seuraajaa. Näiden id-tunnisteiden avulla voitiin pyytää käyttäjistä lisää tietoja. Verkoston näkökulmasta oleellisia tietoja olivat lähinnä käyttäjän nimi sekä kyseisen käyttäjän 100 tuoreinta twiittiä. Näistä 4446 käyttäjästä 161 kohdalla twiittien hakeminen epäonnistui. Ainakin useimpien kohdalla tämä johtui heidän yksityisyysasetuksistaan, jotka rajasivat heidän twiittinsä ainoastaan heidän tiliään seuraavien saataville. Kyseiset käyttäjät jätettiin verkoston ulkopuolelle. Yhteensä twiittejä kerättiin metadatoineen n. 2,7Gb edestä.

Kun lähdedata oli saatu kerättyä, siirryttiin verkoston luomiseen luvussa 2.3 esiteltyjen perusteiden mukaisesti. `@TampereUniTech`-käyttäjän seuraajilta kerätyt twiitit käytiin läpi silmukassa. Jos twiitissä oli mainittu toisen Twitter-käyttäjän nimi, niin maininnan tekijän sekä mainittun käyttäjän välille luotiin yhteys. Yhteys luotiin siten, että kutakin uniikkia Twitter-käyttäjää kohden luotiin verkostoon solmu, ja maininnan tekijää kuvaavasta solmusta luotiin suunnattu yhteys mainittuun käyttäjään. Uusien yhteyksien painoarvoksi asetettiin yksi. Jos yhteys käyttäjien välillä oli jo olemassa, sen painoarvoa kasvatettiin yhdellä. Tuloksena saatiin suunnattu (kuka mainitsi kenet) ja painotettu (montako kertaa maininta tehtiin) verkosto, joka kuvasi Tampereen teknillisen yliopiston Twitter-seuraajien välillä Twitterissä käytyä dialogia. Verkosto tallennettin XML-pohjaisessa `.gefx`-formaatissa. Twiittien käsittelyyn käytettiin Pythonia.

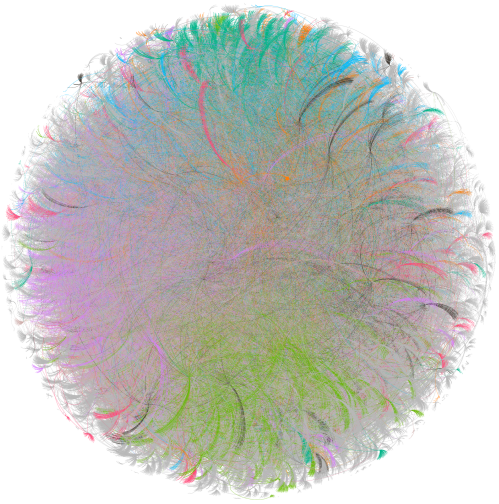
`.gefx`-muotoinen esitystapa visualisoitiin ilmaisella, avoimeen lähdekoodiin perustuvalla Gephi[10]-sovelluksella (Bastian et al., 2009). Koska Twitter-datan pohjalta luotu verkosto sisälsi ainoastaan tiedot solmuista (node) eli käyttäjänimistä sekä heidän välisistään yhteyksistä (kaari, edge), verkosto oli aluksi kaoottisessa järjestyksessä, ilman ladontaa (engl. layout). Latomaton verkosto on esitetty kuvassa 4.4a. Verkosto ladottiin Gephin tarjoamalla ladonta-algoritmilla, jonka parametreja säädettiin käsin. On hyvä huomioida, että eri asetuksilla on mahdollista luoda samasta .gefx-datasta eri tavalla ladottuja verkostoja, jotka mahdollisesti korostavat eri asioita. Esimerkiksi solmujen kokoa ja väriä on mahdollista säätää solmuun yhdistyvien kaarien lukumäärän ja/tai niiden painoarvojen perusteella. Nyt solmut värjättiin Gephin avulla lasketun modulaarisuuten (engl. modularity) mukaisesti, jolloin värit kuvaavat verkoston klustereita. Klusterilla tarkoitetaan solmuja, jotka ovat verkoston näkökulmasta lähellä toisiaan (Soffer & Vázquez, 2005). Gephin avulla ladottu visualisointi on esitetty kuvassa 4.4b.

Ladottu verkosto on jo huomattavasti latomatonta versiota hyödyllisemmän näköinen, mutta se on edelleen niin tiivis, että sen rakenteen hahmottaminen on vaikeaa.

[10]Saatavilla: `https://gephi.org`



(a) *TTY:n Twitter-seuraajien pohjalta luotu verkosto ilman ladontaa Gephi-sovelluksessa visualisoituna. Koska solmut ovat kaoottisessa järjestyksessä ja toistensa päällä, visualisointi on käyttökelvoton.*

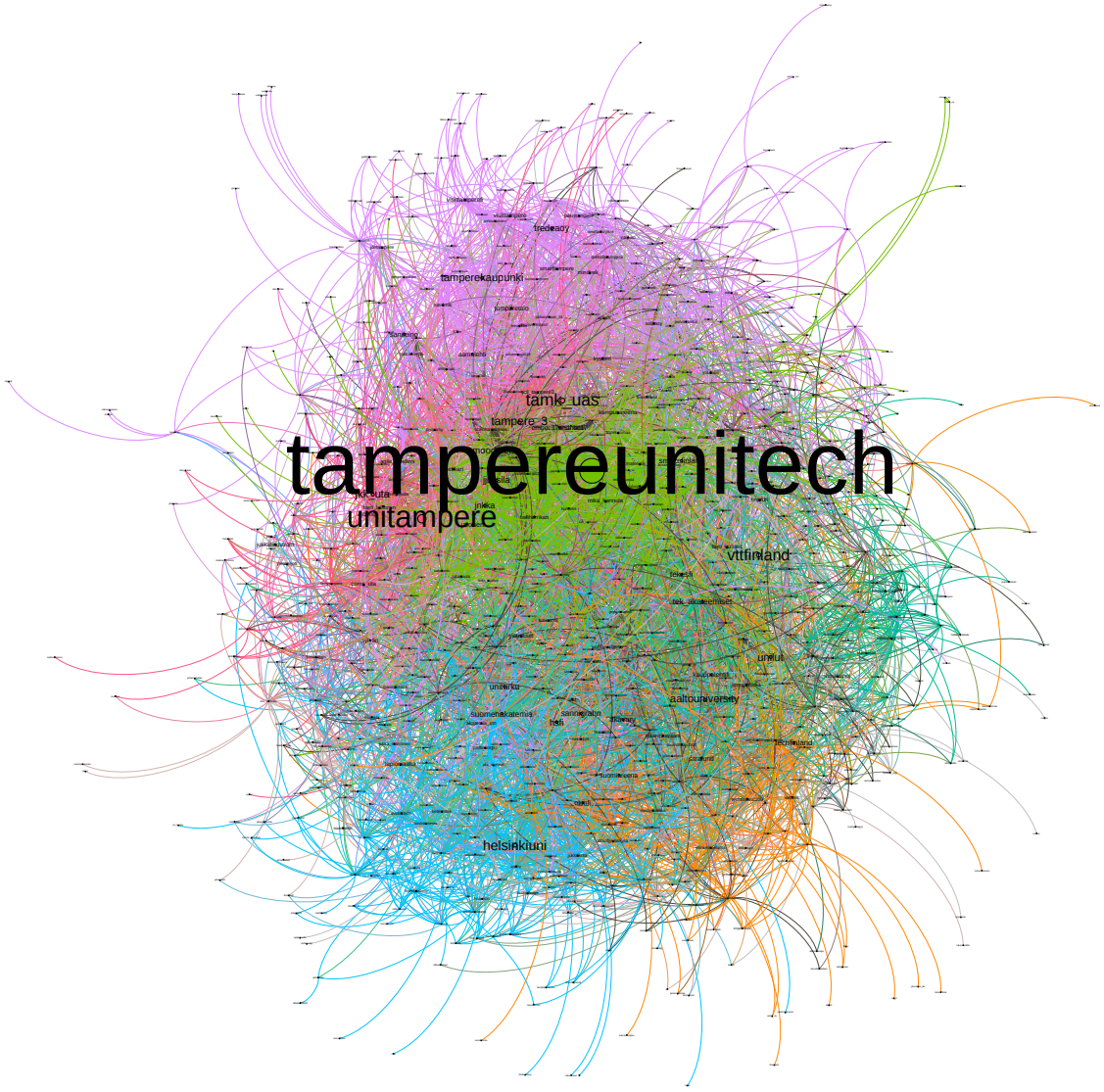


(b) *TTY:n Twitter-seuraajien pohjalta luotu verkosto Gephi-sovelluksessa ladottuna ja värjättynä. Verkostossa on noin 70 000 solmua ja se on edelleen niin tiivis, että sen tulkitseminen on hyvin haastavaa.*

Kuva 4.4. *Verkoston visualisoinnin iterointia*

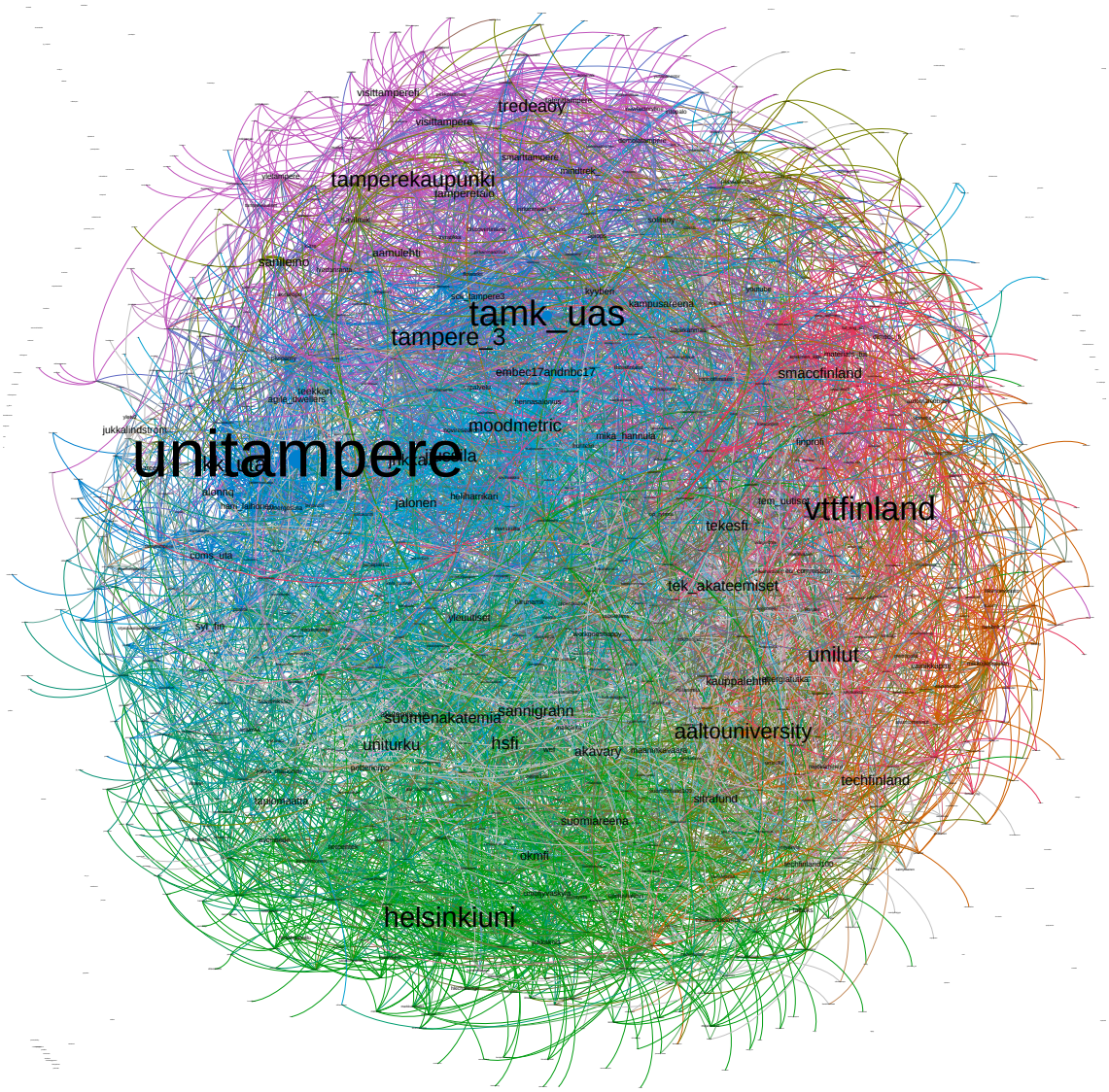
Rakenteen esiintuomiseksi verkostoa voidaan suodattaa (engl. filter) siitä laskettujen tunnuslukujen perusteella. Tässä tapauksessa suodatusperusteeksi valittiin solmuista laskettu PageRank, joka kuvaa todennäköisyyttä sille, että verkoston satunnaisesta solmusta aloittanut ja verkoston kaaria pitkin satunnaisesti solmusta toiseen hyppivä käyttäjä päätyy kyseiseen solmuun tietyllä ajan hetkellä (Page, Brin, Motwani, & Winograd, 1999). PageRank painottaa solmuja, jotka ovat linkittyneet toisiin solmuihin, joten tässä tapauksessa sen oletetaan korostavan verkoston tärkeimpiä solmuja. Verkosto suodatettiin PageRankin perusteella siten, että jäljelle jäi 972 solmua. Alkuperäisessä verkostossa solmuja oli 69 872, eli jäljelle jäi noin 1,4% alkuperäisistä solmuista. Ehtojen mukaisen suodattamisen pystyy tekemään suoraan Gephistä käsin, ja hieman rajattu kuva lopputuloksesta on esitetty kuvassa 4.5.

Suodatettu verkosto ladottiin uudelleen Gephissä. Kyseinen versio on etenkin hieman zoomattuna jo varsin luettava ja käyttökelpoinen. Verkostosta nähdään kuitenkin heti, että `@TampereUniTech`-solmu korostuu huomattavasti muihin solmuihin verrattuna. Tässä tapauksessa tämä on odotettavaa, koska verkosto on luotu kyseisen tilin ympärille. `@TampereUniTech`-solmu ei kuitenkaan ole erityisen kiinnostava, koska on itsestään selvää, että se on oleellinen osa tässä tutkittua verkostoa. Sen sijaan sitä ympäröivät rakenteet ovat tutkimuksen kannalta oleellisempia. Verkostosta poistettiin kokeeksi `@TampereUniTech`-solmu, ja verkosto ladottiin vielä kerran uudelleen. Lopullinen versio on esitetty kuvassa 4.6.



Kuva 4.5. *TTY:n Twitter-seuraajien pohjalta luotu verkosto Gephi-sovelluksessa ladottuna sekä PageRank-algoritmin perusteella suodatettuna*

Kuva 4.6 paljastaa verkostosta jo selvästi enemmän kuin mitä taulukkomuotoinen data paljastaisi, vaikka taulukko olisikin järjestetty esimerkiksi PageRankin mukaiseen järjestykseen. Oleellista on, että solmut ovat jakautuneet kuvassa klustereihin, joita verkostossa käytetyt värit kuvaavat. Sinisessä klusterissa korostuvat Tampereen alueen korkeakoulut `@unitampere`, `@tamk_uas` ja `@tampere_3`, jotka siis ovat selvästi verkostoituneet keskenään. Violetista klusterissa erottuvat muun muassa `@tampereenkaupunki`, `@visittampere` sekä `@aamulehti`, jotka ovat kaikki Tampereen näkökulmasta paikallisia tahoja. Samalle alueelle on keskittynyt merkittävä määrä myös pienempiä paikallisia toimijoita. Vihreässä ja punaisessa klusterissa korostuvat muut korkeakoulut: `@helsinkiuni`, `@uniturku`, `@unilut` ja `@aaltouniversity`. Klusterien välissä ovat muun muassa `@tek_akateemiset`, `@tekesfi` ja `@yleuutiset`. Verkostosta löytyy myös yksityishenkilöitä, mutta ne



Kuva 4.6. *TTY:n Twitter-seuraajien pohjalta luotu verkosto Gephi-sovelluksessa ladottuna sekä sekä PageRank-algoritmin perusteella suodatettuna, ilman @TampereUniTech-käyttäjää*

eivät nouse esille yhtä selvästi kuin suuremmat instituutiot.

4.3 Töiden arviointi

Tässä luvussa arvioidaan case-tutkimusten onnistumista datatuotteelle luvussa 3.6 ehdotetun arkkitehtuurin näkökulmasta sekä vastaavasti ehdotetun arkkitehtuurimallin mielekkyyttä case-tutkimuksissa saatujen kokemusten perusteella. Tämän lisäksi toista case-työtä arvioidaan data-analytiikan ja visuaalisen analytiikan onnistumisen näkökulmasta.

Ensimmäisessä case-tutkimuksessa toteutetun järjestelmän arkkitehtuuri onnistui

luvussa 3.1 mainitussa olemassa olevien ohjelmistokomponenttien hyödyntämisessä. Esimerkiksi käyttäjien hallintaan ja autentikointiin saatiin käytännössä valmis ratkaisu Firebasesta. Tämä nopeutti merkittävästi prototyypin valmistumista. Käyttöliittymää varten tarvittu Vue.js-projektipohja puolestaan saatiin generoitua helposti `vue-cli`-työkalulla. Tämä minimoi projektin pystyttämiseen tarvitun ajan. Vuen komponenttipohjaisuus helpotti koodin uudelleenkäyttöä. Esimerkiksi pitkien listojen kätkemiseen toteutettua käyttöliittymäkomponenttia oli helppo käyttää useammassa paikassa.

Yksittäisiin palveluihin jaettu toteutus tuki myös Ostinato-mallin (luku 3.2) mukaista löyhää sidontaa sekä manuaalisten vaiheiden mahdollistamista. Käyttöliittymää kehitettiin erillään dataa käsittelevistä palveluista. Siksi sen vaihtaminen toiseen toteutukseen olisi varmasti mahdollista ilman että datankäsittelyputken aiempia osia jouduttaisiin uusimaan. Myös toisessa case-tutkimuksessa onnistuttiin palveluiden välisessä löyhässä sidonnassa. Työ eteni datatuotteen kehittämisen vaiheiden mukaisesti selvästi kolmessa osassa, joita olivat *datan kerääminen*, *kerätyn datan käsittely ja verkoston generoiminen* sekä *verkoston visualisointi ja analysointi*.

Nyt toteutetut prototyypit olivat olennaisilta osin aiemmin kuvatun datatuotteen arkkitehtuurimallin mukaisia. Ehdotetun arkkitehtuurimallin onnistumista voidaan siis arvioida niihin pohjautuen. Prototyypeistä saatujen kokemusten perusteella ehdotettua arkkitehtuurimallia voidaan pitää toimivana. Yksittäisiin palveluihin jaettu toteutus tuki valmiiden ohjelmistokomponenttien uudelleenkäyttöä sekä prosessin eri vaiheiden välistä löyhää sidontaa, jotka molemmat todettiin aiemmin tavoiteltaviksi piirteiksi. Toisaalta prototyypit olivat laajuudeltaan rajattuja, vaikka ehdotetun arkkitehtuurin on tarkoitus toimia myös suuremmissa ja monimutkaisemmissa järjestelmissä. Tulosten tarkentamiseksi ehdotusta olisi hyvä kokeilla jatkossa myös monimutkaisemman datatuotteen yhteydessä.

Tutkimuksessa huomattiin myös, että useat luvussa 2.7 mainitut visuaaliseen analytiikkaan liittyvät haasteet ovat edelleen olemassa. Luvussa listatuista seitsemästä tyypillisestä ongelmakohdasta ainoastaan yksi, visualisoinnin toimiminen kaikenlaisilla päätelaitteilla (eli käytännössä sekä tietokoneilla että mobiililaitteilla) on jokseenkin ratkennut modernien web-teknologioiden myötä. Toteuttamalla web-käyttöliittymät responsiivisesti on mahdollista luoda visualisointeja, jotka toimivat niin työpöydällä kuin älypuhelimellakin. Jäljelle jäi kuitenkin teknisiä ja ei-teknisiä haasteita, joita käydään läpi seuraavaksi.

Nyt kerätty data oli teknisesti hyvää, eli se sisälsi luvatut kentät eikä vaatinut puhdistamista. Twitterin REST-rajapinnan rajoituksista johtuen sen kerääminen vei kuitenkin useita vuorokausia. Tällöin ensimmäisenä kerätyt twiitit olivat vanhempia kuin viimeisimpänä kerätyt. Koska twiittejä kerättiin vain 100 kappaletta käyttäjää

kohden, niin aktiivisimpien twiittaajien välinen keskustelu saattoi jäädä aineistossa vain toispuoleiseksi. Tämän vaikutusta kokonaisuuteen on kuitenkin hankala arvioida. Laajemman aineiston (esimerkiksi 1000 twiittiä per käyttäjä) kerääminen olisi tuonut lisää mahdollisuuksia, mutta sen kerääminen Twitterin rajapinnan kautta olisi vienyt lähes kymmenkertaisen ajan nykyiseen verrattuna. Se ei siis olisi ratkaissut edellä mainittuja ongelmia. On myös vaikeaa arvioida, että paljonko sosiaalisen median datan pohjalta tehtyä analyysiä voi hyödyntää käytännön tilanteissa.

Datan keräämisen ajankohdalla oli myös vaikutusta verkoston rakenteeseen. Koska aineistona oli nimenomaan käyttäjien 100 tuoreinta twiittiä, eri ajankohtana kerätyllä aineistolla saadut tulokset voivat olla erilaisia kuin nyt. Verkostoa olisi voinut myös suodattaa tai latoa usealla eri tavalla. Nyt tehty suodatus tähtäsi *Cobweb*-projektin mukaisesti Tampereen teknilliselle yliopistolle läheisten tahojen välisten yhteyksien paljastamiseen. Toinen lähestymistapa saman datan analysoimiseksi voisi olla esimerkiksi `@TampereUniTech`-käyttäjän seuraajien poistaminen verkostosta. Tällöin verkosto sisältäisi pääasiallisesti tahoja, joilla ei olisi suoraa yhteyttä Tampereen teknilliseen yliopistoon. Esimerkiksi markkinointinäkökulmasta he voisivat olla kiinnostavia kontakteja.

Suorituskykyyn liittyviä haasteita ilmeni lähinnä suodattamattoman verkoston visualisoinnin yhteydessä. Lähes 70 000 solmua sisältävä verkosto tuntui olevan Gephin suorituskyvyn ylärajoilla. Verkostoa visualisoitaessa ja etenkin ladottaessa Gephi hidastui merkittävästi. Tämä käytännössä esti vuorovaikutteiset interaktiot verkoston kanssa. Tilanne parani oleellisesti vasta, kun verkostoa suodatettiin PageRankin perusteella. Suodatettu verkosto sisälsi enää noin 1000 solmua, jolloin siihen liittyvä ladonta sekä interaktiot olivat sulavia. Jos dataa olisi ollut merkittävästi enemmän, ladonta tai alustava suodattaminen olisi luultavasti jouduttu ajamaan Gephin ulkopuolella.

5. YHTEENVETO

Työn alussa esitettiin kaksi tutkimuskysymystä. Ensimmäisessä kysymyksessä haluttiin löytää määritelmä "datatuote"-termille kirjallisuuskatsauksen keinoin. Edellä todettiin, että termi ei ole vakiintunut ja aiheeseen liittyvää kirjallisuutta oli saatavilla ainoastaan rajallisesti. Termi saatiin kuitenkin määriteltyä työn vaatimalla tarkkuudella. Saavutettu määritelmä on esitetty alla:

> Datatuote on tietotekninen järjestelmä tai kokonaisuus, joka saa syötteekseen dataa yhdestä tai useammasta lähteestä ja **jalostaa** kyseistä dataa **vuorovaikutteisesti** tuoden siten **lisäarvoa** loppukäyttäjälleen.

Toisessa kysymyksessä haluttiin löytää web-pohjaiselle datatuotteelle soveltuva arkkitehtuurimalli. Vastausta haettiin ADR-tutkimusmenetelmän avulla. Menetelmän ensimmäinen vaihe, ongelman määrittäminen, jakautui työn kontekstista johtuen kahteen osuuteen. Luvussa 3 etsittiin ratkaisua kirjallisuuteen pohjautuvan teorian avulla. Luvussa koottiin aluksi yhteen datatuotteen arkkitehtuurille tavoiteltavia piirteitä. Tämän jälkeen vertailtiin web-kehityksessä käytettäviä arkkitehtuurimalleja ja peilattiin niitä datatuotteen arkkitehtuurin vaatimuksiin. Vertailua ei tehty yksittäisten käyttötapausten näkökulmasta, jotta mallien käsittely voitiin pitää yleisellä tasolla. Sopivin arkkitehtuurimalli on tietysti aina tapauskohtaista, koska eri tilanteissa painottuvat asiat vaihtelevat. Teorian pohjalta kehitettiin ehdotus web-pohjaisen datatuotteen arkkitehtuurista (luku 3.6), ja yleiskuva siitä esitettiin kuvassa 3.7. Ehdotuksen mukainen malli pohjautuu microservices-arkkitehtuuriin. Toiminnallisuus on jaettu eri palveluihin luvussa 3.2 esitettyjen datatuotteen loogisten kehitysvaiheiden mukaan, jotka taas pohjautuvat samassa luvussa esitettyihin CRISP-DM- sekä Ostinato-prosessimalleihin.

ADR-menetelmän toisessa vaiheessa tutkimusongelmaa pyrittiin ratkaisemaan käytännön toteutuksella. Tätä kuvataan luvussa 4. Ensimmäisessä case-esimerkissä toteutettiin datatuotteen prototyyppi eli *MatchUs*-suosittelulistan *Cobweb*-projektin tarpeisiin. Järjestelmän arkkitehtuuri noudatti aiemmin luvussa 3 opittuja periaatteita, joten se toimi ADR-tutkimusmenetelmän mukaisena IT-järjestelmän käytännön toteutuksena. Datan kerääminen ja käsittely toteutettiin erillisinä palveluina, joten niiden tarkempi käsittely rajattiin tämän työn ulkopuolelle. Varsinainen käyttöliittymä toteutettiin nykyaikaisena SPA-sovelluksena, ja sen taustajärjestelmänä toimi Googlen tarjoama Firebase.

ADR-menetelmän kolmannessa ja neljännessä vaiheessa opittiin toteutetusta järjestelmästä sekä yleistettiin opittua geneerisempiin tapauksiin. Työssä toteutetuista prototyypeistä saatujen kokemusten pohjalta arvioiden luvussa 3.6 ehdotettua arkkitehtuurimallia voidaan pitää onnistuneena: järjestelmän yksittäiset osat voitiin toteuttaa toisistaan riippumatta, ja niiden väliset sidokset saatiin pidettyä löyhinä. Toisaalta nyt toteutettu prototyyppi oli suhteellisen yksinkertainen, mutta arkkitehtuurimallin on tarkoitus sopia myös suuremmille järjestelmille. Mallin toimivuutta olisi hyvä testata myös monimutkaisemmalla järjestelmällä.

Toisessa case-esimerkissä toteutettiin Twitter-verkostoanalyysi, joka tuki *Cobweb*-projektin pyrkimyksiä tutkia sosiaalisesta mediasta kerättyä dataa visuaalisen verkostoanalyysin keinoin. Analyysissä muodostettiin verkosto Tampereen teknillisen yliopiston Twitter-seuraajien käymistä keskusteluista sekä analysoitiin sitä eksploratiivisesti visuaalisen verkostoanalyysin keinoin. Verkoston ladonta ja suodattaminen vaativat useampia iteraatioita toimivimman lopputuloksen löytämiseksi. Twitter-datan rajoitteista johtuen on tärkeää huomioida, että analyysin kohteena olevaa verkostoa kannattaa tulkita varovaisesti. Verkostoa analysoidessa tuli myös selväksi, että miksi visuaalisen analytiikan yhteydessä puhutaan lähes aina myös sen haasteista: tuhansia solmuja sisältäneen verkoston visualisointi ihmisnäkökulmasta hyödylliseen muotoon ei ollut triviaali tehtävä. Saman datan pohjalta olisi myös mahdollista luoda lähes ääretön määrä erilaisia verkostoja. Toisaalta tämä toimi hyvänä opetuksena ekploratiivisen data-analytiikan ominaispiirteistä: data itsessään ei vielä vastaa yhteenkään kysymykseen, eikä ensimmäinen sen pohjalta tehty analyysi vie välttämättä lainkaan eteenpäin. Yleensä kuitenkin kannattaa jatkaa ja kokeilla vaihtoehtoisia lähestymistapoja. Jos dataa on riittävästi, niin lähes aina sieltä voidaan löytää ainakin *jotain*. Minimissään tulee oppineeksi useamman lähestymistavan, jotka eivät kyseiselle aineistolle toimi.

LÄHTEET

5 best javascript frameworks in 2017. (2017). Lainattu 4.5.2017, saatavilla https://da-14.com/blog/5-best-javascript-frameworks-2017

Abbott, J. (2001). Data data everywhere – and not a byte of use? *Qualitative Market Research: An International Journal*, *4*(3), 182–192. doi: 10.1108/13522750110393080

Andrienko, N., & Gennady, A. (2006). *Exploratory Analysis of Spatial and Temporal Data*. Berlin/Heidelberg: Springer-Verlag. doi: 10.1007/3-540-31190-4

Aoyama, M., & others. (1998). New age of software development: How component-based software engineering changes the way of software development. Teoksessa *1998 International Workshop on CBSE* (s. 1–5). Citeseer.

Babar, M. A., Zhu, L., & Jeffery, R. (2004). A framework for classifying and comparing software architecture evaluation methods. Teoksessa *2004 Australian Software Engineering Conference.* (s. 309–318). doi: 10.1109/ASWEC.2004.1290484

Basole, R. C. (2009). Visualization of interfirm relations in a converging mobile ecosystem. *Journal of Information Technology*, *24*(2), 144–159. doi: 10.1057/jit.2008.34

Bastian, M., Heymann, S., & Jacomy, M. (2009). Gephi: an open source software for exploring and manipulating networks. *ICWSM*, *8*, 361–362.

Bendoly, E. (2016). Fit, Bias, and Enacted Sensemaking in Data Visualization: Frameworks for Continuous Development in Operations and Supply Chain Management Analytics. *Journal of Business Logistics*, *37*(1), 6–17. doi: 10.1111/jbl.12113

Boehm, B. W., & Scherlis, W. L. (1992). Megaprogramming (Preliminary Version. Teoksessa *in DARPA SW Conference. 1992: Meridien* (s. 28–30).

Bostock, M., Ogievetsky, V., & Heer, J. (2011). D3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, *17*(12), 2301–2309. doi: 10.1109/TVCG.2011.185

Davenport, T. H. (2013). Analytics 3.0. *Harvard Business Review*. Lainattu 23.5.2017, saatavilla https://hbr.org/2013/12/analytics-30

Davenport, T. H., & Kudyba, S. (2016). Designing and Developing Analytics-Based Data Products. *MIT Sloan Management Review; Cambridge*, *58*(1), 83–89.

Fox, P., & Hendler, J. (2011). Changing the Equation on Scientific Data Visualization. *Science*, *331*(6018), 705–708. doi: 10.1126/science.1197654

Gershon, N., & Eick, S. G. (1998). Guest Editors' Introduction: Information Visualization. The Next Frontier. *Journal of Intelligent Information Systems*, *11*(3), 199–204. doi: 10.1023/A:1008680323877

Huhtamäki, J. (2016). *Ostinato Process Model for Visual Network Analytics: Experiments in Innovation Ecosystems.* Tampere University of Technology.

Huhtamäki, J., Russell, M. G., Rubens, N., & Still, K. (2015). Ostinato: The Exploration-Automation Cycle of User-Centric, Process-Automated Data-Driven Visual Network Analytics. Teoksessa *Transparency in Social Media* (s. 197–222). Springer, Cham. doi: 10.1007/978-3-319-18552-1_11

Jacobs, A. (2009). The Pathologies of Big Data. *Commun. ACM*, *52*(8), 36–44. doi: 10.1145/1536616.1536632

Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., & Shahabi, C. (2014). Big Data and Its Technical Challenges. *Communications of the ACM*, *57*(7), 86–94. doi: 10.1145/2611567

Keim, D. (2002). Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, *8*(1), 1–8. doi: 10.1109/2945.981847

Keim, D., Andrienko, G., Fekete, J.-D., Görg, C., Kohlhammer, J., & Melançon, G. (2008). Visual Analytics: Definition, Process, and Challenges. Teoksessa A. Kerren, J. T. Stasko, J.-D. Fekete, & C. North (toim.), *Information Visualization* (s. 154–175). Springer Berlin Heidelberg. doi: 10.1007/978-3-540-70956-5_7

Knight, J. C., & Dunn, M. F. (1998). Software quality through domain-driven certification. *Annals of Software Engineering*, *5*(1), 293.

Labrinidis, A., & Jagadish, H. V. (2012). Challenges and Opportunities with Big Data. *Proc. VLDB Endow.*, *5*(12), 2032–2033. doi: 10.14778/2367502.2367572

Leff, A., & Rayfield, J. T. (2001). Web-application development using the Model/View/Controller design pattern. Teoksessa *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference* (s. 118–127). doi: 10.1109/EDOC.2001.950428

Meyer, M. H., & Zack, M. H. (1996). The Design and Development of Information Products. *Sloan Management Review; Cambridge*, *37*(3), 43.

Mikowski, M. S., & Powell, J. C. (2013). Single page web applications. *B and W*.

Miller, M. (2016). *Sequoia - Innovate or Die: The Rise of Microservices.* Lainattu 19.9.2017, saatavilla `https://www.sequoiacap.com/article/build-us-microservices/`

Mizik, N., & Jacobson, R. (2003). Trading Off Between Value Creation and Value Appropriation: The Financial Implications of Shifts in Strategic Emphasis. *Journal of Marketing*, *67*(1), 63–76.

Namiot, D., & Sneps-sneppe, M. (2014). On micro-services architecture. *International Journal of Open Information Technologies*, *2*(9).

Olshannikova, E., Olsson, T., Huhtamäki, J., & Kärkkäinen, H. (2017). Conceptualizing Big Social Data. *Journal of Big Data*, *4*(1). doi: 10.1186/s40537-017

-0063-x

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). *The PageRank citation ranking: Bringing order to the web.* (Tekn. rap.). Stanford InfoLab.

Patel, K. (2016). *What is reactive programming?* Lainattu 15.9.2017, saatavilla `https://medium.com/@kevalpatel2106/what-is-reactive-programming-da37c1611382`

Perry, D. E., & Wolf, A. L. (1992). Foundations for the Study of Software Architecture. *ACM SIGSOFT Software engineering notes*, *17*(4), 40–52. doi: 10.1145/141874.141884

Petitit, F., & Tricot, M. (2014). *The new Web application architectures and their impacts for enterprises – Part 1.* Lainattu 6.7.2017, saatavilla `http://blog.octo.com/en/new-web-application-architectures-and-impacts-for-enterprises-1/`

Rahm, E., & Do, H. H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, *23*(4), 3–13.

Ries, E. (2011). *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses.* Crown Books.

Roberts, M. (2016). *Serverless architectures.* Lainattu 7.9.2017, saatavilla `https://martinfowler.com/articles/serverless.html`

Runkler, T. A. (2012). *Data Analytics.* Wiesbaden: Springer. doi: 10.1007/978-3-8348-2589-6

Schreck, T., & Keim, D. (2013). Visual Analysis of Social Media Data. *Computer*, *46*(5), 68–75. doi: 10.1109/MC.2012.430

Sein, M. K., Henfridsson, O., Purao, S., Rossi, M., & Lindgren, R. (2011). Action Design Research. *MIS Quarterly*, *35*(1), 37–56.

Shapiro, D. (2016). *Understanding component-based architecture.* Lainattu 20.4.2017, saatavilla `https://medium.com/@dan.shapiro1210/understanding-component-based-architecture-3ff48ec0c238`

Shaw, M., & Garlan, D. (1994). An Introduction to Software Architecture. *Institute for Software Research*.

Shneiderman, B. (1996). The eyes have it: a task by data type taxonomy for information visualizations. Teoksessa *Proceedings 1996 IEEE Symposium on Visual Languages* (s. 336–343). doi: 10.1109/VL.1996.545307

Soffer, S. N., & Vázquez, A. (2005). Network clustering coefficient without degree-correlation biases. *Physical Review E*, *71*(5). doi: 10.1103/PhysRevE.71.057101

Tallon, P. P. (2013). Corporate Governance of Big Data: Perspectives on Value, Risk, and Cost. *Computer*, *46*(6), 32–38. doi: 10.1109/MC.2013.155

Thomas, J. J., & Cook, K. A. (toim.). (2005). *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. Los Alamitos, CA: National Visualization and Analytics Center.

Tukey, J. W. (1977). *Exploratory data analysis*.

Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. (2015). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. Teoksessa *2015 10th Computing Colombian Conference (10ccc)* (s. 583–590). doi: 10.1109/ColumbianCC.2015.7333476

Ward, J. S., & Barker, A. (2013). Undefined by data: a survey of big data definitions. *arXiv preprint arXiv:1309.5821*.

Wasserman, S., & Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press. (Google-Books-ID: CAm2DpIqRUIC)

Weiss, M., & Gangadharan, G. R. (2010). Modeling the mashup ecosystem: structure and growth. *R&D Management*, *40*(1), 40–49. doi: 10.1111/j.1467-9310.2009.00582.x

Wirth, R., & Hipp, J. (2000). CRISP-DM: Towards a standard process model for data mining. Teoksessa *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining* (s. 29–39).

Worrall, P., & Chaussalet, T. (2011). Development of a web-based system using the model view controller paradigm to facilitate regional long-term care planning. Teoksessa *2011 24th International Symposium on Computer-Based Medical Systems (CBMS)* (s. 1–7). doi: 10.1109/CBMS.2011.5999123