Tampereen yliopisto

Antti Arela

PÄÄTÖKSENTEKOON VAADITTAVAN TIEDON KOOSTAMINEN VIRANOMAISPÄÄTÖSJÄRJESTELMÄSSÄ

Informaatioteknologian ja viestinnän tiedekunta
Diplomityö
Kesäkuu 2019

TIIVISTELMÄ

Antti Arela: Päätöksentekoon vaadittavan tiedon koostaminen viranomaispäätösjärjestelmässä
Diplomityö
Tampereen yliopisto
Tietotekniikan DI-tutkinto-ohjelma
Kesäkuu 2019

Tietojärjestelmien dataintegraatioissa yhdistetään useamman verkossa sijaitsevan lähteen tietoja ja muunnetaan niitä tietylle sovellusalueelle tarkoituksenmukaiseksi. Tietoja hyödynnetään toiminnanohjausjärjestelmissä, joiden tukemana suoritetaan yrityksen tai viraston hallinnollisia tai tuotannollisia prosesseja. Toiminnanohjausjärjestelmiä toteutetaan ottamalla käyttöön organisaation laajuudelle yksi tai useampi omaan tarkoitukseensa toteutettu sovellus, jotka keskustelevat toisilleen ja ulkoisille palveluille eri tietoverkkojen sovellustason protokollia hyödyntäen. Osalla sovelluksista voi olla web-käyttöliittymä, jonka kautta itse käyttäjät toteuttavat prosessia.

Innovaatiorahoituskeskus Business Finlandin Rahoitusjärjestelmissä koostetaan tietoa eri integraatiolähteistä ja tieto esitetään tarkoituksenmukaisella tavalla käyttäjälle eli hakijaorganisaation edustajalle tai viraston työntekijälle. Rahoitusprosessissa on eri vaiheita. Niistä yksi on esitysvalmistelu, jossa virkailija muodostaa asiakkaan hakemuksen ja asiakasanalyysin perusteella rahoitusesityksen, jonka perusteella tehdään rahoituspäätös. Asiakasanalyysiä varten hakijaorganisaatiosta tarvitaan taloustietoja eri lähteistä.

Tässä työssä tutkitaan rahoitusprosessin esitysvalmisteluvaiheeseen suunnitellun sovelluksen toteutusprojektin aikana esiin tulleita haasteita ja ongelmia tiedon koostamiseen ja esittämiseen liittyen. Työssä esitetään sovellusarkkitehtuuria ja välimuistirakenteita. Koostamisen haasteita on tiedon sisällön tulkitsemisessa sekä tiedonhakujen kohdistamisessa eri lähteisiin tietyissä tilanteissa. Lisäksi haasteita aiheuttavat tietojen aikariippuvaiset muutokset ja sitä kautta tiedonhakujen ajoitukset. Myös sovelluksen luotettavan toiminnan varmistaminen sekä jatkokehityskelpoisuus ovat tärkeitä osa-alueita. Työn tuloksena on sovelluksen arkkitehtuurikuvaukset, analyysi arkkitehtuuriratkaisuista sekä analyysi sovelluskehityksen menetelmistä.

Avainsanat: integraatio, tiedon koostaminen, sovellusarkkitehtuuri, web, sovelluskehitys

ABSTRACT

Antti Arela: Composition of Data Required for a Decision in an Public Official Decision System
Master of Science Thesis
Tampere University
Degree Programme in Information Technology, MSc
June 2019

The goal of Information System Data Integration is to combine data from multiple sources in a network and transform the data into a representation of information suitable for a certain domain. The information is utilized in Enterprise Resource Planning systems that support the business processes of a corporation or an agency. An Enterprise Resource Planning system can be implemented by deploying one application, or multiple applications for different purposes within an organisation. The applications communicate with each other and external services through various application-level communication protocols. Some of the applications may contain a Web-based user interface through which the business process is performed.

The funding information systems of Innovation Funding Agency Business Finland collect and compose data from various integration sources, and represent the data to the customer or the employee of the agency in an appropriate way. The funding process consists of several phases, one of which is the motion preparation phase. In this phase, an official formulates a funding motion based on the funding application filled by the customer and analysis of the applicant's economical standing. The funding motion is the basis for a funding decision. For the analysis of the economical standing, information is required from multiple sources.

This thesis examines the challenges and problems regarding the composition and representation of data emerged in a software development project. The developed software is an application designed for the motion preparation phase of the funding process. This thesis presents architectural solutions and cache structures. The challenges in data composition consist of interpreting the contents of the data, as well as targeting data queries to various data sources in suitable situations. In addition, challenges emerge from time-based variation of the data and the resulting requirements for proper timing of queries. The reliable operation of the application and enabling further development of the application are also an important aspect during software development. The results of this thesis are the architecture descriptions, the analysis of the architectural solutions, and the analysis of the software development methods.

Keywords: integration, data composition, software architecture, web, software development

ALKUSANAT

Haluan kiittää Business Finlandia siitä, että olen saanut käyttää heidän järjestelmäkehitysprojektia opinnäytetyöni kohteena. Annan erityiskiitoksen Digital Backend -vastuualueen työntekijöille tuesta opinnäytetyön tekemisessä sekä heidän antamastaan palautteesta työn sisältöön.

Haluan kiittää työntantajaani Solitaa joustavuudesta ja tuesta diplomityön kirjoittamiselle. Haluan kiittää myös solitalaista ohjaajaani Janne Rintasta, joka on opastanut opinnäytetyön punaisen langan löytämisessä ja työn jäsentelyssä, sekä tarjonnut palautetta kirjoitusprosessin aikana. Lisäksi haluan kiittää solitalaista Timo Lehtosta, joka on tarjonnut palautetta ja antanut ohjeita diplomityön vaatimuksien täyttämisestä.

Kiitän myös tarkastajaani Timo Aaltosta, joka auttoi löytämään aiheen ja opasti tieteellisen tekstin kirjoittamisessa.

Tampereella, 5. kesäkuuta 2019

Antti Arela

SISÄLLYSLUETTELO

KUVALUETTELO

TAULUKKOLUETTELO

LYHENTEET JA MERKINNÄT

CRM	asiakkuudenhallinta, (engl. Customer-relationship Management)
ERP	toiminnanohjausjärjestelmä (engl. Enterprise Resource Planning)
HTTP	hypertekstin tiedonsiirtoprotokolla (engl. HyperText Transfer Protocol)
JSON	datan esitysmuoto (engl. JavaScript Object Notation
REST	tilanhallinnan ajatusmalli(engl. Representational State Transfer)
SOAP	rakenteellisen tiedon siirtoprotokolla (engl. Simple Object Access Protocol)
URL	verkkosivun osoite (engl. Uniform Resource Locator)

1 JOHDANTO

Business Finland on Työ- ja elinkeinoministeriön alaisuudessa toimiva innovaatiorahoituskeskus [3] (lyh. BF, ennen Tekes). Tässä työssä käsitellään projektia, jossa on toteutettu uusi esitysvalmistelusovellus BF:n rahoitusprosessin tueksi. BF haluaa uudistaa rahoitussovelluksiaan ja pyrkimys on tehostaa virkailijoiden työntekoa tarkoituksenmukaisilla ja sulavilla käyttöliittymäratkaisuilla sekä tiedonhaun automatisoinnilla. BF:n rahoitusjärjestelmien uudistamistavoitteina on purkaa vanhentuneet sovellukset ja rakentaa tilalle modernit sovellukset, jotka palvelevat rahoitusprosessia paremmin.

Esitysvalmistelulla tarkoitetaan vaihetta, jossa yksi tai useampi virkailija tekee asiakkaan rahoitushakemuksen sekä asiakasanalyysin perusteella ns. *rahoitusesityksen*, joka sisältää perusteluja, voiko BF toimia hankkeessa rahoittajana. Perusteluihin lukeutuu virkailijan täyttämien tietojen lisäksi myös hakijan syöttämät tiedot sekä ulkoiset asiakastiedot. Esitysvalmistelussa analysoidaan mm. hakijaorganisaation taloustilannetta, mikä vaatii tiedon keräämistä monesta eri tietolähteestä. Lisäksi virkailija eli esitysvalmistelija perustelee joko myöntämisen tai hylkäämisen sekä taloustietojen että hakijan kirjoittamien tekstien perusteella. Näistä muodostuu *rahoitusesitys*, joka siirretään eteenpäin rahoituspäättäjälle. Päättäjä tekee rahoituspäätöksen esityksen perusteella.

Rahoituksen keskeisenä tietojärjestelmänä on rahoitussovelluskokonaisuus, johon kuuluu useampi käyttöliittymäsovellus, integraatiopalvelu sekä tietokanta. BF:n järjestelmien uudistamisessa keskeisenä osana on purkaa vanha monoliittinen sovellus, joka kattaa rahoitusprosessin kaikki vaiheet. Tilalle halutaan ottaa käyttöön pienempiä sovelluksia mikropalveluarkkitehtuuria soveltaen. Sovelluksien jakaminen pienemmiksi omiin vastuualueisiin ja tehtäviin helpottaa niiden hallittavuutta ja testaamista. Osa rahoitussovelluksista ovat vanhentuneita, mistä johtuen niiden tekninen tuki ovat vähenemässä sekä sovelluskehityksen tehokkuus kärsii. Business Finland myös elää organisaatiomuutoksen tuomaa siirtymisaikaa tietojärjestelmien osalta, mikä tarkoittaa, että tietojärjestelmien on integroiduttava Työ- ja elinkeinoministeriön yhteiseen CRM-järjestelmään.

Tämän työn tavoitteena on kuvata tavoitetilan ratkaisu ja sen saavuttamiseksi tehty toteutustyö, jolla koostetaan tietoja esitysvalmistelijan saataville sekä Business Finlandin sisäisten sekä ulkoisten muutostarpeiden pohjalta. Tämän työn tekijä on ollut sovelluskehittäjänä projektin kehitystiimissä. Tietojen koostamisessa on otettava huomioon sekä tiedon muuntaminen sopivaan muotoon että tiedon ajantasaisuus. Ratkaistava tutkimusongelma on hakijaorganisaation tietojen ja muun prosessin mukaisen tiedon kerääminen useista tietolähteistä sellaiseen muotoon, joka palvelee Business Finlandilla teh-

tävää esitysvalmistelutyötä. Tutkimuskysymyksiä on kolme:

K1) Miten tietoja yhdistetään eri lähteistä yhdeksi kokonaisuudeksi?

K2) Kuinka ajan funktiona muuttuva tieto pidetään ajan tasalla?

K3) Miten kehitetään sovellusta sujuvasti ja mahdollistetaan jatkokehitys?

Luvussa 2 selvitetään teoriapohjaa tietojärjestelmien hyödyntämisestä ja toteutustekniikasta ylipäätään viraston hallinnollisten prosessien ja palveluprosessien toteuttamiseksi. Luvussa 3 käydään läpi lähtötilanne, joka vallitsi Business Finlandin prosesseissa ja rahoitusjärjestelmissä ennen projektin toteutusta. Seuraavaksi luvussa 4 selostetaan, minkälainen tuotos projektin aikana tuotettiin, eli miten järjestelmäkokonaisuus muuttui. Lisäksi esitellään pohdintaa erilaisista ratkaisuvaihtoehdoista ja perusteellaan lopullisia ratkaisuja. Luvussa 5 arvioidaan ratkaisujen järkevyyttä ja riittävyyttä, sekä toteutuksen laadun mittaamista.

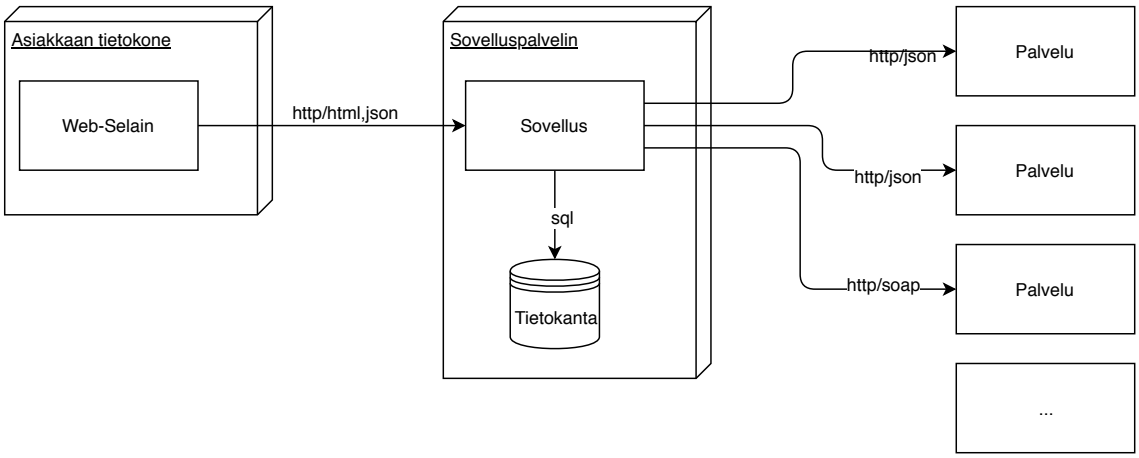
Työn merkittävin tulos on dokumentoida empiiristä aineistoa reaalimaailman sovelluskehitysprojektin ratkaisumalleista. Työ dokumentoi teollisuuden sovelluskehitysprojektin, jonka tilaajana on virasto. Dokumentointiin sisältyy arkkitehtuurilliset ratkaisut, joilla viranomaisprosessista aiheutuvat vaatimukset voidaan täyttää. Lisäksi työssä käsiteltävät kehityksen ratkaisut ja menetelmäkuvaukset osoittavat modernien ja ketterien menetelmien olevan hyödynnettyjä myös sellaisessa organisaatiossa, jonka tavallisesti ajatellaan olevan luonteeltaan byrokraattinen ja muutoskyvyltään jäykkä.

2 TIETOJÄRJESTELMÄT VIRASTON TOIMINTAPROSESSIEN TUKENA

Tässä luvussa selostetaan työn teoreettinen tausta. Se sisältää lähdekirjallisuuden avulla selvitettyä tietoa, joka on yleistä tietoa työn osa-alueen kannalta sekä oleellista työn toteutuksen ja analyysin ymmärtämiseksi. Luku ei esitä yksinomaisia tapoja ratkaista tietojärjestelmän vaatimuksia vaan pohjustaa tyypillisiä ratkaisuja, joita työn tekijä on havainnut sovelluskehitystyössä.

2.1 Verkkoympäristössä toimiva sovellus

Sovelluspalvelin on verkossa oleva tietokone, jossa suoritetaan ohjelmaa, joka palvelee asiakkaiden pyyntöjä. *Asiakasohjelma* on sovelluksen osa, jonka kanssa *käyttäjä* on suorassa interaktiossa. Pyynnöt noudattavat niin kutsuttua pyyntö/vastaus -mallia (request/-response) [30]. Kuvassa 2.1 on esitetty tyypillinen esimerkki web-sovelluksen arkkitehtuurista.



Kuva 2.1. *Web-sovelluksen arkkitehtuuri*

Palvelinjako voi vaihdella eikä ole oleellinen sovelluksen toiminnallisuuden kannalta. Sovelluspalvelimella olevat sovellukset ovat tehty palvelemaan asiakassovellusta tai toisia sovelluksia. Ohjelmakoodia voi suorittaa joko *käyttäjän* tietokoneella asiakasohjelmassa tai sovelluspalvelimella. Web-sovellus itsessään koostuu asiakkaan tietokoneella ajettavasta sovellusosasta sekä sovelluspalvelimella ajettava sovellusosasta. Käyttäjän tietoko-

neella suoritettavaa ohjelmaa kutsutaan front-endiksi kun taas sovelluspalvelimella suoritettavaa ohjelmaa back-endiksi. Front-endin sovelluskoodi palvellaan muiden asiakkaan pyyntöjen tapaan yleensä sivulle saavuttaessa. Kuvassa näkyvät palvelut voivat olla joko organisaation sisäisiä tai ulkoisia palveluita.

Perinteisessä palvelinmallissa sovelluspalvelin on yksi fyysinen tietokone, jolla voidaan ajaa yhtä tai useampaa sovellusta. Palvelin keskustelee muiden palvelimien kanssa tietoliikenneverkon välityksellä. Sovelluksia jaetaan ajettavaksi eri palvelimille kuorman tasaamiseksi sekä tietoturvan mahdollistamiseksi palomuurien avulla. Pilvipalveluympäristössä tietokoneet ovat loogisesti jaoteltu siten, että sovellusten näkökulmasta ne näyttäytyvät kuten perinteisessä palvelinmallissa. Pilvipalveluissa sovelluksia ajetaan virtuaalikoneissa tai ne tarjoavat valmiita sovelluksia tai sovellusalustoja [29]. Kuvassa 2.1 on kuvattu web-sovelluksen verkkoympäristö vain loogisella tasolla ja itse sovelluksen toiminnan kannalta oleellisin osin.

Web-selain sekä palvelinsovellus (kuvassa 2.1 sovelluspalvelimen sisällä oleva sovellus) käyttävät kommunikaatiossa tyypillisesti HTTP-protokollaa. Pyyntö kohdistuu tiettyyn resurssiin pyynnön mukana olevan resurssin tunnisteen (URI) mukaan. Protokollan mukaan pyynnöissä on verbi, joista neljä tyypillisintä on GET (hae), POST (julkaise), PUT (aseta) ja DELETE (poista). Näitä yhdistelemällä web-selaimelle saadaan haettua dataa web-sivun näyttämiseksi sekä web-selaimen käyttäjä voi muokata persistoitua dataa. Pyyntöjen vastauksessa on myös tilakoodi, joka kertoo pyynnön onnistumisesta ja sen seurauksista, tai epäonnistumisesta ja sen syystä. [12]

Http-pyynnön mukana hyötykuormana voi kulkea dataa, jonka tyyppi voi olla mitä vain. Hakupyynnön vastauksena voi tulla esimerkiksi kokonainen HTML-sivu tai pieni palanen esitettävää dataa JSON-muodossa. Perinteisesti web-sivut toimivat siten, että ensimmäisen hakupyynnön vastauksena kokonaisen web-sivun saatuaan käyttäjä täyttää lomakkeita tai siirtyy hyperlinkillä toiselle sivulle. Käyttäjän toimista aiheutuu uusi pyyntö palvelimelle. Palvelinsovellus käsittelee pyynnön, tekee mahdollisesti tiedon tallennuksia ja vastaa uudella kokonaisella web-sivulla. Toinen tapa toteuttaa web-sovelluksia on suorittaa sivulle saavuttaessa sivun alkulataus, joka sisältää web-sivun sekä selaimessa suoritettavan asiakasohjelman. Selainnäkymä muuttuu asiakasohjelman kautta eikä kokonaisia web-sivuja enää palvella kyseisen istunnon aikana. Selain suorittaa vain sellaisia pyyntöjä, joiden hyötykuormana on pelkkää asiadataa eikä lainkaan visuaalista dataa (esimerkiksi HTML-merkkaus). Pyyntöjen mukaan selaimessa suoritettava ohjelma muuntaa näkyvää web-sivua vastaamaan uutta tilaa. Tällaisia kutsutaan nimellä Single Page Application.

Web-sovelluksessa on syytä tehdä käyttäjän syötteen validointia. Syöte voi olla vääräss muodossa tai epäeheää aiheuttaen virhetilanteita. Validoinnilla voidaan sekä estää virheitä tai ohjata käyttämään antamaan pakollisia tietoja. Tiedon validoinnin voi suorittaa http-pyynnön mennessä back-endiin. Vastauksena tulee jonkinlainen virhekoodi tai -viesti, jos syöte on epäkelvollinen. Syötteen validoinnin voi tehdä myös front-endissä estämällä tiedon lähetyksen tai näyttämällä virheviesti käyttöliittymässä. Tehokkain tapa va-

lidoida käyttäjän syöte on suunnitella käyttöliittymä niin, että on käytännöllisesti katsoen mahdotonta syöttää virheellistä tietoa.

Palvelinsovellus myös kommunikoi muiden osapuolten kanssa asiakkaan pyyntöjen toteuttamiseksi synkronisesti tai hakiessaan tietoja asynkronisesti. Synkroninen tietojenhaku tarkoittaa, että asiakas odottaa pyynnön jälkeen vastauksen saapumista ennen käytön jatkumista. Asynkroninen taas tarkoittaa sitä, että asiakasohjelma tekee pyyntöjä käyttäjän tietämättä taustalla ja sovelluksen käyttäminen jatkuu pyynnöistä huolimatta. Kuvassa 2.1 on esitetty esimerkiksi pyynnöt tietokantaan sekä erilaisiin kolmannen osapuolen palveluihin. Palvelinsovelluksen tarkoituksena on peittää asiakkaalta tiedonhakujen yksityiskohdat ja moninaiset osallistuvat osapuolet ja tarjota yksittäinen, selkeä rajapinta sovelluksen toimintojen toteuttamiseksi.

Sovellusten välisessä kommunikaatiossa on varmistettava sovelluksen autentikaatio ja autorisaatio. Tietoverkossa voi olla paha-aikeisia toimijoita, jotka yrittävät päästä käsiksi niille kuulumattomiin henkilökohtaisiin tietoihin. Autentikaatiolla tarkoitetaan palvelun käyttäjän identiteetin toteamista ja autorisaatiolla kyseisen käyttäjän oikeutta käyttää tämän pyytämää palvelua. Identiteetin ja pääsynhallinnan palvelu tarjoaa toiminnon, joiden avulla palvelun käyttäjä autentikoituu ja palveluntarjoaja saa tiedon kyseisen käyttäjän identiteetistä. Palvelun käyttäjänä voi toimia sekä ihminen että järjestelmäkäyttäjä. [2]

Tietokannat ovat tapa pysyväistallentaa (persistoida) dataa tietojärjestelmissä. Tietokannassa on entiteettejä tallennettuna relaatioihin eli tauluihin, joissa on rivejä (monikko) ja attribuutteja (sarakkeet). Relaatiolla on pääavain, jonka avulla tiettyyn relaation riviin voi viitata toisesta relaatiosta. Pääavain voi koostua yhdestä sarakkeesta tai useamman sarakkeen yhdistelmästä ja sen on oltava uniikki. Riveille voi myös luoda sijaisavaimen, joka ei kuulu entiteetin tietosisältöön mutta helpottaa viittausta. Tietokannan sisältöä haetaan ja muokataan SQL-kyselyillä. Tietokannan päivityksessä yleensä päivitetään vain pientä joukkoa riveistä, jolloin päivityksessä rajataan operaation kohdistumaan jonkin attribuutin arvoihin. Suuren datamäärän päivitys tai lisäys kerrallaan onnistuu eräpäivityksellä, jossa yhdelle kyselylle asetetaan joukko parametreja ja sama kysely suoritetaan tietokantajärjestelmässä useasti peräkkäin sen sijaan, että tietokantaa käyttävä sovellus tekisi erillisen kyselyn jokaista päivitettävää riviä kohden. Tietokannat voidaan eristää verkkoympäristössä omille palvelimille, jolloin pysyväistallennuksen varmistaminen onnistuu varmistamalla kyseisten palvelinten saatavuus. Saatavuudella tarkoitetaan sekä tiedon olemista haettavissa mahdollisimman suuren osan ajasta sekä tiedon häivämisen välttämistä häiriötilanteissa. [14]

Verkkoympäristö mahdollistaa toiminnanohjausjärjestelmän (engl. Enterprise Resource Planning, ERP) toteuttamisen. Toiminnanohjausjärjestelmä on organisaation laajuudella käytettävä sovellus, joka tarjoaa toimintoja organisaation liiketoiminnan ohjaamiseksi ja käyttää dataintegraatioita tiedon tuomiseksi keskitetysti yhteen paikaan [4]. Järjestelmä voi olla yksittäinen sovellus, joka kattaa kaikki organisaation toiminnot. Toiminnot voi myös jakaa useampaan sovellukseen, jolloin ne ovat kuitenkin edelleen organisaation laajuudella keskitettyjä.

2.2 Dataintegraatio ja integraatioarkkitehtuuri

Doan et al. tiivistävät *dataintegraation* tavoitteeksi tarjota yhdenmukainen pääsy joukkoon riippumattomia ja heterogeenisiä tietolähteitä [7, s. 6]. Tiedon lähteitä on monia ja kukin käyttää erilaisia rakenteita mallintaakseen tietoa. Tietojen rakenne ja saatavuus voi muuttua arvaamatta. Tällaisiin ongelmiin on tietojärjestelmien toteutuksessa varauduttava sekä ohjelman toiminnallisuuden kannalta että sovelluskehityksen johtamisen kannalta [13]. Toisaalta virastojen tarjoamat datarajapinnat eroavat avoimista tietolähteistä siten, että virastot ovat velvollisia lain nojalla tarjoamaan tiettyjä tietoja ja varmistamaan sen saatavuus. Kuitenkin datan muoto voi erota toisten virastojen tarjoamaan dataan verrattuna ja rajapinnat voivat muuttua, joten datan sovittamista omaan sovellusalueeseen on suoritettava.

Datan integraatio tarkoittaa perustasolla kyselyä, jossa asiakas tekee haun tietyillä kriteereillä ja integraatiokohde eli palvelu vastaa toivotulla datajoukolla, joka vastaa datan tilaa palvelussa kyselyn hetkellä. Kyselyiden monimutkaistuessa syntyy suorituskykyongelmia, jos datan prosessointi on monimutkaista ja dataa on paljon. Kyselyjen kesto voi olla niin suuri, että asiakkaan toiminta käy vaivalloiseksi. Lisäksi eri ajanhetkellä kyselty data voi antaa erilaisia tietoja, kun tieto palvelussa muuttuu. Tästä seuraa, että tieto voi näyttäytyä eri järjestelmissä erilaisena, jolloin tiedosta riippuva prosessi menee sekaisin. Tällaisten ongelmien ratkaisemiseksi on kehitettävä ratkaisuja tietoa koostavassa järjestelmässä [16].

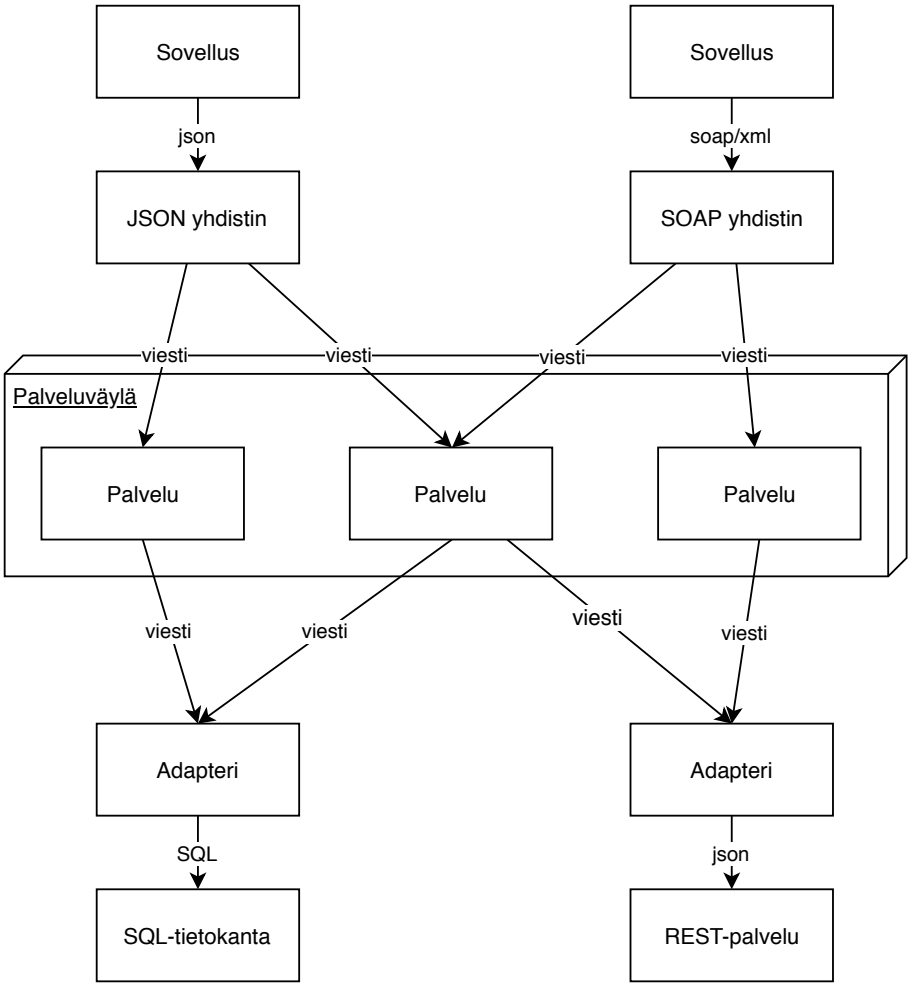
Integraatiossa voidaan noudattaa REST-periaatteita [11], jolloin data on tyypillisesti muotoa JSON tai XML [7]. Pyyntöjen ja vastausten hyötykuormana on rakenteellista dataa. Integraatiota voi toteuttaa myös SQL-pyynnöin. Mitä hyvänsä siirto- ja sovellusprotokollia integraatioissa käytetään, kyse on kuitenkin tietoliikenneverkkojen läpi kulkevasta liikenteestä, jolla tietoa siirretään tietokoneelta toiselle. Tieto liikkuu sieltä, missä se on saatavailla, sinne, missä sitä tarvitaan. Sovelluskehityksen tarkoituksena on kirjoittaa sovelluksen logiikka, joilla näytä tiedonsiirtoja ohjaillaan, sekä sovelluksen osat, joka sovittaa tiedot tarkoituksenmukaiseen muotoon.

REST:n esivaatimukseksi on ehdotettu Richardsonin kypsyysmallia (Richardson Maturity Model) [28]. Mallissa on kolme tasoa: resurssit, verbit ja löydettävyys. Yhden palvelupisteen sijaan palvelun resursseihin on pääsy omista osoitteistaan kommunikaatioprotokollan osoitemekanismin avulla. Resursseihin kohdistuvat operaatiot eri verbeillä ovat ennalta sovittuja. Samanlaiset operaatiot toimivat aina samalla tavalla. Löydettävyys tarkoittaa, että palvelimen vastauksessa kerrotaan, mitä resursseja asiakas voi käsitellä seuraavaksi eli palvelu on itsensä dokumentoiva. REST ei määrittele protokollia, vaan ajatusmallin tilan hallintaan web-palveluissa. HTTP on yksi protokolla, jolla REST voidaan toteuttaa. Organisaation sisäisissä integraatioissa ei yleensä noudateta kolmatta kypsyystasoa. Asiakassovellus olettaa tiettyjen resurssien löytyvän tietyistä osoitteista, eikä palvelinsovellus palauta resurssien sijainteja, tai asiakassovellus ei tulkitse niitä. Uusien luotujen resurssien tunnisteet muokkaavien operaatioiden tuloksena sen sijaan palautetaan, mikä on

välttämätöntä niihin viittaamiseksi asiakassovelluksessa käytön aikana.

Integraatiolähteiden tietojen sovittamiseen yhtenäiseen muotoon yksi ratkaisu on *palveluväylä* eli Enterprise Service Bus (ESB) [5]. Kuvassa 2.2 on esimerkkiarkkitehtuuri palveluväylästä ja siihen liitetyistä osapuolista. Palveluväylän pääperiaatteisiin kuuluu hienojakoisten operaatioiden tai palveluiden koostaminen suuremmaksi palveluksi, joka peittää toteutuksen ja tietomallin yksityiskohtia. Väylä toimii viestinvälittäjän muuntamalla sisään tulevat viestit omaan sisäiseen muotoonsa, minkä jälkeen palvelun ulostulosta tuleva data muutetaan taas palvelun käyttäjän pyytämään muotoon.

Palveluväylä tarjoaa palvelun käyttäjille universaalin tavan kommunikoida keskenään. Väylä sovittaa (adapteri) integraatiokohteen tarjoaman datan sen alkuperäisestä muodosta sellaisiin, joita tiedon käyttäjä ymmärtää. Palveluväylä toimii sovelluksille ikään kuin yhtenä "palvelutiskinä", josta voi palvelun nimellä ja palvelun vaatimilla tiedoilla saada vastauksen pyytämässään muodossa. Palvelun tarkoitus näyttäytyy selkeänä ja yksinkertaisena, mutta sen toteuttamiseksi voi tapahtua useampia askeleita ja tietopyyntöjä.



Kuva 2.2. *Esimerkki palveluväyläarkkitehtuurista [7]*

Integraatioissa ajan kuluessa muuttuvan datan siirtäminen oikeaan aikaan vaatii, että tietojärjestelmien on kommunikoitava myös tietoa itse tiedon tilasta asynkroniesti. Monesti tiedon muuttuminen yhdessä järjestelmässä kiinnostaa myös toista järjestelmää, jolloin

tästä muutoksesta on ilmoitettava toiselle järjestelmälle. Tähän ratkaisuksi tiedosta kiinnostunut järjestelmä voi kysellä tietyin aikavälein joko koko kyseessä olevan datajoukon tai tiedon datajoukon tilasta, kuten päivitysajasta tai laajuudesta. Tämä aiheuttaa kuitenkin paljon liikennettä ja tarpeettomia pyyntöjä. Niinpä tehokkaampi ja elegantimpi ratkaisu on julkaise/tilaa -malli (publish/subscribe) [9], jossa datasta kiinnostuneet osapuolet tilaavat tietyntyyppisiä tapahtumia ja datalähteet julkaisevat näitä tapahtumia. Viestijono välittää julkaistut viestit niiden tyypin perusteella niistä kiinnostuneille osapuolille. Julkaisija ja tilaaja eivät tunne toisiaan.

Integraatiopalveluita ei kannata suunnitella vastuualueltaan liian suuriksi. Integraatioarkkitehtuurissa voi soveltaa suosiotaan kasvattaneita mikropalveluita, jotka ovat pieniä yhden vastuualueen sovelluksia, joita on helppo jalkauttaa, skaalata ja testata toisistaan riippumattomasti [34][37]. Integraatioissa mikropalvelut voivat olla muuntimena oman organisaation ja ulkoisen osapuolen palvelun välissä tai yhdistää useampi ulkoinen lähde yhdeksi palveluksi datan luonteen mukaan. Esimerkiksi kirjanpitodatan yhdistäminen useammasta lähteestä ja niiden tarjoaminen yhtenäisessä muodossa voi olla yhden mikropalvelun vastuulla. Pienemmät ohjelmanpalaset ovat nopeampia ja turvallisempia jalkauttaa tuotantoympäristöön, jolloin uusien toimintojen toimittaminen käyttöön toteutuksen alkamisesta tapahtuu pienemmällä viiveellä. Myös sovelluksien kehittäminen on helppoa jakaa eri tiimien kesken, kun palvelut ovat selkeästi erillisiä yksiköitään [37].

Organisaation sisällä ei välttämättä tarvitse protokollaeroja hävittävää viestiväylää, koska sovellusten välisen kommunikaation käytännöt ovat sovittavissa. Sovellukset voivat integroitua suoraan toisiinsa hyvin pienellä vaivalla, kun protokolla ja tietomallit ovat yhtenäiset. Esimerkiksi JSON-muotoinen data REST-rajapinnan avulla riittää organisaation sisäisissä järjestelmissä, kun JSON-datan rakenne on ennalta sovittu ja dokumentoitu integroituvien järjestelmän kehittäjiä varten.

Integraatioiden tehokkuuden parantamiseksi kannattaa hyödyntää välimuisteja. Välimuisti on tietosäilö, johon integraatiokutsujen vastaukset laitetaan talteen uudelleenhyödynnettäväksi. Välimuisti voi sijaita joko tietoa käyttävän osapuolen tietokoneella tai jollakin palvelimella verkossa käyttäjän ja integraatiokohteen välillä. Data tallennetaan siten, että tietyllä avaimella on mahdollista noutaa siihen liitetty dataresurssi. Avain voi olla esimerkiksi tietokannassa synnytetty avain tai jonkinlainen laskettu tiiviste alkuperäisen pyynnön tiedoista. Välimuistin tarkoitus on perustasolla lyhentää kyselyjen kestoa merkittävästi hyödyntäen sitä, että vanhempaa dataa voi uudelleenhyödyntää, koska uusi kysely ei toisi minkäänlaista uutta tietoa.

Sovellustason välimuisti mahdollistaa hienojakoisemman tiedon rakenteensa asioista paremmin soveltuvat välimuistirakenteet [22]. Sovellustasolla kirjoitetaan ohjelmakoodiin tiettyihin paikkoihin tallennuksia ja hakuja välimuisteihin. Näin voidaan hallita paremmin, mitä tietoja halutaan välimuistissa säilöä ja milloin niitä halutaan päivittää.

2.3 Aikariippuvaisen tiedon haku

Tiedonhaun ajoitusratkaisuiksi voidaan tunnistaa tässä kohdassa esitettyjä menetelmiä [13][16][22][34]. Tietyn integraatioketjun syntymäkohtaa kutsutaan tässä *alkusovelluksek-si*. Alkusovellus on paikka, jossa tiedon tarve syntyy.

Haetaan tieto, kun käyttäjä tarvitsee: Alkusovellus hakee tiedot integraatiokohteista aina, kun niitä on kyseltävä jonkin pyynnön toteuttamiseksi. Integraatiokohteet voivat myös tehdä edelleen integraatiokutsuja ketjussa. Tämä tarkoittaa, että esimerkiksi käyttöliittymässä etusivun avaamisen lisäksi navigoidessa alisivuille tehdään aina tiedonhaku, joka saa aikaan myös tiedonhakuja integraatiokohteisiin.

Hyvänä puolena on, että tieto on aina käytännössä ajantasaista, kun se esitetään. Tieto voi muuttua lähteessä välittömästi tiedonhaun jälkeen, mutta tämä on käytännössä mahdoton skenaario välttää kokonaan tiedon ollessa hajautettuna. Lisäksi tiedolle ei tarvitse rakennella monimutkaisia välimuistirakenteita.

Huonona puolena on, että tämä kuormittaa integraatioita ja voi helposti aiheuttaa suorituskykyongelmia suuren siirrettävän datamäärän johdosta. Myös suuri käyttäjämäärä tai samassa verkkoympäristössä joukko muita järjestelmiä toimimassa samanaikaisesti voi kuormittaa integraatiopalveluita. Tämä voi aiheuttaa myös niin kutsutun 1+n -ongelman, eli yhden kyselyn vastauksena saadun joukon alkioiden mukaan tehdään yksitellen lisää kyselyjä, jos integraatiokohteet eivät mahdollista tietojen kyselyä useamman tunnisteen perusteella.

Haetaan tieto tarvittaessa ja tallennetaan välimuistiin: Sama kuin edellinen, mutta rakennetaan integraatioyhteyksiin sopivia välimuisteja. Välimuisti sisältää integraatiokohteen vastaukset tiettyihin pyyntöihin pyynnön parametrien perusteella. Seuraava pyyntö samoilla parametreilla palauttaa välimuistissa olevan tiedon, eikä alkuperäinen integraatiokohde rasitu sen jälkeen lainkaan, kunnes välimuistin tieto on vanhentunut.

Välimuistien hyvä puoli on, että integraatiopalveluja sekä verkkoyhteyksiä kuormittavaa tietoliikennettä saadaan vähennettyä merkittävästi. Välimuisti myös nopeuttaa alkusovelluksen toimintoja, koska tiedon hauissa tieto löytyy paljon nopeammista muistirakenteista kuin integraatioyhteydet. Huonona puolena on, että tiedon muuttuessa alkuperäisessä lähteessä välimuistissa on vanhentunutta tietoa.

Haetaan tietoa asynkronisesti ja tallennetaan tieto sovellukseen: Haetaan tietoja ajastetusti käyttäjien toiminnasta riippumatta ja tallennetaan tieto sovellukseen joko muistipohjaiseen välimuistiin tai tietokantaan. Tiedon haun voi ajastaa esimerkiksi siten, että tietoa haetaan tiheämmin virka-aikaan, kun käyttäjät tarvitsevat todennäköisemmin tuoretta tietoa.

Hyvänä puolena on, että tieto on suoraan sitä käyttävässä järjestelmässä aina lähes ajantasaista. Lisäksi tiedon siirto käyttäjälle on nopeaa, sillä ei tarvitse odotella mahdollisesti hitaiden integraatiokutsujen valmistumista, sillä tieto tulee suoraan alkusovelluksesta.

Huonona puolena on, että tehdään monesti turhia integraatiokutsuja. Tieto muuttuu yleensä verrattain harvoin siihen nähden, kuinka usein sitä tarvitaan. Esimerkiksi tiedon muuttuessa kerran päivässä tuntemattomana kellonaikana, on tietoa haettava siitä huolimatta suhteellisen tiheästi. Tiedon pitäminen ajantasalla vaatii kohtuullisen lyhyen aikavälin kyselyiden välille.

Kuunnellaan viestejä muuttuneesta tiedosta: Suunnitellaan lähdejärjestelmä siten, että se julkaisee viestin, kun tieto muuttuu. Tietoa käyttävät järjestelmät ottavat viestin vastaan ja reagoivat siihen esimerkiksi tallentamalla viestin sisällön itselleen tai hakemalla lisätietoa.

Hyvänä puolena on, että turhia integraatiokutsuja ei tehdä lainkaan. Alkusovelluksen tehtyä alkulatauksen, tietoa ei liiku sovellusten välillä ennen kuin se on muuttunut. Huonona puolena on, että tämä vaatii lähdejärjestelmän toteutukseen muutoksia. Lähdejärjestelmän ollessa kolmannen osapuolen hallinnassa ominaisuuksien lisääminen siihen on hyvin hankalaa tai mahdotonta. Lähdejärjestelmän on myös varmistettava, että jokainen tiedon käyttäjä on varmasti saanut uuden tiedon.

Näytetään olemassa oleva tieto, ja oletetaan tiedon olevan lopulta yhtenäinen: Haetaan tieto kerran ja tallennetaan se sovellukseen. Käyttäjän kysyessä tietoa jatkossa tieto tulee suoraan alkusovelluksesta. Seuraavan kerran tietoa haetaan integraatiokohteesta joko määrittelemättömän ajan kuluttua tai tietyssä prosessin vaiheessa. Esimerkiksi ajastettu tiedonhaku voidaan tehdä harvemmin, ja järjestelmä voi ohjata käyttäjää tekemään tarkistuksen myöhemmin.

Hyvänä puolena on, että tämä kuormittaa tietoverkkoja mahdollisimman vähän. Alkulatauksessa saatu tila on se, jonka nojalla prosessi etenee ja prosessin perusteena voidaan esittää tietyn ajanhetken tila. Jos prosessissa on rinnakkaisia vaiheita, joiden suoritusjärjestyksellä ei ole väliä, ne voidaan sarjallistaa. Kunkin vaiheen alussa voidaan hakea siihen oleelliset tiedot eikä muita tiedonlähteitä tarvitse rasittaa.

Huonona puolena tässä on, että prosessin eteneminen voi riippua tiedon tilasta. Tietyn tietokentän arvon poiketessa hyväksyttävästä prosessin eteneminen voi muuttua merkittävästi tai katketa kokonaan. Tässä tilanteessa virkailija voi tehdä turhaa työtä, koska lähtöoletukset olivat väärät.

2.4 Sovelluskehitys

Tiedon mallintaminen tietojärjestelmissä, eli domain-mallintaminen, tarkoittaa ulkoisen maailman entiteettien mallintamista ja prosessoimista tietokoneella. Domainiin voi kuulua luonnonilmiöt, organisaatioiden toiminnot tai tiedon rakenteet. Domain-malli auttaa ymmärrystä domainin eli itse sovellusalueen rakenteesta ja käyttäytymisestä sekä sitä kautta tietojärjestelmien vaatimuksista. Domain-mallista voidaan johtaa tietojärjestelmien toteutuksien instansseja. Domain-mallin elinkaari voi ylittää sovelluksien elinkaarien rajat.

Integraatioissa yhteinen sopimus domain-mallista helpottaa integraatioiden toteutusta ja tekee oletukset datan rakenteesta turvallisemmaksi. [8]

Sovelluskehityksen tueksi voi käyttää jatkuvaa integraatiota (Continuous Integration, CI), jota ajetaan palvelimella. Jatkuva integraatio tarkoittaa eri asiaa kuin datan integraatio. Jatkuvassa integraatiossa sovelluskehittäjien muutoksia liitetään jatkuvasti kehityksen alla olevaan sovellukseen [25]. Kun ohjelmoija siirtää koodia versionhallintaan, CI-palvelin huomaa muutokset ja käynnistää *työn*. Työ suorittaa automaattisesti sovelluksen käännöksen, testien ajon sekä koodin staattinen analyysin. Työn voi myös käynnistää manuaalisesti. Työ rakentaa myös sovelluksesta julkaistavan ja asennuskelpoisen paketin. Työ voi myös suorittaa sovelluksien asennuksia kehitysympäristöön, testiympäristöön tai tuotantoympäristöön. Automatisoidusta ratkaisusta, joka tekee asennuksia, käytetään myös nimitystä jatkuva toimitus (Continuous Delivery, CD) [17].

3 LÄHTÖTILANNE BUSINESS FINLANDIN RAHOITUSJÄRJESTELMISSÄ

Tässä luvussa pohjustetaan, millä tavalla Business Finlandin rahoitusprosessi toimii tämän työn kannalta oleellisin osin sekä miten sitä tukevat tietojärjestelmät toimivat lähtötilanteessa. Lisäksi selvitetään minkälaisia kehitystarpeita ja ongelmia niissä on sekä miten niitä tulisi parantaa ja tehostaa jatkossa. Luku pyrkii selittämään motivaation työssä toteutetulle järjestelmälle.

3.1 Rahoitusprosessi

Business Finlandin tehtävä on neuvoa, sparrata ja taloudellisesti avustaa uutta liiketoimintaa, tuotekehitystä ja innovointia edistäviä, suomalaisten yritysten ja tutkimuslaitosten suorittamia hankkeita [3]. BF itse kuvailee toimintaansa seuraavasti: “Autamme yrityksiä muuttamaan kehittämiskelpoisen idean liiketoiminnaksi tarjoamalla rahoitusta ja asiantuntijapalveluja.” [24]. BF:n palveluihin kuuluu siis muutakin kuin rahan myöntäminen; on kyettävä tunnistamaan markkinoiden luonnetta sekä trendejä, jotta voidaan myös tunnistaa asiakkaan, (eli yrityksen) kasvupotentiaali liiketoiminnan kannalta. BF auttaa yritystä mm. mahdollisten yhteistyökumppanien löytämisessä sekä sopivien kehityshankkeiden tunnistamisessa ja käynnistämisessä. Kun asiakas on löytänyt sopivan hankkeen, tekee tämä rahoitushakemuksen hanketta varten. Toisinaan asiakkuus saattaa alkaa suoraan rahoitushakemuksesta, mutta tällaisissa tilanteissa on todennäköisempää, että hakemuksen hylätään tai sen käsittelyaika on pidempi.

BF tarjoaa erilaisia rahoituspalveluita. Rahoituspalvelut on tuotteistettu joko sisällön tai tarkoituksen mukaan. Rahoituspalveluiden sisältöön vaikuttaa lainsäädäntö ja muun muassa EU-komission asetukset. BF tarjoaa asiakasorganisaation elinkaaren eri vaiheissa eri tyyppisiä rahoituspalveluita.

Rahoitushakemus tehdään BF:n sähköisessä asiointipalvelussa. Hakemuksen alussa hakija ohjataan yrityksen koon mukaan valitsemaan hakijan organisaatiolle tarkoitettu rahoituspalvelu. Yrityksen koko ohjaa hakemuksen aloitusvaiheessa asiakkaille tarjottavia rahoituspalveluita. Sen jälkeen toisessa vaiheessa hakija täyttää rahoituspalvelusta riippuvan määrän erinäisiä vapaita tekstejä sekä valintoja tai taulukoita. Hakemuksen lähetyksessä tapahtuu laillinen hallintoasian (myös *rahoitusasia*) vireillepano, jolloin BF:llä on

oikeus hakea hakijaan liittyviä taloustietoja.

Hakemuksen lähetyksen jälkeen alkaa *esitysvalmisteluvaihe*. Tässä vaiheessa BF:n esitysvalmistelija ryhtyy tuottamaan *rahoitusesitystä*. Esitysvalmisteluvaiheeseen kuuluvat myös hakemuksen täydennykset. Jos esitysvalmistelija havaitsee, että jokin tieto ei ole riittävän kattava rahoitusesityksen muodostamiseksi, pyytää tämä täydennystä hakijalta. Esitysvalmistelussa kerätään taloustietoja hakijaorganisaatiosta ja niiden perusteella muodostetaan esitys hakijan hankkeen rahoituksesta.

Esitysvalmistelun valmistuttua rahoitusasia siirretään päätettäväksi, jossa se joko hyväksytään, hylätään tai palautetaan esitysvalmisteluun, jolloin sen voi täydennyksin siirtää jälleen päätettäväksi. Hyväksytyn päätöksen jälkeen asia siirtyy asiakkaalle rahoitusehtojen hyväksymistä varten ja sen jälkeen *seurantavaiheeseen*, jossa pääpiirteinen tarkoitus on seurata hankkeen onnistumista ja asetettujen tavoitteiden saavuttamista. Näiden perusteella maksetaan hakijalle rahaa tämän esittämien tositteiden perusteella. Seurantavaiheessa rahoitukseen voi hakea myös muutoksia sähköisen palvelun kautta.

3.2 Sovellustoimittajan konteksti rahoitusprosessin järjestelmissä

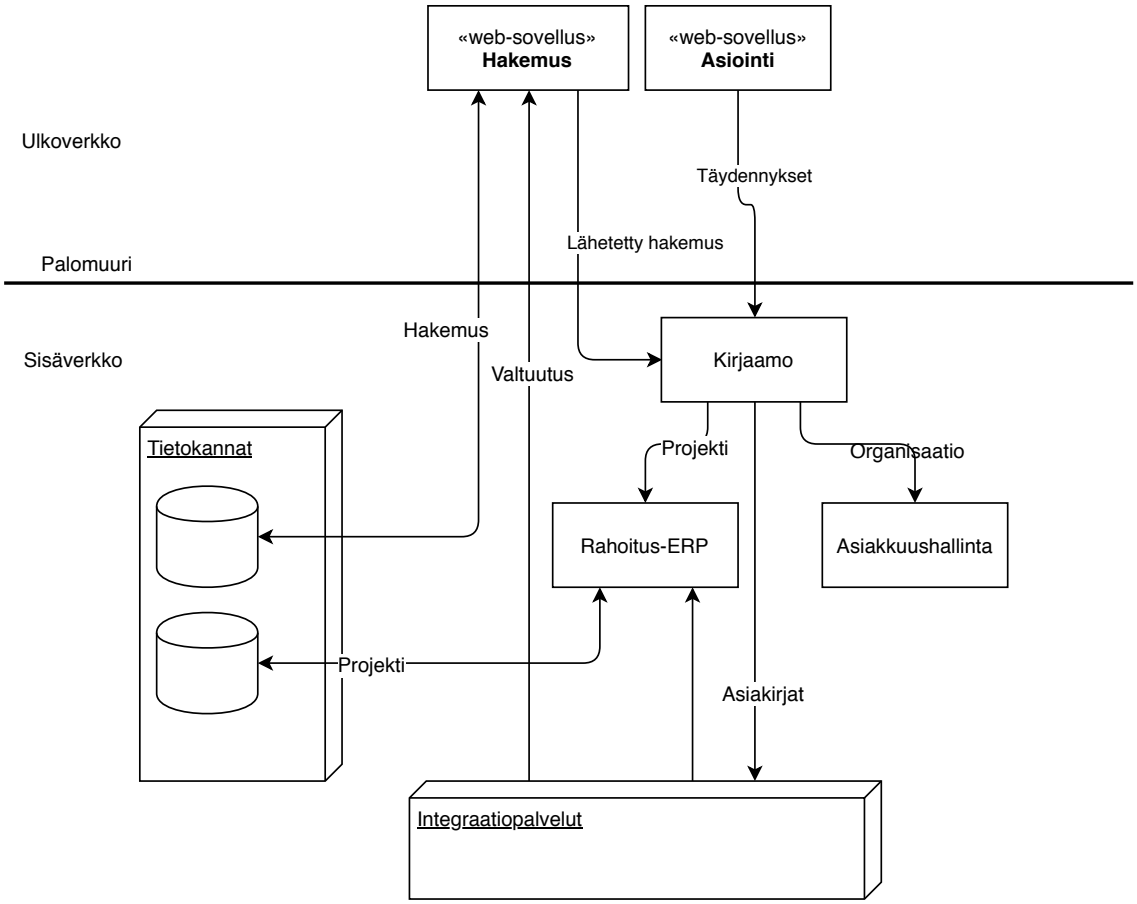
Solita ylläpitää ja jatkokehittää BF:n rahoitusprosessin tietojärjestelmiä. Tietojärjestelmien on tuettava sekä rahoitusprosessin operatiivista toimintaa, että tietojen tarjoamista raportointiin päätöksentekoa ja analytiikka varten.

Kuvassa 3.1 on esitetty pelkistetysti, minkälaisia osia rahoitusjärjestelmäkokonaisuudessa on, missä osat sijaitsevat verkossa loogisella tasolla ja miten osat vuorovaikuttavat keskenään. *Ulkoverkko* tarkoittaa tietoliikenneverkon aluetta, jonne on pääsy julkisesta internetistä. *Sisäverkko* on suojattu palomuurilla. Ulkoverkon sovelluksiin tunnistaudutaan kansallisella tunnistautumispalvelulla, ja sisäverkon sovelluksiin BF:n työntekijöitä ja konsultteja varten on oma identiteetin- ja pääsynhallintaratkaisunsa [18].

Rahoitusjärjestelmäkokonaisuuteen kuuluu käyttöliittymällisiä, web-pohjaisia käsittelysovelluksia, joiden avulla loppukäyttäjä vaikuttaa operatiivisten tietojen kanssa. Osa websovelluksista käyttää myös SQL-tietokantaa [23], johon tämä tallentaa juuri kyseisen sovelluksen tarvitsemia operatiivisia tietoja. Tietokannat on varmistettu saatavuuden kannalta.

Kokonaisuudessa on myös integraatiosovelluksia, jotka tarjoavat koneluettavassa muodossa rajapintoja sovelluskokonaisuuden kannalta yleiskäyttöisen datan lukemiselle ja tallentamiselle. Data voi olla itse tuotettua ja ylläpidettyä tai kolmannen osapuolen tarjoamaa. Osa integraatiopalveluista toimii muuntajana BF:n domainin datarakenteiden ja kolmannen osapuolen datarakenteiden välillä.

Asia on rahoitushakemuksesta syntyvä yksittäinen kokonaisuus, joka etenee rahoitusprosessin eri vaiheiden läpi. Eri prosessin vaiheissa ja konteksteissa se saa eri nimityksiä.



Kuva 3.1. *Rahoitusjärjestelmien pelkistetty, looginen sovellusarkkitehtuuri ja datavirrat sovellusten välillä hakemusvaiheessa*

Hakemuksen luonnissa hakemukselle varataan *diaari*, joka toimii asian tunnisteena. Hakemus täytetään ja lähetetään. Kirjaamo vastaanottaa hakemuksen tiedot, minkä jälkeen kirjaamon käyttäjä luo niistä Rahoitus-ERP:iin projektin, jonka tunnisteena toimii sama diaari.

Aloitustilanteessa projektin (asia) päädyttyä Rahoitus-ERP:in, ei sitä sieltä poisteta eikä sen käsittelyä jatketa muissa sovelluksissa. Rahoitus-ERP saa asiakkaan Asiointi-sovelluksessa täyttämiä sisältöraportteja ja tilityksiä projektissa/hankkeissa toteutuneista kustannuksista, joiden perusteella virkailijat toimeenpanevat maksuja. Tässä vaiheessa on paljon integraatioita erilaisten pankkien ja kirjanpitojärjestelmien kanssa.

3.3 Tavoitetila

Rahoitusjärjestelmät ovat tuotantokäytössä ja ovat yleisesti ottaen täysin käyttökelpoiset BF:n rahoitusprosessia varten tähän mennessä, mutta kehittämisen kohteita on. Tekesin sovelluksien ylläpidon ja kehityksen tarjouspyynnön mukaan BF:n tavoitteena on muun muassa kehittää järjestelmiään ketterästi [24]. Sovelluskehityksen sekä sovellusten asennuksen on toimittava nopeilla jaksoilla.

Kehitettävien järjestelmien muutosten tulee noudattaa palvelupohjaista arkkitehtuuria [24]. Sovelluksien vastuualueita on pienennettävä ja suurien sovelluksien toimintoja jaettava pienemmiksi sovelluksiksi. Asioiden keskittyminen samaan tietomalliin prosessin eri vaiheissa on johtanut monimutkaiseen ja vaikeasti hallittavaan tilanteeseen. Vaatimuksiin kuuluu myös järjestelmien asennettavuuden parantaminen ja sen ketteryys tuotantokäyttöä häiritsemättä. Tämän mahdollistamiseksi asennettavien yksiköiden kokoa on pienennettävä, ja niiden testattavuutta on parannettava. Pienemmät sovellusyksiköt tukevat myös järjestelmien skaalautuvuutta, sillä niiden yhdisteleminen ja monistaminen helpottuu. Suurien kokonaisuuksien hahmottaminen on vaikeaa. Kestää kauan, ennen kuin uudet sovelluskehittäjät oppivat järjestelmän toiminnan. Tämä hidastaa huomattavasti uusien ominaisuuksien kehittämistä sovelluksiin.

Sovellukset on toteutettu vanhentuneilla tekniikoilla ja koodin rakenne on tullut vaikeasti ylläpidettäväksi. Ne edellyttävät modernisointia ja refaktorointia [24]. Modernit teknologiat nopeuttavat kehitystyötä, sillä kehittyneellä teknologioilla päästään helpommin tarkoituksenmukaiseen lopputulokseen. Vanhentuneiden teknologioiden osaaminen yleisesti vähenee, kun asiantuntijat siirtyvät käyttämään uudempia. Vanhentuneiden teknologioiden tekninen tuki vähenee tai saattaa loppua kokonaan, jolloin sovelluksien tietoturva alkaa järkkymään. Myös sovellusten vanhentunut ulkoasu ja käytettävyysongelmat vaivaavat käyttäjiä.

BF haluaa parantaa järjestelmien testattavuutta ja automatisoida rutiinitestausta [24]. Testausautomaatio on hyvä työkalu sekä sovelluskehittäjän työssä että järjestelmien laadun varmistamisessa. Testausautomaatiolla mahdollistetaan myös regressiotestaus. Oikeanlainen sovellusarkkitehtuuri tukee testattavuutta.

BF:llä esiintyy myös organisaatiorakenteen muutoksesta johtuvia vaatimuksia. Vanha Asiakkuudenhallinta on korvattava kaikille TEM:n alaisuudessa toimivien yhteisellä CRM-järjestelmällä. BF:lle oleellinen asiakasdata on vääjäämättä eri muotoista, kuin rahoitusjärjestelmissä on ennestään oletettu, eikä BF:lle erityistä dataa välttämättä ole saatavilla lainkaan. Tähän on kehitettävä uudenlaisia integraatioratkaisuja, joita muiden uudistettavien sovelluksien on hyödynnettävä. Rahoitussovelluksissa on suoritettava rahoitustyön lisäksi hallinnollisia työvaiheita. Näitä on mahdollista automatisoida tietojärjestelmillä, mistä syntyy vaatimuksia sovellusarkkitehtuurin suunnittelulle. Lisäksi tekoälyratkaisujen hyödyntäminen ja niille opetusdatan tarjoaminen on otettava huomioon.

Osa tavoitteista on jo saavu tettu. Esimerkiksi Hakemus-sovelluksen toteutus on osa julkiverkkoon näkyvän asioinnin uudistamista. Esitysvalmistelu on yksi uudistusprojekteista, joiden tuloksena päästään lähemmäksi tavoitetilaa.

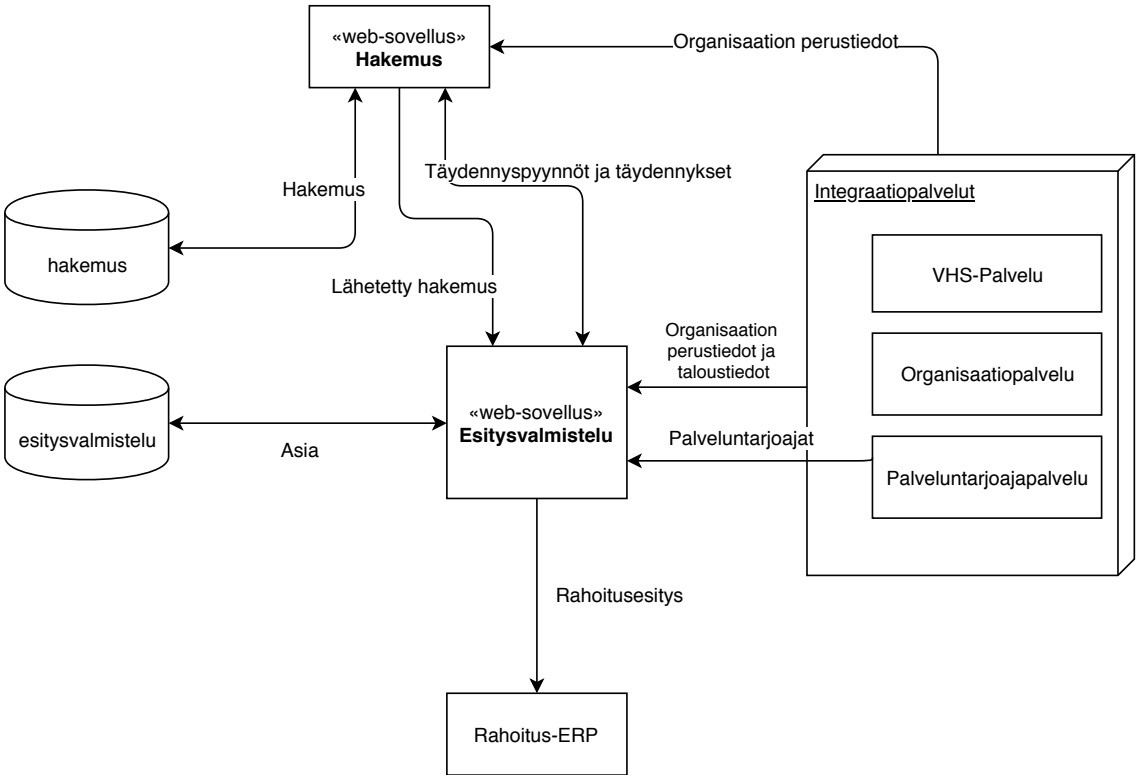
4 TOTEUTETTU JÄRJESTELMÄ

Tässä luvussa esitellään tarkempi kuvaus tuotetusta muutoksesta järjestelmäkokonaisuuteen. Kuvauksiin sisältyy tiedon tarve ja sitä kautta johdetut sovellusarkkitehtoniset ratkaisut. Luvussa vertaillaan luvussa 3 esitettyyn lähtötilanteeseen. Tuodaan esille myös vaihtoehtoisia ratkaisuja sekä perusteellaan niitä puolestaan tai vastaan, sekä luetellaan yksityiskohtaisia jatkokehitysideoita. Luvussa havainnollistetaan tiedon etenemistä kuvitellun esimerkin startup-yrityksen Spagee Consulting hakemuksesta otsikolla Joulupukin uudet vaatteet.

Kuvassa 4.1 on esitetty *EV*-projektin (Esitysvalmistelun uudistus) aikana tuotetun järjestelmäkokonaisuuden osan järjestelmäarkkitehtuuri ja tietovirrat. Keskeisimpänä tuotoksena on uusi *Esitysvalmistelu-sovellus*. Esitysvalmistelu-sovellus on täysin uusi osa järjestelmäkokonaisuutta mutta suurin osa siihen liittyvistä sovelluksista on toteutettu osana aikaisempia projekteja. Liittyviin sovelluksiin tehtiin kuitenkin EV-projektin aikana muutoksia Esitysvalmistelu-sovelluksen tarpeiden pohjalta. Sovelluksien jatkokehityskelpoisuus on otettava huomioon sovelluskehityksessä.

Pääpiirteittäin erona aikaisempaan arkkitehtuuriin on asian eteneminen Hakemuksesta Esitysvalmistelu-sovellukseen sekä Asiakashallinnan ja Kirjaamon poistuminen. *Palveluntarjoajapalvelu*n sekä muiden integraatiopalveluiden toiminnot on kuvattu yksityiskohtaisesti myöhemmin. Tavoitetilassa ei ole enää integraatiota Asiakashallintaan, vaan arkkitehtuurissa varaudutaan tulevaan CRM-muutokseen integroitumalla kehityksen alla olevaan Organisaatiopalveluun. Organisaatiopalvelu on toisen projektitiimin tuottama, mutta vaikuttaa oleellisesti myös Esitysvalmistelun toteutukseen.

Esitysvalmistelu-sovelluksen lopullinen tarkoitus on toimia yksinomaisena käyttöliittymänä rahoitusesityksen muodostamiseen. Tässä vaiheessa toteutettiin kuitenkin vain yhden rahoituspalvelun kattava sovellus. Projektin vaiheistus on hallinnon kannalta oleellista työmäärien arvioimiseksi sekä resurssien allokoimiseksi. Toteutettu rahoituspalvelu on Innovaatioseteli, jossa myönnetään kiinteä summa rahaa tuotteiden tai palveluiden ostamiseksi toiselta osapuolelta eli palveluntarjoajalta [21]. Kun aikaisemmin hakemus siirtyi Kirjaamon kautta Rahoitus-ERP:iin, nyt Esitysvalmistelu-sovellus on vastuussa hakemuksien vastaanotosta ja sen mukaan syntyvän rahoitusasian käsittelystä. Tämä selkeyttää vastuunjakoa sovellusten välillä ja poistaa myös hallinnollisen ja täysin automatisoitavan vaiheen, jossa virkailijan oli käytävä Kirjaamon käyttöliittymästä siirtämässä hakemus eteenpäin. Rahoitus-ERP ei tavoitetilassa ole keskitetty toiminnanohjausjärjestelmä, vaan sovelluksien kokonaisuus muodostaa toiminnanohjausjärjestelmän.



Kuva 4.1. *Esitysvalmistelun uudistus -projektin tuotoksen järjestelmäarkkitehtuuri ja tietovirrat*

4.1 Integraatiot

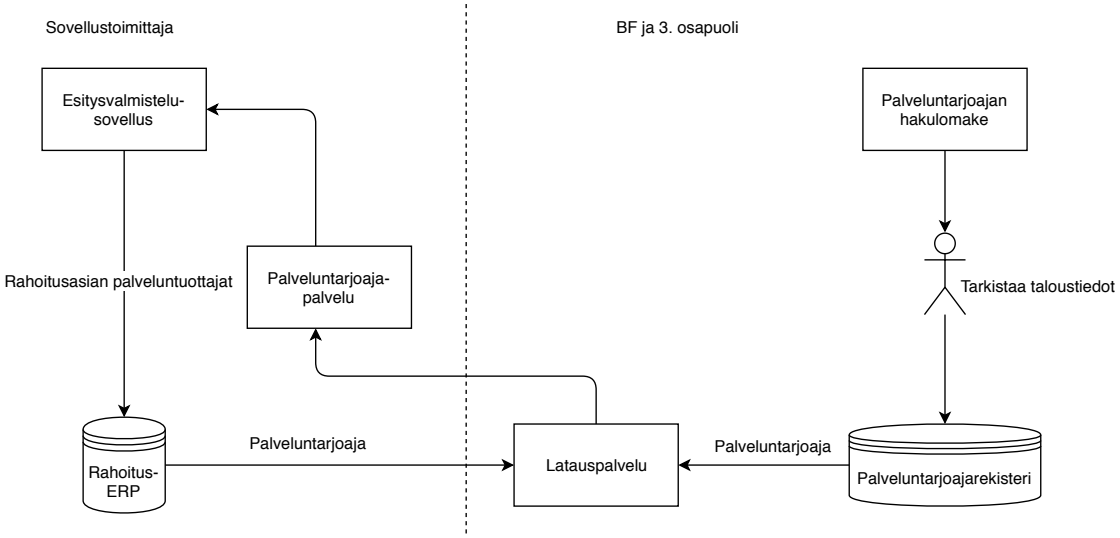
Tämä kohta pohjustaa vastausta tutkimuskysymykseen K1 selostamalla, miten Esitysvalmistelu-sovellus koostaa tietoa itselleen useasta lähteestä esitysvalmistelun mahdollistamiseksi. Oleellisen tiedon yhdistäminen, tiivistäminen ja esittäminen samalla näytöllä käyttöliittymässä helpottaa käyttäjän kannalta tilanteen kokonaiskuvan hahmottamista. Tiedon jäsentely taas auttaa yksityiskohtaisissa näytöissä keskittymään tiettyyn osakokonaisuuteen. Sen lisäksi, että tietoa koostetaan ja esitetään esitysvalmistelua varten, tietoa koostetaan ja siirretään siis myös eteenpäin muille sovelluksille ja integraatiokohteille. On tärkeää, että tiedot ovat sopivalla tavalla tallennettu myös rahoitusprosessin tuleviin vaiheisiin siirtoa varten. Tässä kappaleessa on kuvattu Esitysvalmistelu-sovelluksen näkökulmasta lähimpänä olevat integraatiokohteet ja niiden tärkeimmät toiminnot.

4.1.1 Palveluntarjoajapalvelu

Palveluntuottajiin liittyvä integraatio on monivaiheinen. Palveluntuottaja on organisaatio, jolta hakijaorganisaatio ostaa palveluita hyväksytyn rahoitushakemuksen perusteella saaduilla varoilla. Rahoitusprosessin mukaan palveluntuottajilta on tarkistettava verovelkatiedot ja luottoluokitus [1]. Palveluntarjoaja on BF:n esihyväksymä organisaatio, jonka luottoluokitus- ja verovelkatiedot ovat sekä hyväksyttävissä rajoissa että tarpeeksi tuorei-

ta, jolloin niitä ei tarvitse tarkistaa. Tällaisille organisaatioille rakennettiin Esitysvalmistelusovelluksen käyttöön integraation.

Kuvassa 4.2 on esitetty palveluntarjoajan eteneminen ja päätyminen Palveluntarjoajapalveluun. Katkoviivan vasemmalla puolella on sovellustoimittajan kehitystiimin hallinnassa olevat järjestelmäosat ja katkoviivan oikealla puolella on BF:n sekä kolmannen osapuolen toimittamat ja ylläpitämät osat.



Kuva 4.2. *Palveluntarjoajien tietovirrat BF:n kontekstissa*

Palveluntarjoaja tunnistetaan y-tunnuksen perusteella. Hakemukseen on kirjattu hyödynnettävien palveluntuottajien y-tunnukset (maksimissaan kaksi), jotka Esitysvalmistelusovellus jäsentelee ohjelmallisesti. Esitysvalmistelu-sovellus kysyy Palveluntarjoajapalvelusta, onko kyseisillä y-tunnuksilla hyväksyttyjä palveluntarjoajia. Jos palveluntarjoajaa ei löydy hakemuksessa ilmoitetun palveluntuottajan y-tunnuksen mukaan, on kyseessä uusi palveluntarjoaja. Uusien palveluntarjoajien taloustietojen tarkastuksesta on selostettu tarkemmin kappaleessa 4.2.4. Kun asiasta on tehty rahoituspäätös, siirtyy rahoitusasian palveluntuottajat latauspalveluun, mistä ne päätyvät lopulta palveluntarjoajaksi. Toinen tapa päätyä palveluntarjoajaksi on täyttämällä erillinen web-lomake palveluntarjoajaksi haluavan yrityksen toimesta. BF:n työntekijä lukee lomakkeen vastaukset, tarkistaa yritysten taloustiedot ja lisää yrityksen palveluntarjoajarekisteriin, joka on palveluntarjoajien alkuperäissijainti. Latauspalvelu yhdistää sekä Rahoitus-ERP:n tietokannasta että palveluntarjoajarekisteristä palveluntarjoajat Palveluntarjoajapalveluun, josta ne ovat uudelleenhyödynnettävissä tulevia hakemuksia varten.

Esimerkkihakemuksessa Joulupukin uudet vaatteet haetut palveluntuottajat ovat Vaatetaitelija Möttönen (5604776-4) ja Ompelija Pasi Zeus (3884844-5). Möttösen tiedot löytyvät Palvelutarjoajapalvelusta, mutta Zeuksen eivät. Zeus löytyy hakemuksen käsittelyssä sitten, kun nykyiseen rahoitusasiaan on tehty rahoituspäätös.

Palveluntarjoajien luottoluokitukset ja verovelkatiedot olisi teknisesti mahdollista tarkistaa myös ohjelmallisesti, jolloin esitysvalmistelijan ei tarvitsisi tarkistaa tietoja manuaali-

sesti uusien palveluntarjoajien kohdalla. BF tarkistaa samoja tietoja ohjelmallisesti myös hakevista organisaatioista (ks. kappale 4.1.2). Palveluntuottajien verotietoja ei kuitenkaan voida hallinnollisista syistä hakea ohjelmallisesti. Verovelkatieto katsotaan virkailijan toimesta YTJ:n (Yritys- ja yhteistietojärjestelmä) [38] asiointipalvelusta ja Rating Alfa -luottoluokitus asiakastieto.fi -verkkopalvelusta [33].

Y-tunnuksen jäsentäminen leipätekstistä on epäluotettavaa. Käyttäjän tekemät kirjoitusvirheet johtavat siihen, että palvelutuottajaa ei voida tunnistaa. Palveluntuottajien tulkitseminen hakemuksesta olisi vakaampaa, jos palveluntuottaja olisi taltioitu Hakemus-sovelluksessa rakenteelliseen muotoon, eli y-tunnus olisi erillinen tietokenttä. Hakemuksen käyttöliittymässä voisi olla y-tunnukselle erillinen kenttä, joka validoi käyttäjän syötteen ja tätä y-tunnusta voisi käyttää suoraan palveluntarjoajan tunnistamiseen. Y-tunnuksen validoinnissa voi käyttää y-tunnuksen tarkistusnumeroa [26]. Tämäkään ei kuitenkaan takaa, että y-tunnus olisi syötetty oikein. Vaikka y-tunnus olisi muodollisesti pätevä, voi se olla väärän organisaation y-tunnus. Tässä voisi hyödyntää myös palveluntuottajan nimeä sen vahvistamiseksi Palveluntuottajapalvelusta, mutta nimessäkin voi olla käyttäjän tekemiä virheitä jopa vielä todennäköisemmin, joten se ei tuo sinänsä lisäarvoa tarkistukseen.

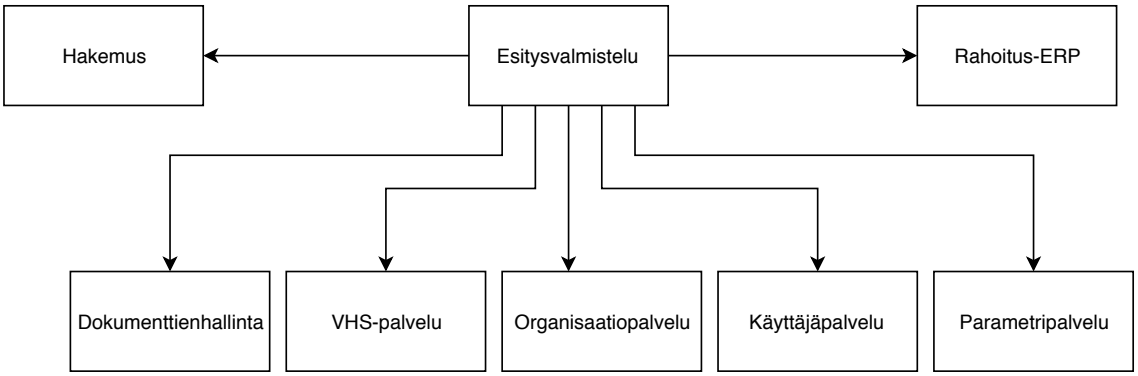
Suoraviivaisempi ratkaisu olisi, että Hakemus-sovellus integroituisi suoraan Palveluntuottajapalveluun, jolloin palveluntuottajien vahvistaminen tapahtuisi jo aikaisemmassa vaiheessa. Hakemuksen täyttäjälle voisi kertoa suoraan, onko kyseinen palveluntarjoaja on BF:n hyväksymä. Hakemus-sovellukseen tällaista ei kuitenkaan päätetty toteuttaa kehitysresurssien rajallisuuden vuoksi. Virheellisten Y-tunnuksien tapauksissa tieto oikaistaan hakemuksen täydennystoiminnallisuudella. Lisäksi y-tunnukseen liittyvät kirjoitusvirheet ovat harvinaisia.

Palveluntarjoajapalveluun voisi jatkokehityksenä tehdä myös toiminnallisuuden, joka ottaa vastaan hyväksyttyjä palveluntarjoajia. Näin käsittelysovelluksesta palveluntuottajan tarkistuksien jälkeen palveluntuottaja päätyisi palveluntarjoajaksi, jolloin sitä voisi hyödyntää tulevissa hakemuksissa saman tien. Muuten rahoitusasian kautta palveluntarjoajaksi päätyy vasta rahoituspäätöksen jälkeen. Epäsynkronoitu kehitys eri sovellusten ja sovellustoimittajien osalta on kuitenkin johtanut monimutkaiseen tilanteeseen ja tiedon vienti palveluntarjoajarekisteriin ei ole mahdollista.

4.1.2 Muut integraatiot

Tässä aliluvussa on kuvattu muut integraatiot, joiden kanssa Esitysvalmistelu-sovellus on tekemisissä. Ne on havainnollistettu kuvassa 4.3. Esitysvalmistelu-sovellus on tietoliikenteen kannalta aina kyselevä osapuoli.

Hakemus-sovellus tarjoaa sisäverkon sovelluksille rajapinnan, jonka avulla integroituvat sovellukset voivat listata ja ladata hakemuksia. Hakemus-sovellus muodostaa oman koostetun kokonaisuuden hakemuksen vapaamuotoisista teksteistä ja rakenteellisista syöte-



Kuva 4.3. *Esitysvalmistelu-sovelluksen käyttämät integraatiot Innovaatiosetelihakemuksissa*

kentistä sekä hakijaorganisaation tiedoista. Hakemuksen ollessa ladattavissa Hakemus-sovelluksen rajapinnasta, voidaan olettaa siihen liittyvän hakijaorganisaation perustiedot löytyvän Organisaatiopalvelusta. Hakemus-sovellus tarjoaa myös rajapinnan täydennyspyyntöjen julkaisemiselle sekä hakemuksen lukitukselle. Yleensä täydennystä pyydetään, kun jokin hakemuksen tiedoista on vajaa tai virheellinen. Hakemuksen voi lukita, jos esitysvalmistelija ei enää halua hakemuksen tietojen muuttuvan. Hakija voi täydentää hakemusta oma-aloitteisesti täydennyspyynnön puuttuessakin.

Organisaatiopalvelu tarjoaa mm. organisaation perustiedot (nimi, osoitteet, yritysmuoto, ym.) sekä joitain taloustietoja (luottoluokitus, koko, tilinpäätöstietoja ym.). Organisaatiopalvelussa tapahtuu samantapaista tiedon koostamista kuin Esitysvalmistelu-sovelluksessa. Tiedot pidetään ajan tasalla tapahtumapohjaisella mekanismilla, joka tallentaa muutoksia alkuperäislähteistä ajastetusti Organisaatiopalveluun, josta ne ovat nopeasti tarjolla rahoitussovelluksien käyttöön. Organisaatiopalvelu koostaa myös siirtymävaiheessa tietoa käytöstä poistettavista sisäisistä järjestelmistä.

Esitysvalmistelu-sovellukseen tulee hakemuksen ohella hakijan lähettämät liitteet. Sen vastuulla on myös arkistoida liitteet dokumenttienhallintaan. Esitysvalmistelun aikana liitteitä saapuu sekä hakemuksen ensilähetyksen yhteydessä että täydennyksien yhteydessä. Ennen päätöksentekoon siirtoa on varmistettava, että jokainen liite on arkistoitu. Siirron jälkeen asia ei ole enää Esitysvalmistelu-sovelluksen omistuksessa ja rahoitusprosessin tulevien vaiheiden sovellukset olettavat kaikkien liitteiden olevan saatavilla dokumenttienhallinnasta. Liitteitä yritetään arkistoida aina sitä mukaan, kun niitä saapuu, mutta joskus niiden vienti saattaa epäonnistua. Esitysvalmistelu-sovellus pitää kirjaa liitteiden arkistoinnin onnistumisesta ja koettaa aika ajoin arkistoida arkistoimattomia liitteitä.

VHS-palvelu tarjoaa Verohallinnon Velvoitteidenhoitoselvityksen [35] tietoja. Palvelu toimii fasadina verohallinnon palvelulle. Se tarjoaa rahoitussovelluksille kolme päätoimintoa: selvityksen tilaaminen, selvityksen tilan kysely sekä raporttien lataus. Palvelut ovat käytettävissä myös muille rahoitussovelluksille. Uuden asian tullessa vireille Esitysvalmistelu-sovellus tilaa hakijaorganisaatiolle uuden selvityksen, jos sellaista ei ole tai se on liian vanha. Esitysvalmistelu-sovelluksen käyttöliittymän toimintoihin kuuluu myös mahdolli-

suus lukea VHS-raportteja. Raportit voi ladata VHS-palvelusta, joka taltioi niitä itselleen.

Käyttäjäpalvelu koostaa BF:n rahoitusprosessien työntekijöiden tietoja rahoitussovelluksien saataville. Palvelun tarjoaa hakurajapinnan käyttäjille, jotta niitä voi listata sovelluksissa työnjakoon liittyvissä toiminnoissa. Käyttäjäpalvelu koostaa käyttäjien tietoja BF:n identiteetinhallinnasta.

Parametripalvelu tarjoaa referenssidataa rahoitussovelluksien käyttöön. Referenssidata sisältää rahoitussovelluksien domainin mukaista universaalia dataa. Referenssidatassa on myös rahoituspalveluiden konfiguraatio, joka määrittää esimerkiksi sen, mitä kysymyksiä kysytään tai mitä kustannuslajeja rahoituspalveluun kuuluu. Referenssidata pysyy muuttumattomana operatiivisessa toiminnassa, ja muuttuu vain silloin, kun sovelluksista asennetaan uusia versioita. Esimerkiksi “Innovaatiosetelillä ostettava palvelu“ on yksi referenssidatassa olevista *kysymyksien* otsikoista.

Rahoitusesitys siirretään päätöksentekovaiheeseen Rahoitus-ERP:iin. ERP tallentaa hakemuksen ja sen perusteella tehdyn esityksen omaan tietokantaansa ja sovittaa ne omaan tietomalliinsa. ERP:n tietokannan tietomallissa on otettava huomioon sen monimutkaisempi tietomalli. Siksi ERP:iin suunniteltiin rajapinta, jossa ERP:n sisäistä tietomallia ei tarvitse integroituvan sovelluksen tuntea. ERP:n vastaanottorajapinta suunniteltiin domainin mukaista mallia noudattaen. ERP oli ennen yksinomainen virkailijoiden sovellus rahoitusprosessin suorittamiseksi, eli tiedot on tallennettava ERP:n tietokantaan samassa tilassa, aivan kuin esitysvalmistelu olisi tehty vanhojen toimintamallien mukaisesti. Näin prosessin seuraavat vaiheet toimivat entiseen malliin ja palasteltu järjestelmien uudistaminen voi toteutua.

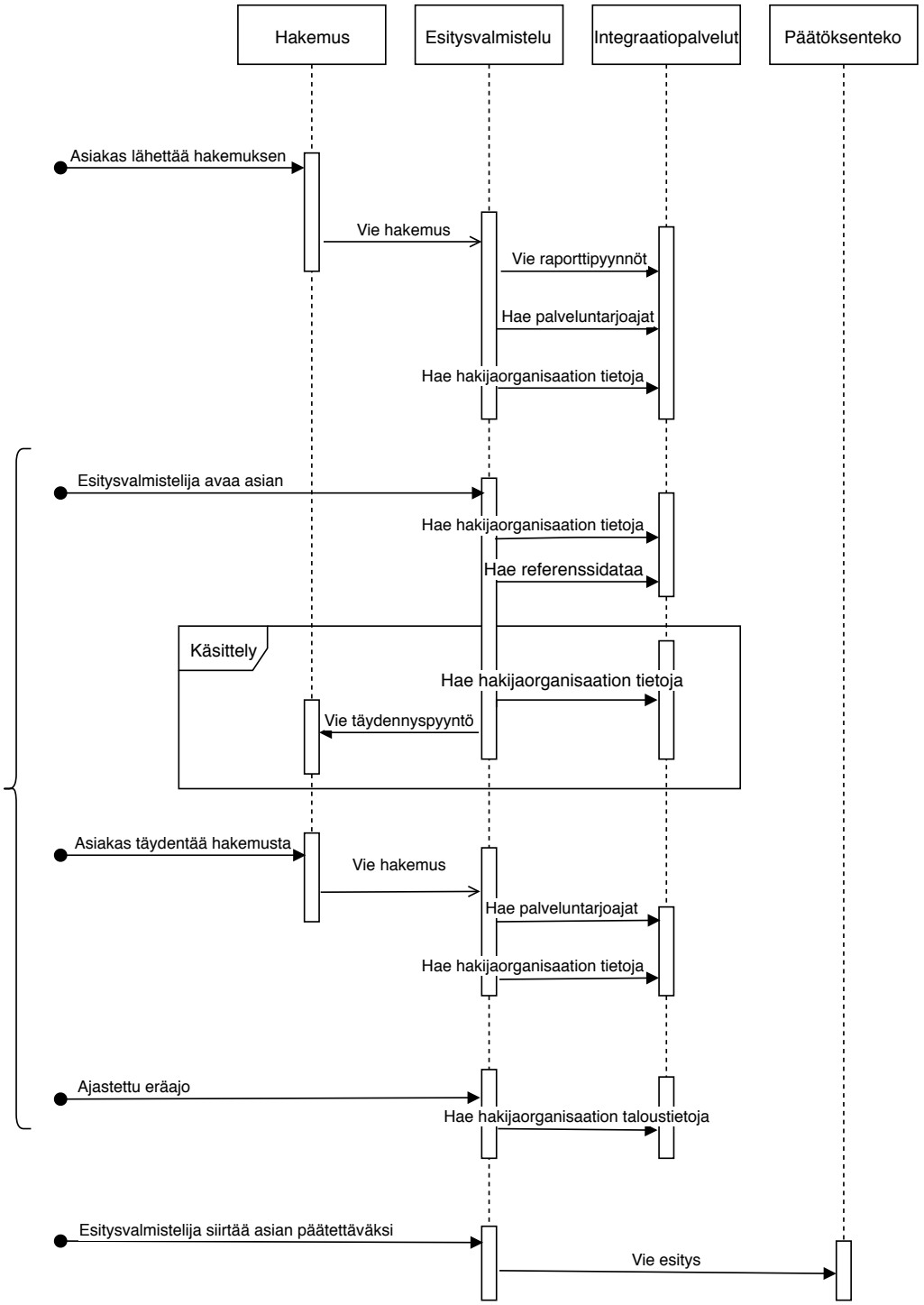
4.2 Tiedon koostaminen, prosessointi ja jäsentely

Tiedon koostamisessa on kaksi pääongelmaa: miten tietoja yhdistetään eri lähteistä yhdeksi kokonaisuudeksi ja kuinka ajan funktiona muuttuva tieto pidetään ajan tasalla. Tässä kappaleessa otetaan kantaan tutkimuskymykseen K1 kuvaamalla tiedon koostamista ja esittämistä eri lähteistä rahoitusjärjestlemän domain-mallin näkökulmasta sekä tutkimuskysymykseen K2 pohtimalla erilaisia ratkaisuja tiedon koostamisen ajastuksen ongelmiin. Sen ohella pohditaan ratkaisujen hyviä ja huonoja puolia sekä vaihtoehtoisia ratkaisuja.

4.2.1 Toteutettu koostaminen

Esitysvalmistelu-sovelluksen toteutuksessa on käytetty useampaa kappaleessa 2.3 mainittua menetelmää kullekin sopivassa tilanteessa. Esitysvalmistelu-sovelluksen näkökulmasta asiaan liittyviä tietoja haetaan tai viedään integraatioihin sekä hakemuksen latauksen yhteydessä, asynkronisena tausta-ajona, että käyttöliittymän toimintojen yhtey-

dessä. Välimuistirakenteet sijaitsevat Esitysvalmistelu-sovelluksen back-endissä. Kuvassa 4.4 on sekvenssikaavio, jossa on esitetty tiedonhaut ja -viennit eri järjestelmien välillä. Kuva havainnollistaa tiedonhakujen järjestyksen ja käynnistävän tekijän.



Kuva 4.4. *Kutsut Esitysvalmistelu-sovelluksen ja sen integraatiokohteiden välillä sekvenssikaaviona. Mustasta pallosta alkavat nuolet kuvaavat käynnistävää tekijää.*

Kuvassa 4.4 vasemman reunan pallot kuvaavat *integraatiotapahtumien* käynnistävät tekijät. Kolmiopäinen nuoli tarkoittaa synkronista kutsua. Tikkupäinen nuoli asynkronista kutsua, jossa tiedon vastaanottaja kyselee tietoja ajastetusti. Aaltosulkeella ryhmiteltyjä ta-

pahtumia voi esiintyä asian esitysvalmistelun aikana mielivaltaisen monta kertaa ja niiden järjestys voi vaihdella vapaasti. Käsittely-kehyksessä olevat kutsut tapahtuvat esitysvalmistelijan toiminnan aikana käyttöliittymän kautta ja niitä voi tapahtua myös mielivaltaisen monta kertaa. Asian avaaminen on aina käsittelyn edellytys. Asian avaaminen tapahtuu käytännössä vähintään kerran. Hakemuksen täydennys hakijalta onnistuu, vaikka esitysvalmistelija ei olisi täydennyspyyntöä julkaissutkaan, ja koko esitysvalmistelu voidaan suorittaa ilman ainuttakaan täydennystä.

Esitysvalmistelu-sovellus kyselee hakemuksia ajastetusti tietyin väliajoin ja alustaa hakemuksen pohjalta asian omaan tietokantaansa. Hakemuksien latauksessa ei käytetä minkäänlaista välimuistia, koska alustuksessa tiedon on aina vastattava täsmälleen Hakemuksen tilaa eikä käyttäjä koskaan suoraan hae hakemuksia. Alustuksessa organisaation taloustietoja haetaan tietty osajoukko, jotka tallennetaan tietokantaan. Osajoukkoon kuuluu sellaiset tiedot, jotka on haettavissa ainoastaan organisaatiokohtaisesti mutta joita tarvitaan kaikkien asioiden listauksessa. Taloustietoja haetaan Organisaatiopalvelusta ja VHS-palvelusta. Jokaisen organisaation osalta olisi tehtävä erilliset integraatiokutsut, joten listauksen suorituskyvyn kannalta tiedot on hyvä tallentaa sovelluksen omaan tietokantaan. Yksi listaava tietokantakysely on huomattavasti nopeampi. Asian alustukseen kuuluu myös prosessin mukaiset raporttipyynnöt sekä palveluntarjoajien haku. Hakemuksen alustuksessa asetetaan myös *esitykseen* kuuluvia tietoja. Esitys sisältää ne tiedot, joiden alkuperä on Esitysvalmistelu-sovelluksessa. Näiden alkuarvoiksi valitaan hakemuksessa vastaavia tietoja tai niiden mukaan pääteltyjä arvoja parametrihallinnassa olevien sääntöjen mukaan.

Kaaviossa 4.4 näkyvät hakijaorganisaation taloustietojen haut ovat Organisaatiopalvelun osalta todellisuudessa kokonaisen laajan tietojoukon haku, koska rajapinta ei tarjoa niiden hakua hienojakoisemmin. Tämä ei kuitenkaan ole periaatteessa haitallista suorituskyvyn kannalta, sillä Organisaatiopalvelu on toteutettu niin, että koko organisaation tiedot tulevat tietokannasta valmiiksi koostettuna. Esitysvalmistelu-sovelluksen näkökulmasta pyyntöjä on joka tapauksessa tehtävä vähintään yksi eikä vastauksen pituus ole merkittävästi suurempi, että siitä koituisi suorituskykyhaittaa.

Hakemuksen täydennys toimii samankaltaisesti kuin hakemuksen lähetys. Kun hakemus päivittyy, Esitysvalmistelu-sovellus lataa sen Hakemus-sovelluksesta ja päivittää sen mukaan tietoja tietokantaansa sekä hakee integraatioista hakemuksen sisällöstä riippuvia tietoja. Esimerkiksi palveluntuottajien päivittyessä ne on tarkastettava uudelleen. Toisin kuin hakemuksen alustuksessa, iso osa esityksen tiedoista jätetään kuitenkin ennalleen, sillä esitysvalmistelijan asettamia arvoja ei haluta ylikirjoittaa.

Asian avaamisessa organisaatiosta haetaan laajemmat tiedot kuin hakemuksen latauksessa. Tässä tilanteessa halutaan näyttää organisaation perustietoja ja taloustietoja laajemmin kuin asioiden listauksessa. Taloustietoja haetaan Organisaatiopalvelusta ja VHS-palvelusta. Molempien haku päivittää myös Esitysvalmistelu-sovelluksen tietokannan varaisen taloustietojen välimuistin yksittäisen organisaation osalta. Asian avaamisessa ladataan myös referenssidata. Sen haussa voi käyttää pitkäikäistä muistipohjaista välimuis-

tia, sillä se muuttuu harvoin ja käytännössä sen muuttuessa referenssidataa käyttävien sovelluksien välimuistit tyhjennetään. Asian käsittelyn jälkeen asia siirretään Rahoitus-ERP:iin päätöksentekoon.

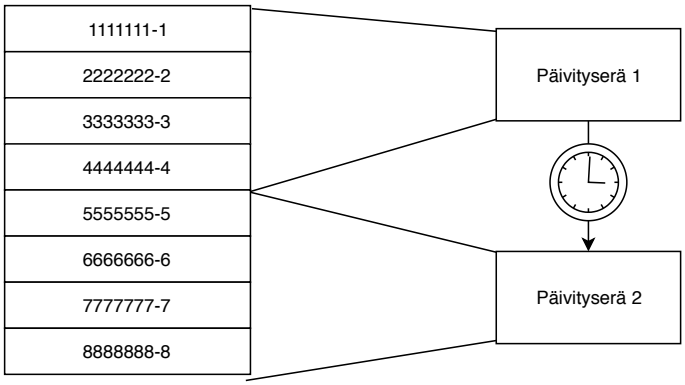
Asian avaamisen jälkeen esitysvalmistelija käsittelee asiaa. Käsittelyn aikana joidenkin toimintojen ohella haetaan mahdollisesti organisaation taloustietoja tai viedään täydennyspyyntö Hakemus-sovellukseen. Näihin kuuluu esimerkiksi talousraporttien haku tai pakotettu luottoluokituksen päivitys.

Esitysvalmistelu-sovelluksessa on myös ajastettu ajo, joka päivittää hakijaorganisaatioiden taloustietoja taustaeräajona. Sillä mahdollistetaan se, että Esitysvalmistelu-sovelluksen tietokannassa olevat asialistaukseen vaikuttavat tiedot päivittyvät välimuistiin tarpeeksi pienellä viiveellä. Lisäksi eräajo varmistaa, että työlistan tilaan (kuva 4.9) vaikuttavat taloustiedot päivittyvät jokaiselle listan asialle ilman, että jokainen asia on erikseen avattava. Taustakyselyllä taataan se, että ulkoisessa järjestelmässä tiedon muututtua tiedon tila Esitysvalmistelu-sovelluksessa vastaa muuttunutta tietoa. Tieto päivittyy ilman, että käyttäjältä vaaditaan aktiivisuutta, jolloin käyttäjälle jää aikaa tehdä muuta työtä. Tiedonhakujen aikavälin voi asetettaa esimerkiksi 5-20 minuuttiin, jolloin tieto päivittyy kohtuullisessa ajassa mutta tietojärjestelmiä ei rasiteta liikaa.

Esimerkkitapauksessa hakemus diaarilla 90210/31/2019 löytyy uutena hakemuksen listausrajapinnasta ja Esitysvalmistelu-sovellus tulkitsee, että tätä diaari ei vielä ole tämän tietokannassa. Esitysvalmistelu-sovellus alustaa asian itselleen, hakee organisaation y-tunnuksella 9123456-5 Organisaatiopalvelusta ja VHS-palvelusta taloustiedot, sekä palveluntuottajien y-tunnuksilla 5604776-4 ja 3884844-5 palveluntuottajat. Samalla raporttipyynnöt hakijaorganisaation y-tunnuksen mukaan päätyvät niihin kuuluviin palveluihin.

4.2.2 Tiedonhakujen tehostaminen integraatioista

Tiedonhakueriä voi lomittaa siten, että tietoja haetaan pienemmällä aikaväleillä ja pienemmillä organisaatiojoukoilla. Kuva 4.5 havainnollistaa, miten eräajot voidaan jaksottaa. Kokonainen päivityskierros sisältää useamman päivityserän, joka koostuu pienestä joukosta lähtöarvoja. Tässä tapauksessa lähtöarvot ovat y-tunnuksia. Päivityserien välissä on pieni viive, joka voi olla huomattavasti pienempi kuin kokonaisen päivityskierroksen välissä. Päivityseriin jakamisella saavutetaan aika-akselilla tasaisemmin jakautunut kuorma. Kokonaisen päivityskierroksen jälkeen jokainen lähtöarvo on käyty läpi. Päivityserän aikana on tehtävä jokaiselle organisaatiolle yksittäiset kutsut organisaatiotietojen integraatioihin, joten erän koolla ei integraatoiden kannalta ole merkitystä. Sovelluksen omaan tietokantaan tallennuksen kannalta erien koko on hyvä kuitenkin sopeuttaa sellaiseksi, että tietokannan päivitysoperaatioita ei tehdä liian usein. Peräkkäisten integraatiokutsujen tulokset voidaan yhdistää ja tallentaa erätallennuksena. EV-projektissa ei tiedonhakujen lomittamista toteutettu, sillä Innovaatioseteli-hakeumuksien osuus rahoitusasioista on pieni.

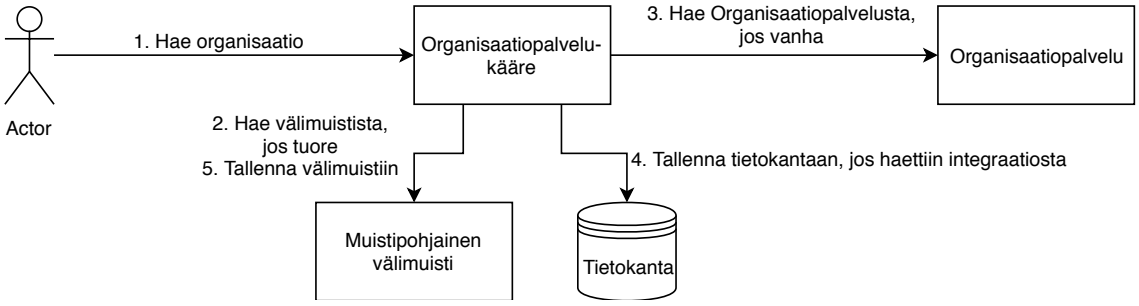


Kuva 4.5. *Tiedonhaun lomittaminen hakuerittäin*

Back-endin kannalta organisaatioiden hakeminen yksittäisillä kutsuilla eräajona ei ole sen ajankäytön kannalta suorituskykyisempää kuin organisaatioiden hakeminen työlistan latauksen yhteydessä. On otettava kuitenkin huomioon vasteajat käyttäjälle. Työlistan lataaminen käyttäjälle saattaisi kestää useampia sekunteja, ellei minuutteja. Tämän takia organisaatioiden hakua on vältettävä käyttäjän operaatioiden aikana. Suorituskyky front-endiä palvellessa on huomattavasti parempi, kun käytetään tausta-ajoa päivittämään välimuistia. Organisaatioiden yksittäiseltä lataukselta ei kuitenkaan voida välttyä, sillä se on ainoa tapa hakea niitä.

Taloustietojen välimuistin ei tarvitsisi olla välttämättä tietokannassa. Muistipohjainen välimuisti ajaisi saman asian yksittäisen asian avaamisen että asioiden listauksen suhteen. Muistipohjaisesta välimuistista hakeminen perustuu organisaation yksilöivään avaimeen, mutta välimuistitoteutukset voivat mahdollistaa myös koko välimuistin sisällön hakemisen. Tietokannassa säilöminen tekee kuitenkin välimuistidatasta persistenttiä. Sovelluksen käynnistyessä edellinen tila on yhä tallessa, eikä työlistan lataaminen aiheuta heti kaikkien organisaatioiden hakua.

Kuvassa 4.6 on esitetty Organisaatiopalvelun integraation välimuistirakenne Esitysvalmistelu-sovelluksessa. Kuvassa oleva *kääre* (engl. wrapper) on sovelluksen osa, joka käärii jonkin palvelun ja näyttäytyy samanlaisena kuin varsinainen palvelu. Kääre piilottaa käyttäjältä palvelun kutsumiseen liittyviä yksityiskohtia ja voi sisältää lisälogiikkaa.



Kuva 4.6. *Organisaatiopalvelun integraation välimuistirakenne*

Organisaatiovälimuisti rakentuu sekä muistin että tietokannan varaan. Muistissa säilötään niitä tietoja, jotka tarvitaan vain yhdestä organisaatiosta tietyllä hetkellä. Tietokantaan tal-

lennetaan tiedot, joita tarvitaan myös asialistauksessa. Kääreessä tehdään tarkastus, onko muistissa oleva organisaatio tarpeeksi tuore. Jos on, niin se palautetaan vastauksena suoraan. Muussa tapauksessa haetaan organisaatio Organisaatiopalvelusta ja päivitetään sekä muistissa että tietokannassa oleva välimuisti.

Jos asiaa availlaan tiheästi, organisaatiovälimuistin suorituskyky tulee erityisesti hyödyksi. Asian avauksia tapahtuu tiheästi esimerkiksi silloin, kun asiaa käsittelee useampi käyttäjä samanaikaisesti. Tietojen päivittäminen jokaisen rinnakkaisen käyttäjän asiakasohjelmaan vaatii asian latauksen eli avaamisen, kun yhden käyttäjän muutokset on näytettävä toisen käyttäjän käyttöliittymässä. Tietokannan päivittäminen on sivuvaikutus, josta hyödytään asialistauksen latauksessa.

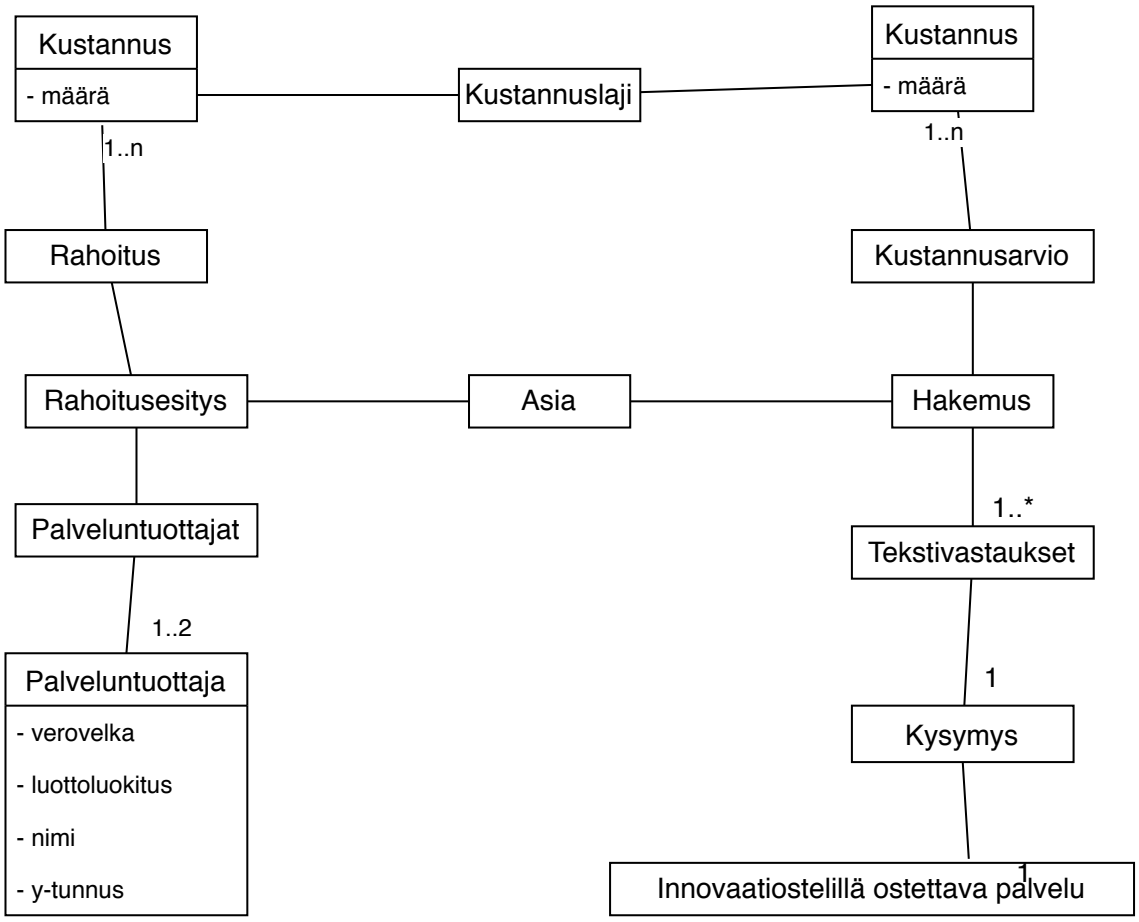
Taloustietojen välittämiseen voisi hyödyntää myös julkaise/tilaa -mallista (publish/subscribe) ratkaisua. Taloustietojen muuttuminen on arvaamatonta, jolloin siihen reagoiminen muutoksen tapahtuessa olisi huomattavasti tehokkaampaa ajastetun eräajon sijaan. Ajastettu eräajo ei tässä tapauksessa edes poista tarvetta hakea jokaisen organisaation tiedot erikseen. Toteutusprojektin aikana kuitenkin resurssien vähyys ja aikataulu eivät mahdollistaneet viestijonon ja -kuuntelijan toteuttamista. Viestijonosovelluksen jalkauttaminen ja sovellusten integrointi siihen olisi vaatinut työtä. Lisäksi valmiit tuoteratkaisut eivät tarjonneet suoraan mahdollisuutta viestijonojen persistointia tietokantaan ja siten niiden saatavuuden varmistamista BF:n kontekstissa. Muistinvaraiset viestijonot kadottaisivat tietoa palvelimien ongelmatilanteissa, joten joka tapauksessa suorat integraatiokutsut ja tietokantaan tallennus ovat välttämättömät.

4.2.3 Tiedon mallinnus

Käsitemalli kuvassa 4.7 esittää hakemuksen ja esityksen sisältöjen suhteita. Käsitemallin voidaan ajatella kuvaavan tiedon loogisia ja luonnollisia suhteita, eikä tämä vastaa mitään tarkkaa teknistä mallia. Asiaan voidaan viitata globaalisti diaarin avulla ja siihen liittyy myöhemmissä rahoitusprosessin vaiheissa paljon muitakin käsitteitä. Asia on olemassa eri järjestelmässä rahoitusprosessin eri vaiheissa.

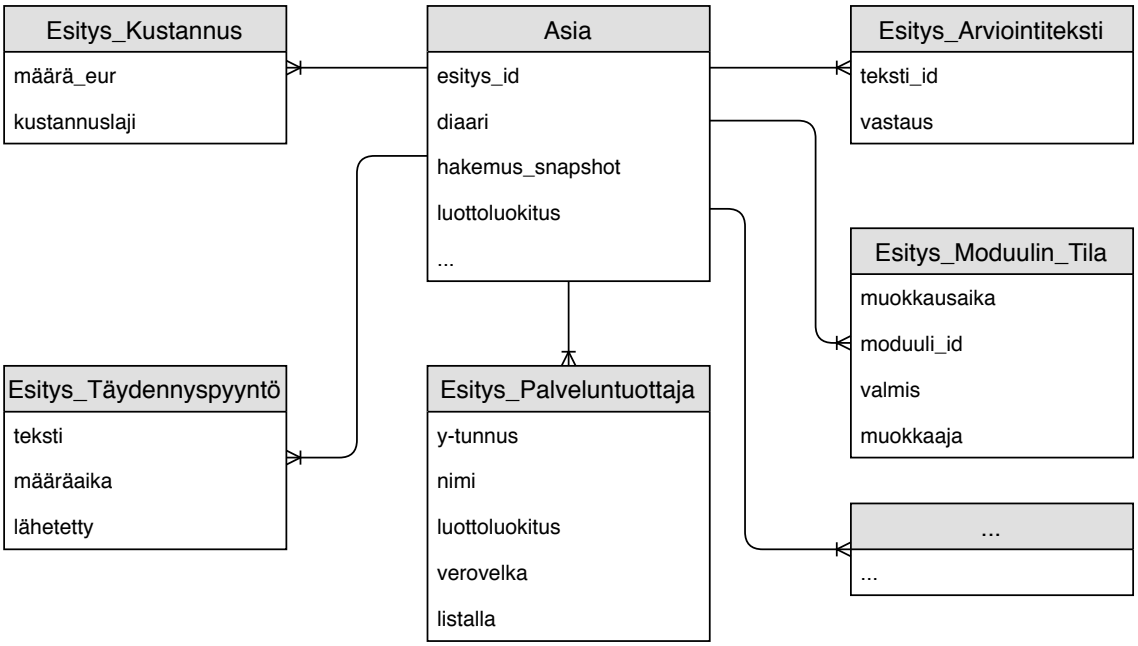
Hakemuksen ja rahoitusesityksen välillä käsitteissä on selkeä vastaavuussuhde, vaikka tieto voi olla hieman eri muodoissa. Palveluntuottajat ilmoitetaan hakemuksessa Innovaatiosetelillä tuotettava palvelu -nimisessä tekstivastauksessa, kun taas esityksessä palveluntuottajat on prosessoitu ja tallennettu rakenteeseen, joka vastaa niiden luonteesta johdettua todellista rakennetta. Hankkeen kustannukset taas ovat rakenteeltaan samanlaiset molempien sovelluksien käsitteinä, joskin niillä on eri tarkoitus ja haetut kustannukset ovat oleellisia myös esitysvalmistelussa. Esitetty kustannus on rahoituksen perusteeksi hyväksytty virkailijan arvioima kustannuksen suuruus.

Kuva 4.8 esittää relaatiotietomallia, jonka mukaan asia taltioidaan Esitysvalmistelu-sovelluksen tietokantaan. Kuva ei vastaa tarkalleen tietokannan skeemamääritystä, mutta antaa kuitenkin kuvan tietokokonaisuuksien jäsentelystä ja suhteista. Asia-taulussa on



Kuva 4.7. *Pelkistetty asian käsitemalli esitysvalmisteluvaiheessa erään rahoituspalvelun osalta*

diaarin lisäksi myös sijaisavain esitys-id, jolla sovellus voi sisäisesti viitata asioihin. Esitys_Kustannus -taulu on esimerkki tietosisällöstä, joka alustetaan hakemuksen mukaan oletusarvioihin mutta jonka arvoja voi muuttaa esitysvalmistelussa. Täydennyksen yhteydessä hakemuksesta tulevat uudet kustannukset eivät ylikirjoita esitysvalmistelussa asetettuja arvoja. Esitys_Palveluntuottaja-taulu on esimerkki taulusta, jonka sisältö muuttuu sekä hakemuksen latauksessa että täydennyksessä. Taulussa on aina täsmälleen ne palveluntuottajat, jotka hakija on ilmoittanut ja jotka esitysvalmistelija on lisännyt. Esitys_Täydennyspyyntö -taulu taas liittyy prosessiin mutta ei rahoitusesityksen sisältöön. Taulussa säilötään täydennyspyynnön yksityiskohdat, jotka viedään Hakemus-sovellukseen vasta täydennyspyynnön julkaisussa. Esitys_Moduulin_Tila puolestaan on esimerkki tiedosta, joka ei liity prosessiin vaan käyttöliittymän toimintojen ohjaamiseen. Esitys_Arviointiteksti on samanlainen käsite kuin hakemuksen kysymysteksti, mutta siihen vastaa esitysvalmistelija hakija sijaan. Taloustietojen välimuistin tiedot sijaitsevat Asia-taulussa.



Kuva 4.8. *Asian relaatiomalli Esitysvalmistelu-sovelluksen tietokannassa*

4.2.4 Koostetun tiedon esittäminen ja muokkaaminen

Tiedon koostamisen tarve käyttöliittymällisessä sovelluksessa on myös se, että tieto on esitettävä käyttäjälle. Esityksen on palveltava käyttäjää prosessin suorittamiseksi. Tietokannan relaatiomallissa oleva data ja integraatioista saatu data on tuotava käyttöliittymään, missä on tehtävä yhtälailla muuntelua kuin integraatioissa.

Esitysvalmistelu-sovelluksen käyttöliittymässä näytetään siis eri järjestelmien ja käyttäjien tuottamaa tietoa. Tietoja tallennetaan Esitysvalmistelu-sovelluksen tietokantaan ajanhetken näkökulmasta kolmella eri periaatteella.

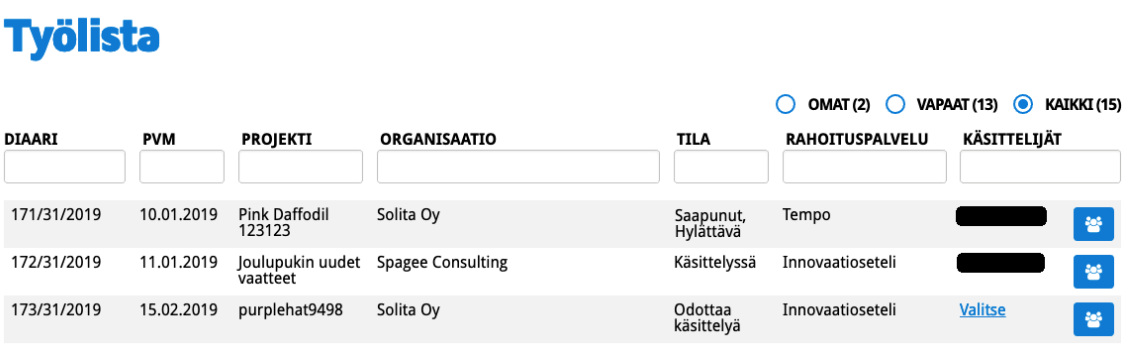
- Hakemuksen latauksen yhteydessä
- Taustalla järjestelmän toimesta ajastetusti
- Käyttäjän käyttöliittymän muokkausoperaatioiden aikana

Tätä on havainnollistettu myös kuvassa 4.4, missä tiedon päivityksen heräte voi olla joko käyttäjän syöte tai järjestelmän ajastus.

Palveluntuottajatietoja päivitetään tietokantaan sekä hakemuksen latauksen yhteydessä että käyttöliittymän kautta tapahtuvilla muokkausoperaatioilla. Hakemuksen latauksen aikana palveluntarjoajalistasta haetaan hakemuksen tekstistä jäsenneltyjen y-tunnuksien perusteella rivit ja tallennetaan Esitysvalmistelu-sovelluksen tietokantaskeeman mukaiseen Palveluntuottaja-tietokantatauluun. Jos palveluntuottajia ei löydy listalta, niitä voi lisätä käyttöliittymästä, jolloin ne tallentuu samaan tietokantatauluun. Ainoastaan käyttöliittymän kautta lisättävien palveluntuottajien taloustietosarakkeisiin (verovelka jne.) tallennetaan tietoa; listalla olevien palveluntuottajien osalta nämä tiedot on prosessin mukaisesti tarkistettu, jolloin niiden arvoilla ei ole merkitystä Esitysvalmistelu-sovelluksen nä-

kökulmasta ja tällaisten palveluntuottajien tunnistamista varten on tietokannassa sarake "listalla". Palvelutarjoajia ei tarvitse hakea asian avauksen yhteydessä, sillä niiden haku on oleellinen ainoastaan silloin, kun haetut palveluntuottajat muuttuvat. Palveluntuottajataulu toimii eräänlaisena välimuistina.

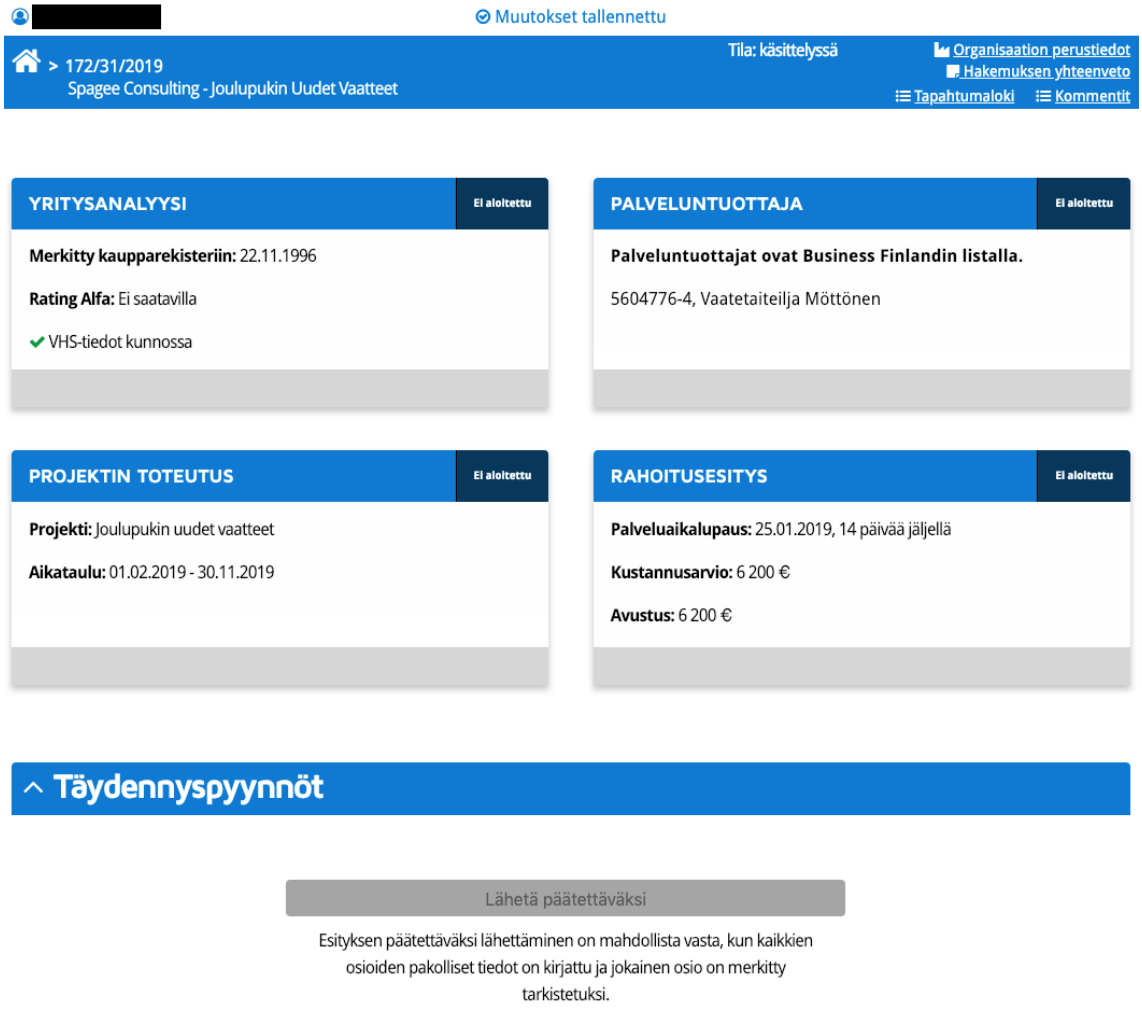
Työlistalla (kuva 4.9) näytetään lista asioista, jotka ovat esitysvalmisteluvaiheessa. Asioilla on tila, joka ilmaisee, onko asiaa käsitelty, onko sen käsittely kesken vai onko se lähetetty päätöksentekoon. Asian tila ilmoittaa myös, ylittävätkö taloustiedot rahoituspalvelun minimivaatimukset. Tilassa on myös mahdollista ilmoittaa, kun taloustietojen valmistumista odotetaan, mutta tällaista ei tämän projektin aikana toteutettu. Listalta voi muuttaa asian työryhmää, eli asettaa ja poistaa siltä käsittelijöitä. Listalla näytetään kunkin asian senhetkinen työryhmä, ja käyttäjä voi myös suodattaa listaa siten, että näkee ainoastaan ne asiat, joiden työryhmään hän kuuluu. Listalta voi myös navigoida yksittäisen asian etusivulle.



Kuva 4.9. *Esitysvalmistelu-sovelluksen työlista*

Asian esitysvalmistelun käyttöliittymä on jaoteltu *moduuleihin*. Moduulit on jaoteltu niiden sisältämän tiedon tai tarkoituksen perusteella. Esimerkiksi Palveluntuottajat-moduulissa tarkastetaan hakijan ilmoittamat palveluntuottajat ja Yritysanalyysissä analysoidaan hakijaorganisaation taloudellista tilaa. Asian etusivulla näkyy kaikki moduulit, jotka liittyvät asian rahoituspalveluun. Etusivun moduuleissa näkyy tiivistelmä moduulin sisällöstä, eli oleellisimmat tiedot moduulin sisällöstä. Käyttäjän ei siis tarvitse navigoida moduulin sisälle nähdäkseen hyvin karkealla tasolla, kuinka pitkällä moduulin työstö on. Moduulin otsikosta navigoidaan moduulin sisälle. Kuva 4.10 havainnollistaa asian etusivu -näkymää.

Kuva 4.11 demonstroi, kuinka ensimmäinen hakijan ilmoittaman palvelutuottajan y-tunnus 5604776-4 on löytynyt hakemuksen vastaanottohetkellä esihyväksyttyjen palveluntarjoajien listalta mutta toinen y-tunnus 3884844-5 ei, jolloin esitysvalmistelijan on tarkistettava verovelkatieto ja luottoluokitus manuaalisesti kenttien vieressä olevista web-linkitetyistä palveluista. Erona aikaisempaan on, että Esitysvalmistelu-sovelluksessa ennestään hyväksyttyjen palveluntarjoajien haku on automatisoitu. Vanhassa esitysvalmistelusovelluksessa palveluntuottajat oli tarkastettava aina palveluntuottajarekisteristä manuaalisesti. Nyt hyväksyttyjen palveluntuottajien osalta esitysvalmistelijan ei tarvitse tehdä minkäänlaisia tarkasteluja, vaan etusivulla sekä Palveluntuottajat-moduulissa näytetään, että pal-

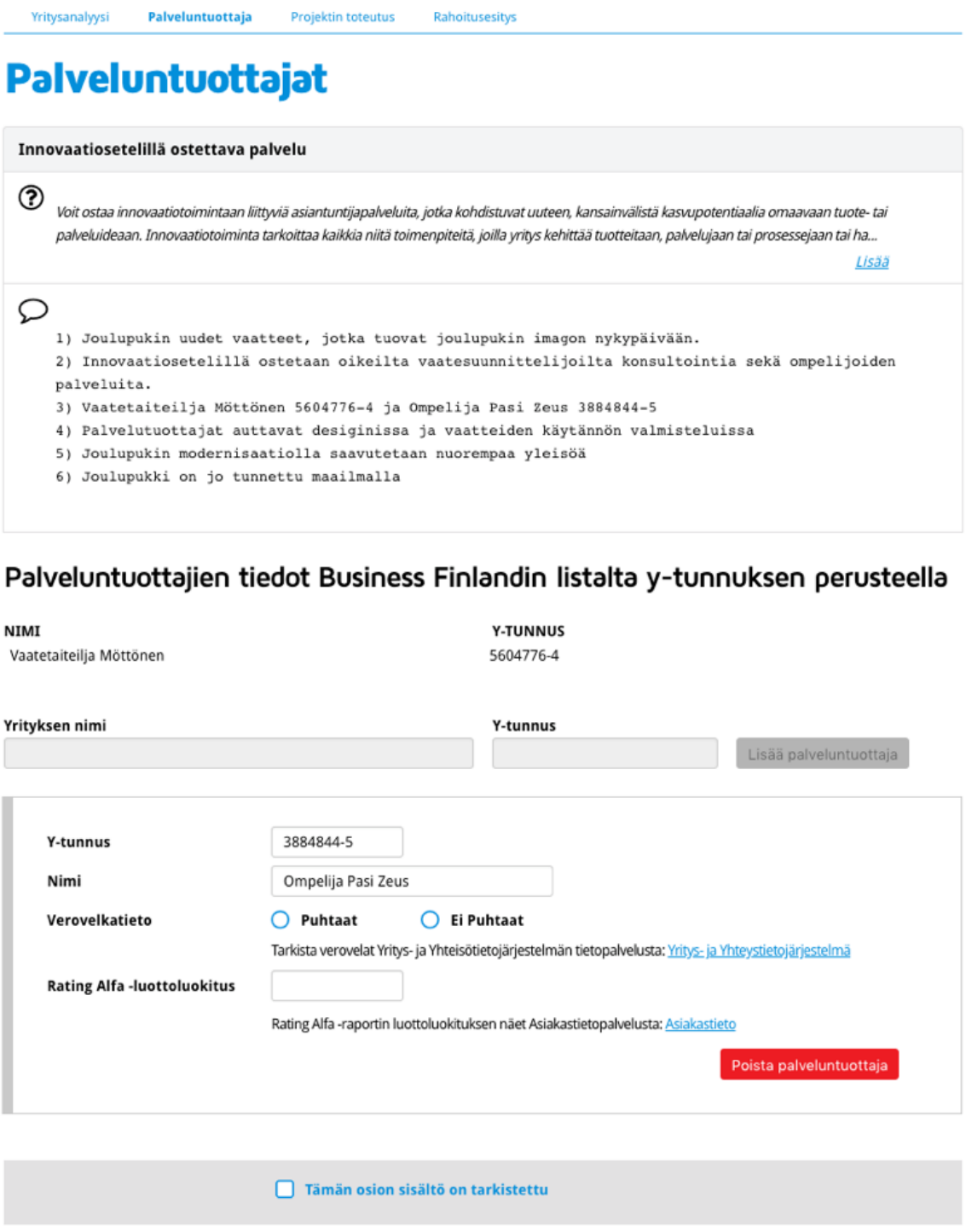


Kuva 4.10. *Asian etusivu Esitysvalmistelu-sovelluksen käyttöliittymässä*

veluntuottajat ovat hyväksyttyjä.

Rahoitusesitys-moduulissa on taulukko rahoitusasian kustannuksista (kuva 4.12). Taulukkossa on kahdesta eri lähteestä koostettuna tiedot, jotka ovat käsitteellisesti rinnakkaiset ja samankaltaiset. Taulukon riviotsikoina olevat kustannuslajit ovat universaalit, ja niiden avulla eri lähteiden tiedot yhdistetään. Kuva demonstroi kustannustaulukon ratkaisun yleisyyttä ja soveltuvuutta laajempiin palveluihin. Innovaatioseteli-rahoituspalvelussa on vain yhden tyyppinen kustannus.

Jokaisen moduulisivun alareunassa on myös alue, jossa sijaitsee Tämän osion sisältö on tarkistettu -valinta. Kun esitysvalmistelija on tarkistanut moduulin sisällön ja aktivoi valinnan, moduulin tilaksi tulee valmis, joka näkyy myös asian etusivulla. Esityksestä voi valmistella pieniä palasia kerrallaan ja etusivulla voi nähdä nopeasti yleiskuvan esitysvalmistelun tilasta. Tarkistettu-valinnan alapuolella listataan myös moduulin puutteita, jos sellaisia on (esimerkiksi palveluntuottajan puuttuminen kokonaan). Puutetarkastelut ovat yksinkertaisia ohjelmallisia tarkasteluja, jolla voidaan nopeasti tulkita, onko esitykseen syötetty kaikki tarvittava tieto tai onko kaikki päätöksentekoprosessiin kuuluvat tiedot saatavilla ulkoisista lähteistä. Tarkastelu ei lähtökohtaisesti tulkitse mitenkään tiedon



Kuva 4.11. *Palveluntuottajat-moduuli Esitysvalmistelu-sovelluksen käyttöliittymässä*

sisältöä, sillä sellaisen toteuttaminen on hyvin työlästä ja käyttäjien voidaan olettaa syöttävän työtehtäviinsä liittyvät asiat asianmukaisesti. Tarkistus koskee ainoastaan sitä, onko yksittäisen kentän tieto olemassa vai ei. Puutetarkastelu ohjaa käyttäjää antamaan kaikki tiedot, jotka prosessissa vaaditaan. Tällä varmistetaan, että rahoitusesitys siirretään eheänä prosessin seuraavaan järjestelmään.

KUSTANNUSLAJI	HAETTU €	ESITETTY €
Rahapalkat	123	123
Henkilösivukustannukset	123	123
Prosenttia palkoista	100 %	100 %
Ostettavat palvelut	123	123
Muut kustannukset	123	1 232
Prosenttia palkoista ja ostettavista palveluista	50 %	500.8 %
Yhteensä	**492**	**1 601**

Kuva 4.12. *Kustannustaulukko Esitysvalmistelu-sovelluksen käyttöliittymässä*

4.2.5 Käyttöliittymän käytön sujuvuus

Moduulijaottelu mahdollistaa useamman rinnakkaisen käyttäjän yhteiskäytön samanaikaisesti. Esitysvalmistelu-sovelluksessa rahoitusesitykseen liittyviä tietoja voi muokata käyttöliittymän kautta kenttäkohtaisesti. Esimerkiksi palveluntuottajan nimi, y-tunnus, verovelka ja luottoluokitus ovat kaikki neljä erillistä, tallennuksen kohteena olevaa kenttää, joista yhden muuttaminen saa aikaan sen tallennuksen tietokantaan.

Rinnakkaisessa käytössä on peräkkäisten tallennusten johdosta ongelma, jossa ensimmäisenä tallennettu tieto hukkuu. Ongelma syntyy silloin, kun useampi käyttäjä ei tiedä muokkaavansa samaa kenttää rinnakkain. Aikaisemman tallennuksen tekijän käyttöliittymään ei tule tietoja seuraavasta tallennuksesta. Ongelma on mahdollista ratkaista siten, että jokaisen muokkaavan operaation jälkeen kunkin aktiivisen käyttäjän käyttöliittymään tuodaan päivitetty tieto. Tämä koskee siis kaikkia käyttäjiä, joiden näytöllä kyseinen tieto esitetään. Tämä vaatisi, että selaimessa suoritettavan sovellusosan olisi kuunneltava muutoksia palvelimella tapahtuvista tiedon muokkauksista. Viiveen tulisi olla käytännössä olematon, jotta tiedon häviämiseltä vältytään. Lisäksi rinnakkaisen muokkauksien yhteensovittaminen on erittäin haastavaa, sillä esimerkiksi tekstiä voi poistua tai lisääntyä mihin tahansa kohtaan. Määrittelyjen aikana todettiin, että rinnakkaista käyttöä tulisi ilmenemään vähän ja erityisesti tuskin koskaan samassa moduulissa, joten ongelma ratkaistiin siten, että kukin moduuli lukitaan sitä muokkaavalle käyttäjälle muokkausviiveen ajaksi.

Jokaisen tallennusoperaation mukana menee tallennuspyynnön mukana metatietona asian sisäinen tunniste sekä moduulin tunniste, jotka tallennetaan aikaleiman ja muokkaajan ohella tietokantaan Esitys_Moduulin_Tila -tauluun. Käyttäjän ei tarvitse erikseen lukita moduuleja. Asiakassovellus kyselee 30 sekunnin välein, onko kyseisen asian kyseistä moduulia muutettu muokkausviiveen sisällä ja hakee koko asian datan, jos viimeisin päivitysaika on edellisen hakuajan jälkeen. Käyttöliittymässä estetään tiedon tallentaminen kyseisessä moduulissa, jos tieto on lukittuna toiselle käyttäjälle. Jos joku kuitenkin tekee muokkauksen, ennen kuin tieto sen muuttumisesta on ehtinyt saapua asiakassovelluksel-

le, näytetään virheilmoitus, jolloin käyttäjä ei voi virheellisesti olettaa tiedon tallentuneen. Tällöin kukin käyttäjä näkee uusimman tiedon suhteellisen pienellä viiveellä, muttei pysty muokkaamaan dataa ylikirjoittaen samalla toisen käyttäjän muutoksia. Rinnakkaiset muokkaukset eri käyttäjien toimesta eri moduuleissa kuitenkin sallitaan. Selkeä parannus vanhaan toimintamalliin on, että Rahoitus-ERP:ssa koko projekti lukittui muokkauksen yhteydessä, joten rinnakkaismuokkaus oli mahdotonta.

Kenttäkohtainen tallennus mahdollistaa sujuvan käytön. Käyttäjän ei tarvitse täyttää kokonaisia sivuja ja sen jälkeen lähettää koko sivua lomakkeena. Käyttöliittymässä ei tarvitse myöskään olla tallennusnappeja, kun tieto tallennetaan automaattisesti heti muokkauksen jälkeen. Joidenkin tietojen osalta on toisaalta tallennettava useampi kenttä kerrallaan tiedon eheyden vuoksi. Esimerkiksi palvelutuottajan nimi ja y-tunnus ovat molemmat pakollisia sarakkeita, sillä ne toimivat relaatiomallissa pääavaimena. Sama y-tunnus voi ilmetä useammassa palveluntuottajassa, sillä saman organisaation eri osastot voivat olla BF:n listalla.

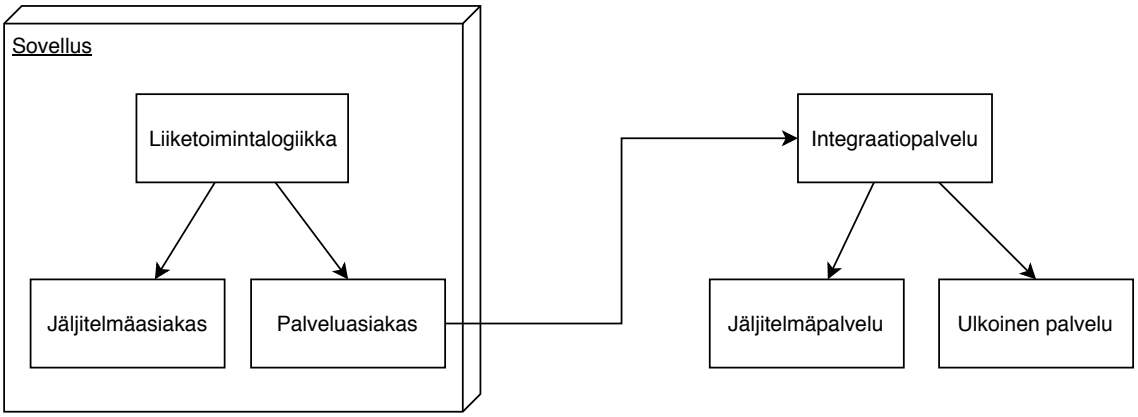
4.3 Sovelluskehityksen riippuvuus integraatioista

Sovelluskehityksen aikana varsinaiset palvelut eivät ole saatavilla kehitysympäristössä mutta kuitenkin sovelluksen toiminta niitä vasten on vahvistettava ja testattava. Integraatiopalvelu muuntaa tiedon kolmannen osapuolen esitysmuodosta domain-muotoon, mutta sitäkin on integraation käyttäjän osattava tulkita. Tiedon koostamiseen ja sen jälkeen sen esittämiseen liittyvän koodin toiminta on varmistettava. Kukin tieto on näytettävä oikeassa paikassa oikeanlaisena. Tiedon puuttuessa tai ollessa vaillinainen, tiedon esityksen on vastattava tiedon tilaa. Esimerkiksi käyttöliittymässä voidaan piilottaa tietty taulukko tai painike, mikäli jotain tietokenttää ei ole saatavilla. Integraatioihin liittyy myös autentikointi, jonka toimivuus on hyvä varmistaa mahdollisimman aikaisessa vaiheessa.

Integraatioiden toimivuus on varmistettava, ennenkö sovellukset jalkautetaan tuotantoon. Jos tuotanto on ainoa paikka, jossa tarvittava integraatio on saatavilla, on kehitettävä ratkaisuja toimivuuden varmistamiseksi testausympäristöissä. Tähän tarkoitukseen tehtiin jäljitelmäpalveluita (eng. mock service) [31]. Niiden rajapinta matkii varsinaisen palvelun rajapintaa. Kuvassa 4.13 on esitetty, miten jäljitelmäpalvelut soveltuvat arkkitehtuurillisesti tietojärjestelmäympäristöön.

Testitilanteessa kuvassa oleva varsinainen integraatiopalvelu on sama kuin tuotantokäytössä. Integraatiosovellus taas puolestaan kytkeytyy normaalisti 3. osapuolen palvelurajapintaan. Tämä yhteys voidaan vaihtaa jäljitelmäpalveluun. Integraatiopalvelusta viitataan vain url-osoitteella kolmannen osapuolen integraatiokohteisiin, jolloin integraatiopalveluun voi konfiguroida jäljitelmäpalvelun osoite. Tällä järjestelyllä voidaan testata sekä itse integraatiopalvelun toimintaa että siitä riippuvien sovelluksien toimintaa.

Samaa jäljitelmäperiaatetta voi käyttää myös sovelluksen tasolla, kuten kuvassa 4.13 on sovelluksen sisällä havainnollistettu. Sovelluksen koodi on modularisoitu, ja sekä varsi-



Kuva 4.13. *Palvelujäljitelmien sijoittuminen sovellusarkkitehtuurissa*

nainen palveluasiakas että jäljitelmäasiakas tarjoavat samanlaisen rajapinnan muun sovelluksen käyttöön. Liiketoimintalogiikka on sama sekä testitilanteessa että tuotantokäytössä. Näin liiketoimintalogiikan ja kaikkien siitä riippuvien sovellusosien testaaminen on mahdollista.

Jäljitelmädatan sisältö on staattista, parametroitua tai satunnaista. Staattinen data palvellaan aina sellaisenaan sovelluksesta. Parametroitu data on myös luonteeltaan staattista, mutta palveltava tietojoukko voidaan valita integraation kutsun parametrien perusteella. Satunnaisen datan tapauksessa tietoa puolestaan tuotetaan oikeaan rakenteeseen ja kenttien arvot satunnaistetaan yksitellen. Näistä voi käyttää myös yhdistelmiä.

Esimerkiksi VHS-palvelun käyttämä verohallinnon järjestelmän jäljitelmä ottaa vastaan raporttipyyntöjä, palauttaa niiden tiloja sekä palvelee staattisia dokumentteja tai satunnaisia datajoukkoa raportteina. Raporttipyyntöjen perusteella mielivaltaisen ajan kuluttua raportin tilaksi asetetaan "valmis". Näin emuloidaan tuotannossa olevaa verohallinnon palvelua.

Jäljitelmäpalveluista on erityisesti hyötyä niin itse sovelluskehittäjän tietokoneella sijaitsevassa kehitysympäristössä kuin CI-ympäristössä. Jäljitelmäpalveluja voidaan ajaa samalla palvelimella kuin muita sovelluksia, mikä helpottaa kehitysympäristön perustusta. Sovellustason jäljitelmää käytettäessä ei tarvitse sovellusriippuvuuksia asentaa lainkaan. Tämä on hyödyllistä etenkin, kun sovelluksien riippuvuusketjut ovat pitkiä. Sopivan riippuvuusvälin katkaiseminen jäljitelmällä on riittävä sekä sovellukehityksen ja testauksen kannalta että kehitysympäristön perustuksen puolesta. Kehitysympäristössä ja CI-ympäristössä myös autentikointiin liittyä identiteetinhallinta on jäljitelty.

4.4 Toteutusteknologia

Esitysvalmistelun back-end on toteutettu Spring ja Spring Boot -teknologialla [32]. Spring on Java-pohjainen ohjelmointikehys, joka tarjoaa monipuoliset ominaisuudet web-sovelluksen ja integraatioiden rakentamiseen. Sen ominaisuuksiin lukeutuu muun muassa

ajastetut ajot, tietokantahaut, välimuistit, rajapintojen suojaus, REST-palveluihin integroituminen ja REST-rajapinnan tarjoaminen. Spring tarjoaa myös toiminnallisuuden, jolla voi palvella käyttöliittymän staattisia resursseja kuten HTML-sivuja, CSS-tyylejä ja web-selaimessa ajettavissa olevia JavaScript-tiedostoja.

Spring-sovelluksen rakenne perustuu beaneihin eli komponentteihin, jotka riippuvat toisistaan puumaisesti. Spring-ohjelmointikehys tarjoaa myös kattavia työkaluja sovelluksien testaamiseen. Testejä voi ajaa injektoimalla beaneihin jäljitelmiä eristäen näin testeihin tiettyjä alueita sovelluksesta.

Front-endin pääasiallinen toteutusteknologia on React [27] ja sen lisäosiksi kehitetyt kirjastot. React on ohjelmointikehys HTML-pohjaisten käyttöliittymien toteuttamiseen. Reactin liitännäiskirjastot tarjoavat muun muassa tilanhallinnan apuvälineitä, käyttöliittymäkomponentteja ja testialustan. Front-endin koodia voi testata riippumattomasti back-endistä.

5 ANALYYSI

Design Science -menetelmäoppi on kehitetty tuotesuunnittelun designin tutkimiseen. Se tutkii ongelmanratkaisua käytännön ongelmiin ja perustuu kaksivaiheeseen prosessiin: toteuta ja arvioi. Näitä vaiheita iteroidaan, ja edellisen vaiheen arvioinnin tuloksia hyödynnetään seuraavassa iteraatiossa. Design Sciencen tavoitteena on tuottaa artefakteja, jotka kuvaavat käsitettä, mallia, menetelmää tai instantaatiota, joka tuottaa lisäarvoa tai ratkaisuja tietyn ongelman ratkaisemiseksi tietystä lähtökohdasta. Teoria jakautuu myös hienommin 7 eri ohjesääntöön, joilla ratkaisuja kehitetään ja arvioidaan. Yksi seitsemästä ohjesäännöstä on suunnittelun arviointi (evaluation). Tutkimustyön tuloksena voi olla itse tuotetun sovelluksen lisäksi myös muun muassa sovelluskehityksen menetelmät. [15]

Tuotettu artefakti ratkaisee käytännön ongelman. Sidosryhmät määrittelevät vaatimukset tai odotukset artefaktin vaatimuksista. Tutkimuksen tekijän on määriteltävä ratkaistava ongelma, siihen liittyvät sidosryhmät sekä arvioitava ehdotettu ongelman ratkaisuksi tuotettu artefakti. Tutkimuksen lopputuloksena on tuotetun artefaktin kuvaus ja sen liityntäkohdat ennestään tunnettuun teoriaan. [19]

Tässä työssä artefaktin ratkaisema ongelma on toteuttaa uusi järjestelmä, jonka on koostettava tietoa eri lähteistä esitysvalmisteluprosessin edellyttämiseksi tiettyjen reunaehtojen mukaisesti (luku 3). Sidosryhmiä ovat esitysvalmistelijat eli loppukäyttäjät, tuoteomistajat, jotka johtavat järjestelmäkehitystä, sekä sovelluskehittäjät, jotka tarjoavat ammattitaitoaan järjestelmäkehitykseen ja pyrkivät mahdollistamaan sen jatkuvuuden. Ehdotettu artefakti koostaa tietoja eri lähteistä ja muodostaa sekä tiivistelmänäkymiä että käsitteellisesti jaoteltuja yksityiskohtaisia näkymiä tietoon (luku 4). Tämän työn tuloksena on kuvauksia toteutetusta arkkitehtuurista ja sen perustelut sekä sen aikaansaamiseksi hyödynnettävistä menetelmistä. Ratkaisumalleja voisi hyödyntää vastaavanlaisissa projekteissa, joissa on eri tietolähteitä, joista muodostetaan yksi käyttöliittymä, jossa tuotetaan lähtötietojen perusteella uutta tietoa.

Tässä luvussa suoritetaan arviointia sekä toteutuksen tarkoituksenmukaisesta toiminnasta että sovelluskehityksen menetelmien toimivuudesta. Toteutetun järjestelmän arviointiin voi käyttää muun muassa näitä menetelmiä [15]:

- Arkkitehtuurianalyysi: Tutki artefaktin sopivuutta tietojärjestelmäarkkitehtuuriin
- Toiminnallinen testaaminen: Ajetaan testejä artefaktin rajapintoja vasten vikojen löytämiseksi ja tunnistamiseksi
- Kenttätutkimus: Seurataan artefaktin käyttöä

Kappale 4 sisältää toteutuksen kuvauksen ohella analyysiä sovelluksen ratkaisuista ja pyrkii siten argumentoimaan toteutetun järjestelmän käyttökelpoisuudesta. Yksityiskohtaiset ongelmanratkaisukuvaukset, sekä selostetut hyvät ja huonot puolet osoittavat ratkaisujen olevan harkittuja. Vaihtoehtojen tunnistaminen ja niiden sovellettavuuden tai sen puutteen perustelu vahvistaa valittujen ratkaisujen järkevyyttä. Ratkaisujen toimivuutta havainnollistetaan myös esimerkein. Lisäksi se, että ratkaisuista voi johtaa kehityskelpoisia jatkokehitysideoita, vahvistaa ratkaisujen pitävyyttä. Tämä toteuttaa arvioinnissa arkkitehtuurianalyysin menetelmän.

Projektin aikana hyödynnettiin loppukäyttäjien palautetta, jota kerättiin sekä käyttäjien suorana palautteena koko projektin ajan että projektin loppuaikana hyväksymistestauksen [36] muodossa. Nämä vastaavat tutkimuskysymykseen K3. Hyväksymistestauskierrokset toteuttavat arvioinnissa kenttätutkimuksen muodon siten, että kehittäjät voivat seurata loppukäyttäjien toimintatapoja ja arvioida, onko sovellus tarkoituksenmukainen ja käytettävä. Loppukäyttäjän hyväksymistestauksessa annetaan käyttäjille suoritettavaksi tehtäviä toteutetulla järjestelmällä ja kerätään palautetta. Tämä suoritettiin sellaisessa vaiheessa, että käyttöominaisuuksia oli valmiina sen verran, että itse rahoitusprosessi saattoi olla suoritettavissa ja käyttöliittymää oli jo alettu viimeistelemään. Ensimmäisellä testauskierroksella (HyTe1) tuoteomistajat käyttivät sovellusta testiympäristössä ja kirjasivat muistiinpanoja. Tuoteomistaja voi toimia loppukäyttäjän roolissa testitapauksissa, koska tämä tuntee rahoitusprosessin. Näin ei myöskään viedä aikaa todellisilta loppukäyttäjiltä. Toinen hyväksymistestauskierros (HyTe2) toteutettiin tilaisuudessa, johon kokoontui yhteen otos loppukäyttäjistä, tuoteomistajan edustaja sekä kehitystiimin edustajia. Käyttäjän suullisesta palautteesta tehtiin muistiinpanoja. Hyväksymistestauskierrokset toteuttaa toiminnallisen testaamisen siltä osin, että loppukäyttäjä toimii vahvistajana toimintojen oikeellisuudesta. Toteutuksen hyvyyden arviointiin sovelletaan tässä työssä myös sovelluksen julkaisun jälkeinen palaute.

Rahoitussovelluksiin kohdistetaan myös testausautomaatiota aina kehittäjien tekemien muutoksien yhteydessä. Testausautomaatio toteuttaa arvioinnin toiminnallisen testaamisen osalta ja ne vahvistavat sovelluksen toiminnan sekä uusien ominaisuuksien osalta että olemassa olevien ominaisuuksien osalta. Testausautomaatio on menetelmä varmistaa sovelluksen toiminta ja edesauttaa jatkokehityksen sujuvuutta (K3).

5.1 Arviointi

Kehitystyön henkilöstöön kuului sekä toimittajan sovelluskehittäjiä ja arkkitehteja, että tilaajan tietohallinnon edustajia, joita voidaan nimittää *tuoteomistajiksi*. Tuoteomistaja on henkilö, joka tuottaa priorisoidun listan tehtävistä ja ominaisuuksista, joita toimittajalta ostetaan. Esitysvalmistelusovelluksen toteutuksen aikana hyödynnettiin tuoteomistajien osaamista sekä järjestelmä- että prosessituntemuksen kannalta. Tuoteomistajat ovat olleet pitkään mukana BF:n rahoitusprosessin tietojärjestelmien kehityksessä ja ylläpidossa, joten he tuntevat prosessit ja järjestelmät hyvin. Näin vaatimuksien ja määrittelyjen

laatiminen pääsi alkuun tosi helposti.

Ratkaisuja alettiin etsiä käyttöliittymärautalankakuvilla [10] sekä prosessikuvilla (K3). Tuoteomistajat olivat arvokas resurssi kuvien piirtämisessä. Rautalankakuvasta on johdettavissa tiedon tarve taustajärjestelmistä. Kuvat ovat ilmiselvä pohja käyttöliittymän suunnittelulle. Toisaalta ne antavat lähtökohdat tiedonhakujen toteutukselle. Tiedonhaun toteutuksen aikana alkaa väistämättä selviämään, mitä tietoa on mahdollista tämänhetkisistä integraatiopalveluista hakea. Tästä voidaan johtaa suunnittelutehtäviä joko integraatiokohteiden kehittämiseksi tai käyttöliittymän toimintojen suunnittelemiseksi toisella tavalla.

Määrittelyssä iteratiivisuus nousi tärkeään rooliin. Monesti toistui tapaus, jossa on toteutettava jonkinlainen luonnos lopullisesta tahtotilasta, ennen kuin päästiin yhteisymmärrykseen sovelluksen lopullisesta toimintamallista. Tämä on tyypillinen piirre ohjelmistojen suunnitteluprosessissa [6]. Sovelluksen toimintoja on mahdotonta suunnitella valmiiksi etukäteen, koska mieli ei voi hahmottaa kaikkia mahdollista lähtötietoja ja ongelmia, ja johtaa niihin ratkaisuja hetkessä. Ratkaisujen löytämiseen auttaa se, että toteutetaan luonnosmainen versio, jota voi arvioida. Mieli osaa ottaa kantaa konkreettiseen luonnokseen. Erityisesti on helppoa nähdä, mikä ratkaisussa toimii väärin tai tehottomasti.

5.1.1 Integraatiot ja välimuistit

Ajanmittaan muuttuvan tiedon pitäminen ajan tasalla Esitysvalmistelu-sovelluksessa (K2) on toteutettu tiedon lähteestä ja tarpeesta riippuen erilaisilla synkronisilla ja asynkronisilla hauilla. Esitysvalmistelun integraatiossa Hakemus-sovellukseen ei käytetä lainkaan välimuistia. Hakemuksen tila on aina nähtävä täsmällisenä. Parametrihallinnan ja käyttäjäpalvelun suhteen käytetään suhteellisen pitkäikäistä muistipohjaista välimuistia, sillä tieto muuttuu harvoin ja sen haku käyttöliittymään halutaan nopeaksi. Organisaation taloustietoihin liittyvissä integraatioissa on lyhytikäinen välimuisti, josta lähekkäin toistuvat tiedonhaut palvellaan nopeasti, mutta arvaamattomasti muuttuvan tiedon muuttuminen havaitaan nopeasti. Taloustietojen välimuisti mahdollistaa myös suorituskykyisen latauksen käyttöliittymän listausnäkymiä varten.

Tiedon suora haku sitä tarvittaessa ratkaisee suurimman osan integraatiotarpeista. Tämä kuitenkin rasittaa sovelluksia ja tietoliikenneverkkoa, eli sovellusten suorituskyky kärsii. Lisäksi tietoa on aktiivisesti kyseltävä jatkuvasti ilman, että on ennustettavissa, onko tieto päivittynyt edellisen kyselyn jälkeen. Käyttäjien aikaa ei kannata tuhlata siihen, että he käynnistelisivät ajoittain tietojen hakua sellaisin odotuksin, että esimerkiksi jokin raportti olisi valmistunut. Automaattiset tietojen haut taustalla minuuttien viiveellä tuovat muuttuneet tiedot käyttäjän tietoon käytännöllisesti katsoen riittävän nopeasti ja vaivatta.

Tiedon päivittymisen aikariippuvuus hahmottui vasta toteutuksen alkamisen jälkeen. Vaikka saatavilla olevat tietolähteet olivat tiedossa jollain tasolla projektin alussa, joskin rakenteeltaan ei välttämättä alun alkaen selkeitä, tietojen päivittyminen ja se, milloin mitäkin tietoa on haettava, ei ollut ilmeistä. Sekvenssikaaviossa 4.4 olevat tiedon päivittämisen

käynnistävät impulssit syntyivät ohjelmoijien pohdinnasta koodin kirjoittamisen lomassa sekä käyttäjäpalautteen perusteella.

Integraation aktiivisen osapuolen tulisi olla se, missä tieto muuttuu. Tiedon muuttumisesta tuleva heräte aktivoi tiedon päivittymään sitä tarvitseville osapuolille. Näin ei liikuteta turhaa tietoa osapuolten välillä. Tämä vaatisi kuitenkin sen, että tiedon julkaisijan on tiedettävä, ketkä tiedosta on kiinnostuneita. Optimaalisin ratkaisu on palvelu, jonka tehtävän on ottaa vastaan julkaisu tiedon lähteestä ja jakaa se siitä kiinnostuneille. Viestinvälittäjän käyttöönotto vaatii työtä, jos sellaista ei ole organisaatiossa otettu käyttöön mutta sen hyödyt ovat pitkällä tähtäimellä todennäköisesti panostuksen arvoisia.

5.1.2 Käyttäjäpalautteen perusteella tehdyt muutokset

Hyväksymistestauksessa syntyy enimmäkseen käyttöliittymän toimintaan liittyvää palautetta. Tämä on odotettavissa, sillä käyttäjät ovat sen kanssa suoraan vuorovaikutuksessa eivätkä käyttäjät tiedä yksityiskohtia taustalla olevista tiedonhauista. Hyväksymistestauksessa nousi esiin esimerkiksi sovelluksen navigointiin liittyviä ongelmia sekä käyttöliittymän puutteellinen palaute joidenkin käyttäjän toimintojen vaikutuksesta. Tällaiset muutokset liittyvät pelkästään selaimen asiakasohjelman toteutukseen. Käyttäjien palautteesta on kuitenkin tunnistettavissa selkeästi tiedon koostamiseen liittyvää parannettavaa.

Järjestelmien uudistaminen mahdollistaa prosessimuutoksia sen tehostamiseksi. Kuitenkin vallitsevat prosessit on otettava huomioon palastellussa järjestelmäuudistuksessa. HyTe2:ssa kirjattu muistiinpano “Projektiehto ei välity päätöskäsittelyyn ja pitää olla "palvelutuottaja on...- Pitää siirtyä evaliin” johtuu siitä, että aikaisemmin palveluntuottajat tarkistettiin ja kirjoitettiin käsin "Projektiehto-arviointivastaukseen. Eristäminen palveluntuottajat erilliseksi tarkistukseksi on järkevää, mutta vanha järjestelmä ei osannut ottaa huomioon tiedon muuttunutta rakennetta. Tekstiin oli liitostettava vanhan prosessin aikana tyypillistä tekstivastausta muistuttava teksti palveluntuottajatietoineen.

HyTe2:ssa nousi esiin aihe tietojen automaattisesta päivittymisestä. Palaute “Mistä tietää että puutteelliset tiedot on saapuneet, esim. päivitetty vhs? Nouseekö työpöydälle? Pitääkö käydä tarkistamassa tilanne?” tiivistää ongelman hyvin. Tässä vaiheessa VHS:n tila ladattiin suoraan VHS-palvelusta asian avauksessa. Selvityksen tila ei siis näkynyt "työpöydällä"eli työlistalla (Kuva 4.9). Tuoreen tilan tuomiseksi työlistalle toteutettiin ajastettu tausta-ajo, joka päivitti organisaatioiden taloustietoja työlistaa varten. Tämä oli prosessin tehostamiseksi merkittävä parannus, jonka ratkaisuna oli automatisoitu tiedonhaku

Palautteessa nousi esiin täydennettyjen tietojen visualisointi siten, että esitysvalmistelija näkisi, mitkä tiedot hakemuksesta on muuttunut. Tämä on monimutkainen ongelma, koska hakemuksessa on paljon erilaisia tietoja eri muodossa. Tämän perusteella tehtiin kuitenkin käyttöliittymään ratkaisu, joka pakottaa esitysvalmistelijaa tarkastamaan koko esityksen täydennyksen saavuttua. Täydennyksien visualisointi jatkokehitysideoihin, koska se on vaativa toiminnallisuus, jonka toteuttamiseen kannattaa panostaa aikaa.

Hyväksymistestauksessa esiintyi kommentteja siitä, että joku tieto ei näytä tietyssä tilanteessa oikeanlaiselta. Tämän perusteella voidaan arvella, että integraatiot eivät tarjoa samanlaista dataa testiympäristössä ja tuotantoympäristössä. Syynä voi olla myös sovelluksessa piilevä bugi, jota ei ole löydetty ohjelmallisilla testeillä. Syystä huolimatta palauteen perusteella kehitystiimi tutkii ongelman testiympäristössä ja tekee korjaustoimenpiteitä. Korjaus voi olla tietyn testitapauksen toteuttaminen jäljitelmäpalveluun tai yksikkötesteihin. Loppukäyttäjien testaus auttaa minimoimaan sovelluksen bugeja ennen tuotantoonmenoa ja löytämään bugeja, joita testausautomaatiolla ei välttämättä löydetä.

5.1.3 Testausautomaatio

EV-projektin aikana käytettiin erilaisia testaustautomaation muotoja laadun varmistamiseksi sekä virheiden palautejakson minimoimiseksi. Taulukko 5.1 kuvaa aikana käytettyjä eri testaamisen muotoja [36] sekä niiden suoritustiheyttä. Testeissä suoritetaan osa sovelluskoodia ja vahvistetaan, että sen lopputulos on odotettu. Testejä voi käytännössä ajaa missä tilanteessa tahansa, mutta eri tyyppiset testit ovat sopivampia tiettyihin tilanteisiin kuin toiset. Yksinkertaisemmat ja eristetymmät testit ovat nopeita ajaa ja siten soveltuvat ajettavaksi tiheään, kun taas monimutkaiset ja useammasta järjestelmän osasta riippuvia testejä kannattaa ajaa harvemmin.

Taulukko 5.1. *Automaattisten testien muodot Esitysvalmistelu-sovelluksessa*

Testausmuoto	Kuvaus	Ajotiheys
Yksikköstestit	Testaa sovelluskoodin liiketoimintalogiikkaa aliohjelmakohtaisesti	Jatkuvasti ohjelmoinnin ohella, CI muutoksien yhteydessä
Integraatiotestit	Testaa sovelluksen eri osien välistä tai sovelluksen ja integraatiokohteiden välistä yhteistoimintaa	Jatkuvasti ohjelmoinnin ohella, CI muutoksien yhteydessä
Suorituskykytestit	Mittaa rajapinnan eri operaatioiden viiveet	CI kerran vuorokaudessa
Käyttöliittymätestit	Automaatti käyttää sovellusta web-käyttöliittymän kautta kuten oikea käyttäjä	CI muutoksien yhteydessä

Yksikkötestissä testataan pieni eristetty osuus tai moduuli sovelluksen koodista. Se testaa yleensä yksittäisen aliohjelman (subroutine) ja vahvistaa, että tietyillä lähtöarvoilla saadaan oikea tulos. Jos aliohjelmalla on riippuvuuksia toiseen moduuliin, se jäljitellään. Integraatiotesteillä vahvistetaan, että sovelluksen eri osat toimivat keskenään ja että eri osien oletukset toisistaan ovat oikeat. Integraatiotesteissäkin jäljitellään sovellusten osia tai integraatiokohteita, jotka eivät ole oleellisia kyseisen testin kannalta. Suorituskykytes-

teissä testaan, kuinka tehokkaasti sovellus toimii. Testissä mitataan, miten kauan kestää operaation aloittamisesta sen valmistumiseen sekä kuinka tiheästi useampia peräkkäisiä tai rinnakkaisia operaatioita voi tehdä. Käyttöliittymätesteissä eli päästä päähän -testeissä [20] käytetään sovellusta automaatin avulla, kuten ihminen käyttäisi. Automaatio simuloi vuorovaikutusta sovelluksen kanssa web-selaimen kautta.

Uudet testit auttavat tunnistamaan virheitä välittömästi. Sovelluskehityksen ohella kirjoitetut testit vahvistavat uuden koodin toimivuuden käytännössä saman tien. Vanhat testit varmistavat, että koodin muutokset eivät ole rikkoneet aikaisempaa toteutusta. Näin toteutuu regressiotestaus [36]. Jokaisen koodimuutoksen yhteydessä voi ajaa kaiken tyyppisiä testejä. Regressiotestaus on erityisen arvokasta, kun sovelluksen elinkaari ja kehitys jatkuu pitkään. Sovelluskehittäjät todennäköisesti eivät osaa ottaa huomioon kaikkea tietystä muutoskohteesta riippuvia kokonaisuuksia muutoksia tehdessään. Lisäksi työntekijät voivat vaihtua, jolloin kehittäjällä ei ole sovelluksen toiminnasta käsitystä. Testiregressio ilmaisee muutoksien aiheuttaneen sovelluksen toimintaa oletuksien vastaiseksi, kun taas ilman tällaisia testejä sovelluksen rikkoutuminen havaitaan vasta myöhemmin. Tässä tilanteessa joko uuden koodin on otettava huomioon myös vanhat oletukset tai vaatimuksista riippuen vanhan koodin oletuksia on muutettava. Testiregression luotettavuus edellyttää, että testikattavuus on korkea, eli mahdollisimman suuri osa sovelluskoodista suoritetaan testien aikana.

Testien ajaminen avustaa tiedon koostamisen toteutusta kehitystyön aikana. Eri tietolähteiden perusteella muodostettu koostenäkymä tai tiivistelmä on helppoa saattaa toimivaksi, kun koodia voi testata paloittain ja usein. Koosteeseen voi liittyä sekä tietolähteistä saatavia yksittäisiä tietokenttiä tai niistä johdettuja tai laskettuja arvoja. Testit ovat tehokas keino varmistaa tuloksien oikeellisuus. Tiedon oikeellisuus prosessin kannalta vahvistuu vasta hyväksymistestauksessa, mutta mahdollisimman aikaisessa mahdollisimman monen virheen poistaminen helpottaa myös loppujenkin virheiden havaitsemista hyväksymistestauksessa.

5.2 Arvioinnin puutteet

Arkkitehtuurin toimivuuden ja tehokkuuden takaamiseksi on suoritettava myös teknisiä arviointeja. Tässä työssä ei ole tehty kvantitatiivisesti mittauksia, jotka osoittaisivat arkkitehtuurin tehokkuuden. Sen edellytykseksi vaadittaisiin perusteellinen kuvaus arkkitehtuurista, mikä ei ole mahdollista tietojärjestelmissä, joiden tarkkoja yksityiskohtaisia kuvauksia ei voi julkisesti esittää. Arkkitehtuuria ei vertailla muihin vastaavalaisiin toteutuksiin, koska vertailu on hankalaa. Muista toteutuksista ei saatavilla julkisia arkkitehtuuriratkaisujen kuvauksia.

5.3 Jatkokehityskohteet

Esitysvalmistelu-sovellusta kehitetään lähitulevaisuudessa käsittelemään enemmän rahoituspalveluita. Tämän seurauksena sovellukseen tulee muun muassa lisää moduuleita käyttöliittymään, arviointikysymyksiä ja taloustarkastuksia. Toteutuksen nojautuminen parametrien konfiguraatioon mahdollistaa sen, että osa rahoituspalveluiden välisistä eroista ei vaadi lainkaan koodimuutoksia sovelluksen varsinaiseen koodiin. Testeissä uudet käyttötapaukset on kuitenkin otettava huomioon. Lisäksi uudenlaiset taloustarkistukset, joihin ei ole tehtävissä yleistettyä ratkaisua, vaatii lisäominaisuuksien ohjelmointia. Laajempien rahoituspalveluiden osalta myös täydennyksien visualisoinnin parantaminen nousee yhä tärkeämmäksi.

Taloustarkistuksien määrän kasvaessa myös välimuistirakenteisiin tulee muutostarpeita. Tietoa on jaettava yhä useampiin eri palasiin ja integraatiokohteista saatavien eri resurssien lukumäärä voi kasvaa. Lisäksi prosessiin kuuluvista tarkistuksista on todennäköisesti vietävä tietoja myös ulospäin. Tarkistuksissa tuotettua tietoa halutaan hyödyntää myöhemmin rahoitussovelluksissa tai kolmannen osapuolen järjestelmissä.

Sovelluksen laajentuessa on syytä tehdä käyttöliittymän parannuksia. Sovelluksen elinkaaren mittaan monimutkaisempien rahoituspalvelujen ja muuttuvien vaatimuksien myötä käyttöliittymän moduulijaottelut ja niiden tiivistelmät voi olla hyvä tarkistaa uudelleen. Käyttöliittymän ominaisuuksien ja kenttien lisääntyessä kokonaisuuksien johdonmukaisuus ja loogisuus sekä tarkoituksenmukainen visuaalinen ryhmittely voi käydä epäselväksi.

Tapahtumiin pohjautuva tiedon päivittäminen on oleellinen osa rahoitusjärjestelmien toiminnassa. Ajastettuihin kyselyihin nojautuva tapahtumapohjainen arkkitehtuuri on käytössä rahoitussovelluksissa yleisemminkin. Viestijonoihin ja valjasti kytkettyihin pohjautuva arkkitehtuuri toisi etuja tietojenhaun suorituskyvyssä.

Tietojärjestelmillä voi automatisoida prosessia vapauttaen ihmisresursseja. Esimerkiksi hylkäysesityksen tekeminen täysin automaattisesti on mahdollista, kun tietyt kriteerit rahoitusasiassa jäävät täyttymättä. Esimerkiksi hakijan riittävän huono luottoluokitus johtaa suoraan hylkyyn.

6 YHTEENVETO

Tässä työssä suunniteltiin, arvioitiin ja toteutettiin uuden sovelluksen arkkitehtuurivaihtoehtoja päätöksentekoon vaadittavan tiedon koostamiseksi viranomaispäätösjärjestelmässä. Työssä toteutettiin uusi sovellus Business Finlandin rahoitusprosessin esitysvalmisteluvaihetta varten työntekijöiden käyttöön. Sovellukseen toteutettiin integraatioita toisiin prosessiin liittyviin sovelluksiin sekä integraatiopalveluihin. Esitysvalmistelun uudistusta tehdään vaiheistetusti ottaen huomioon jatkokehitettävyyden seuraavia vaiheita varten. Ensimmäisessä vaiheessa käsitellään ainoastaan yhden rahoituspalvelun rahoitusasioita.

Tutkimuskysymys *K1) Miten tietoja yhdistetään eri lähteistä yhdeksi kokonaisuudeksi?* pureutuu tiedon koostamisen ongelmaan. Viranomaisen rahoituspäätöksen tekoon pohjana olevaan rahoitusesitykseen on koostettava tietoa monesta lähteestä. Yhdistettynä on asiakkaan hakemuksen tiedot, ulkoisten taloustietojärjestelmien tarjoamat tiedot, sisäisten rahoitusprosessin operatiivisten koostepalveluiden tiedot sekä esitysvalmistelijan täyttämät tiedot. Näistä muodostetaan yhtenäinen ja johdonmukainen kokonaisuus rahoitusprosessin edistämiseksi. Tietoja koostetaan useasta lähteestä yhteen sovellukseen dataintegraatiotekniikoilla. Integraatioihin on rakennettu välimuistirakenteita sovelluksen käyttöliittymän käytön sujuvuuden edellyttämänä. Eri lähteistä koostetuista tiedoista muodostetaan käyttöliittymään loogisia kokonaisuuksia. Moduulijaottelu mahdollistaa sekä keskittymisen sopivaan kokonaisuuteen yksityiskohtaisesti että tiedon häviämiseltä välttymisen samanaikaisten muutoksien yhteydessä.

Tutkimuskysymys *K2) Kuinka ajan funktiona muuttuva tieto pidetään ajan tasalla?* etsii vastauksia ajan funktiona muuttuvan tiedon reagointiin liittyviä ratkaisuja. Ajan funktiona muuttuvaa tietoa haetaan taustalla automaattisesti ajoittain. Ulkoisissa järjestelmissä tapahtuu ennustamattomissa olevia tiedon päivityksiä, joihin halutaan reagoida ilman, että ihmiskäyttäjien on aktiivisesti kyseltävä tiedon muutoksia. Muuttunut tieto tuodaan käyttöliittymään automaattisesti. Yksittäisten tietohakujen koostaminen listausnäkymään vaatii tietojen tallentamisen sovellukseen, josta ne voidaan palvella käyttöliittymälle nopeasti. Taustalla tehtävät tiedonhaut on syytä ajastaa siten, että niiden kuormitus jakautuu ajallisesti tasaisesti. Jos mahdollista, tehokkain ratkaisu on rakentaa ajastetun kyselyn sijaan viestinkuuntelija, joka odottaa tietoja tiedon muuttumisesta järjestelmän aktiivisen kyselyn sijaan.

Tutkimuskysymys *K3) Miten kehitetään sovellusta sujuvasti ja mahdollistetaan jatkokehitys?* tutkii menetelmiä, joilla sovelluskehitys voidaan pitää ketteränä ja jatkuvana. Tuo-

teomistajan alustavien vaatimuksien perusteella tehty toteutusehdotus antaa pohjaa sovelluksen iteratiiviselle kehitykselle. Kehitysversioita on hyvä antaa tuoteomistajan testattavaksi, minkä pohjalta seuraavien iteraatioiden vaatimukset tarkentuvat. Tuoteomistajien vankka prosessiymmärrys on arvokas resurssi kehitystyössä ja iteraatioissa. Loppukäyttäjien palautteen kerääminen tarjoaa hyvää palautetta sovelluksen viimeistelyyn sekä vahvistaa sovelluksen toimintojen oikeellisuutta. Kehitystyössä ulkoisten integraatiokohteiden jäljitelmillä emuloidaan ulkoisia tiedonlähteitä mahdollistaen ympäristön, joka muistuttaa todellista tuotantoympäristöä. Jäljitelty data on sepitettyä mutta oikean muotoista. Testiautomaatiolla vahvistetaan myös sovelluksen vakaata ja odotetunlaista toimintaa testaamalla sovelluksen toimintalogiikkaa sekä eri osien yhteistoimintaa.

Sovelluksen toiminnallisten tavoitteiden saavuttaminen voidaan todeta onnistuneeksi, jos järjestelmä on otettu käyttöön, käyttäjien palaute ei sisällä kriittisiä ongelmia ja sovelluksen jatkokehitys on mahdollistettu kestävästi. Rahoitusprosessin vaiheet ja yksityiskohdat ovat laissa määritettyjä ja viranomaisten linjaamia. Jos prosessi toimii, voidaan tehdä johtopäätös, että sovellus on tarkoitukseensa kelvollinen.

LÄHTEET

[1] *Asiakkaan dokumentaatiojärjestelmä (Wiki)*. Sisäinen.

[2] E. Bertino, K. Takahashi ja I. Books24x7. *Identity Management: concepts, technologies, and systems*. English. Boston, MA; London: Artech House, 2010; 2011, 21–43. ISBN: 1608070409.

[3] *Business Finland*. Business Finland, verkkosivu. 2018. URL: http://www.businessfinland.fi (viitattu 16. 11. 2018).

[4] A. Bhumgara ja I. Sayyed. ENTERPRISE RESOURCE PLANNING SYSTEMS. English. *International Journal of Advances in Engineering & Technology* 10.2 (2017), 283.

[5] C. A. Binildas ja I. ebrary. *Service oriented Java business integration: enterprise service bus integration solutions for Java developers*. English. Birmingham: Packt, 2008.

[6] F. P. Brooks. *Design of design: essays from a computer scientist*. 2010.

[7] A. Doan, A. Halevy, Z. G. Ives ja I. Books24x7. *Principles of data integration*. English. 1. painos. Waltham, MA: Morgan Kaufmann, 2012. ISBN: 9780124160446;9780123914

[8] *domain modelling*. English. 2016. ISBN: 0199-688974.

[9] P. Eugster, P. Felber, R. Guerraoui ja A.-M. Kermarrec. The many faces of publish/-subscribe. English. *ACM Computing Surveys (CSUR)* 35.2 (2003), 114–131.

[10] S. Faranello. *Balsamiq Wireframes Quickstart Guide*. English. 1. painos. Birmingham: Packt Pub, 2012. ISBN: 9781849693523.

[11] R. T. Fielding. Architectural styles and the design of network-based software architectures. Tohtorinväitöskirja. University of California, Irvine, 2000.

[12] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach ja T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. http://www.rfc-editor.org/rfc/rfc2616.txt. RFC Editor, kesäkuu 1999. URL: http://www.rfc-editor.org/rfc/rfc2616.txt.

[13] A. D. Giordano. *Data Integration Blueprint and Modeling: Techniques for a Scalable and Sustainable Architecture*. Pearson Education, 2010.

[14] J. L. Harrington ja J. L. Harrington. *Relational database design and implementation: clearly explained*. 3rd. Amsterdam; Boston: Morgan Kaufmann/Elsevier, 2009.

[15] A. R. Hevner, S. T. March, J. Park ja S. Ram. Design Science in Information Systems Research. *MIS Quarterly* 28.1 (2004), 75–105. ISSN: 02767783. URL: http://www.jstor.org/stable/25148625.

[16] G. Hohpe ja B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Signature Series (Fowler). Pearson Education, 2012. ISBN: 9780133065107.

[17] J. Humble ja D. Farley. *Continuous delivery*. English. Upper Saddle River (N.J.): Addison-Wesley, 2011. ISBN: 9780321601919.

[18] *IdP (Identity Provider)*. URL: http://kb.mit.edu/confluence/display/glossary/IdP%20(Identity%20Provider).

[19] P. Johannesson, E. Perjons, S. universitet, S. fakulteten ja I. för data- och systemvetenskap. *An Introduction to Design Science*. English. 2014. painos. Cham: Springer International Publishing AG, 2014. ISBN: 3319106317.

[20] V. Kähkönen. Web-sovelluksen tehokas päästä päähän -testiautomaatio. Finnish. Tutkielma. Tampereen teknillinen yliopisto, 2017.

[21] E. Lukin. *Innovaatioseteli startupeille*. URL: https://www.businessfinland.fi/suomalaisille-asiakkaille/palvelut/rahoitus/startup/innovaatioseteli/.

[22] J. Mertz ja I. Nunes. Understanding Application-Level Caching in Web Applications: A Comprehensive Introduction and Survey of State-of-the-Art Approaches. *ACM Computing Surveys (CSUR)* 50.6 (2018; 2017), 1–34.

[23] *Oracle Concepts*. URL: https://docs.oracle.com/cd/E11882_01/server.112/e40540/toc.htm.

[24] *Osallistumispyyntö kilpailulliseen neuvottelumenettelyyn. Tekesin operatiivisten sovelluksien ylläpito ja kehitys*. 2014. URL: https://docplayer.fi/46000034-Osallistumispyynto-julkinen-65-25-dm.html.

[25] N. Pathania. *Pro Continuous Delivery: With Jenkins 2.0*. English. 1. painos. Place of publication not identified: Apress, 2017. ISBN: 9781484229125.

[26] *PRH - YTJ ja Y-tunnus*. URL: https://www.prh.fi/fi/kaupparekisteri/yleista/ytj.html.

[27] *React – A JavaScript library for building user interfaces*. URL: https://reactjs.org/.

[28] *Richardson Maturity Model*. URL: https://martinfowler.com/articles/richardsonMaturityM
html.

[29] D. Rountree ja I. Castrillo. *Basics of cloud computing: understanding the fundamentals of cloud computing in Theory and Practice*. English. Amsterdam: Elsevier Syngress, 2014. ISBN: 0124055214.

[30] *Single-Request-Response*. URL: https://www.w3.org/2000/xp/Group/1/10/11/2001-10-11-SRR-Transport_MEP.

[31] D. Spadini, M. Aniche, M. Bruntink ja A. Bacchelli. Mock objects for testing java systems: Why and how developers use them, and how they evolve. English. *Empirical Software Engineering* (2018).

[32] *spring.io*. URL: https://spring.io/.

[33] *Suomen Asiakastieto Oy - Parhaat päätökset helposti - Suomen Asiakastieto Oy*. URL: http://www.asiakastieto.fi/.

[34] J. Thones. Microservices. English. *IEEE Software* 32.1 (2015), 113.

[35] *Verotus*. URL: https://www.vero.fi/harmaa-talous-rikollisuus/torjunta/verotus/.

[36] J. Watkins ja S. Mills. *Testing IT: an off-the-shelf software testing process*. English. 2nd. New York: Cambridge University Press, 2011. ISBN: 1139010298.

[37] *What are microservices?* URL: https://microservices.io/.

[38] *YTJ - Yritys- ja yhteisötietojärjestelmä - Yrityshaku*. URL: https://tietopalvelu.ytj.fi/.