Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka
Marko Kekäläinen

Opinnäytetyö

Agilefant-projektinhallintatyökalun SOA-rajapintatoteutus ja integrointi Oikeat Oliot Oy:n tuntikirjanpitojärjestelmään

Ohjaava opettaja lehtori Erkki Hietalahti
Työn teettäjä Oikeat Oliot Oy, valvojana laatupäällikkö Antti Sinisalo
Tampere 2010

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma, Ohjelmistotekniikka

Tekijä	Marko Kekäläinen
Työn nimi	Agilefant-projektinhallintatyökalun SOA-rajapintatoteutus ja integrointi Oikeat Oliot Oy:n tuntikirjanpitojärjestelmään.
Sivumäärä	34
Valmistumisaika	05/2010
Ohjaava opettaja	lehtori Erkki Hietalahti
Työn tilaaja	Oikeat Oliot Oy, valvojana laatupäällikkö Antti Sinisalo

---

Tiivistelmä

Opinnäytetyön aihe käsittelee organisaatioiden järjestelmäintegraatiota ja valittua SOA-arkkitehtuurin mukaista toteutustapaa. Organisaatiot elävät jatkuvassa muutoksessa, joka synnyttää aina uusia haasteita myös käytetyille tietojärjestelmille. Järjestelmäintegraatio on yksi keskeisimmistä tekniikoista pyrittäessä kustannustehokkaasti vastaamaan syntyneisiin muutostarpeisiin.

Tehdyssä opinnäytetyössä yksinkertaistettiin Oikeat Oliot Oy:n sisäinen tuntienkirjausmenettely integroimalla operatiivinen Agilefant-projektinhallintatyökalu ja yrityksen Tukki-tuntikirjanpitojärjestelmä. Integraatiototeutus mahdollistaa Agilefantiin tehtyjen tuntikirjausten siirtämisen automatisoidusti tuntikirjanpitojärjestelmään, jolloin vältytään kahteen järjestelmään tehtävältä kirjaamismenettelyltä.

Agilefant on suomalainen, avoimen lähdekoodin projektinhallintatyökalu, jota kehitetään Teknillisen korkeakoulun alaisessa ohjelmistoliiketoiminnan ja -tuotannon laboratoriossa. Järjestelmäintegraation mahdollistamiseksi Java EE -ohjelmistoalustalle kehitettyyn projektinhallintatyökaluun toteutettiin ReST-arkkitehtuurin mukainen palvelurajapinta.

Avainsanat organisaatioiden järjestelmäintegraatio, SOA, ReST, Java EE

Tampere University of Applied Sciences
Computer Science, Software Engineering

Author	Marko Kekäläinen
Work label	Enterprise integration based on service-oriented architechture.
Number of pages	34
Graduation time	May 2010
Thesis supervisor	lecturer Erkki Hietalahti
Commissioned by	Oikeat Oliot Oy, Head of Quality Antti Sinisalo

---

Abstract

Topic of this thesis is tighly bonded to Enterprise application integration. Organizations live in constant change which generates a lot of new challenges to software systems they use. Organization application integration is one indisputable tool for responding to those challenges.

Practically the major goal is thesis was to prevent overlapping in Oikeat Oliot Oy's workers procedure of subscribing hour entries. The procedure was simplified by integrating the operative Agilefant project management tool and Tukki accounting system. Implementation provides automated hour entry transfer from Agilefant to Tukki and consequently removes the occasion to submit entries for both systems.

Agilefant is modern Java EE based open source tool aimed for managing agile software processes. The Agilefant is pure finnish product engineered by the laboratory of software business and production in department of Information Technologies in Aalto University School of Science and Technology. To enable the integration the ReSTful Web Service interface was implemented to Agilefant.

Key words Enterprise Integration, SOA, ReST, Java EE

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma, Ohjelmistotekniikka

Esipuhe

Opinnäytetyö toteutettiin Oikeat Oliot Oy:lle vuoden 2010 ensimmäisellä puoliskolla. Projekti kasvatti huomattavasti omaa osaamistani Java EE -ohjelmistoalustasta, järjestelmäintegraatioista sekä ketterän kehityksen periaatteilla toteutettavista ohjelmistotuotantoprosesseista.

Kiitän suuresti valvoja Antti Sinisaloa sekä konsultti Niko Munteria projektin aikana saaduista valveutuneista teknisistä ja projektinhallinnollisista neuvoista. Lisäksi kiitos toimitusjohtaja Jukka Aunulle projektin resurssien järjestymisestä.

Tampereella huhtikuussa 2010

Marko Kekäläinen

Sisällysluettelo

Käytetyt lyhenteet ja termit

Scrum	Ketterä ohjelmistokehitysmenetelmä.
Agilefant	Avoimen lähdekoodin projektinhallintatyökalu.
SOA	Palvelukeskeinen arkkitehtuuri (Service-oriented Architecture).
Web Service	W3C:n määrittelemä ohjelmistojärjestelmä, joka mahdollistaa keskenään yhteensopivien tietokoneiden vuorovaikutuksen tietoverkon avulla. Muodostuu kolmesta protokollasta: SOAP, WSDL ja UDDI.
Enterprise Integration Patterns	Kokoelma järjestelmäintegraatioissa hyödynnettäviä ratkaisumalleja.
Messaging	Viestittämiseen perustuva integraation suunnittelutyyli.
Send - and -forget	Messaging-tyylin mahdollistama asetelma, jossa viestin lähettäjä voi unohtaa viestin sen lähetettyään.
Callback-menettely	Asynkronisessa viestittelyssä voidaan informoida lähettäjää viestin onnistumisesta ns. ”takaisinkutsulla”.
ReST	Representational State Transfer. Hypermediajärjestelmien toteutuksessa hyödynnettävä arkkitehtuurityyli.
Messaging-system	Viestijärjestelmä, joka huolehtii Messaging -tyyppisessä järjestelmäintegraatiossa mm. viestien välittämisestä.
XML	eXtensive Markup Language on tietotekniikassa yleisesti käytetty rakenteellinen merkintäkieli.
URI	Uniform Resource Identifier. Osoite, jolla viitataan esim. johonkin tiettyyn hypermediaresurssiin.
CRUD	Tietotekniikassa tiedon pysyvyyden (engl. persistence) kuvaamiseen käytetty termistö: create, read, update ja delete.
WADL	Web Application Description Languate on verkkopalveluiden kuvaukseen käytetty XML-pohjainen merkintäkieli.
MIME	Multipurpose Internet Mail Extensions. MIME-tyypillä voidaan HTTP -tiedonsiirrossa määrittää mm. siirrettävän tiedon formaatti.
Java EE	Java Enterprise Edition on erityisesti organisaatioiden tarpeisiin kehitetty Java-sovellusalusta.

Spring Framework	Avoimen lähdekoodin Java-sovelluskehys.
Hibernate	Relaatiotietokantojen oliomallintamiseen soveltuva avoimen lähdekoodin Java-kirjasto.
DI (Dependency Injection)	Tekniikka, jolla olioiden väliset riippuvuudet ulkoistetaan erilliselle DI-säiliölle.
Struts2	Avoimen lähdekoodin Java-sovelluskehys Web-sovellusten käyttöliittymäkerroksen kehittämiseksi.
Tukki	Oikeat Oliot Oy:n sisäisenä kehitysprojektina toteutettu tuntikirjanpitojärjestelmä.
dom4j	Avoimen lähdekoodin kirjasto XML-dokumenttien käsittelyyn.
DSL	Domain Specific Language tarkoittaa pientä, johonkin tiettyyn tarkoitukseen laadittua ohjelmointikieltä.
POJO	Plain Old Java Object, Java-ohjelmoinnissa käytetty luokkatyyppi, joka tarjoaa yksinkertaisen ohjelmointirajapinnan get- ja set-metodeilla.
DAO	Data Access Object, relaatiotietokantojen oliomallinnuksessa käytetty tietokantakyselyiden rajapintaluokka.

1 Johdanto

Organisaatiot joutuvat toimialasta riippumatta aika ajoin päivittämään tietojärjestelmäkokonaisuuksia, jotta ne tukisivat mahdollisimman hyvin vallitsevia toimintamalleja. Tällöin on usein kyse järjestelmäintegraatiosta, jolla pyritään vastaamaan organisaation tietojärjestelmältä edellytettäviin muutoksiin. Muutostarve on usein seurausta organisaatiossa syntyneelle tarpeelle kehittää jotain tiettyä liiketoiminnan osa-aluetta tai sisäistä toimintatapaa. Käytännössä kyse voi olla esimerkiksi uuden järjestelmän käyttöönotosta, johon usein liittyy myös tarve sen integroimisesta osaksi laajempaa kokonaisuutta.

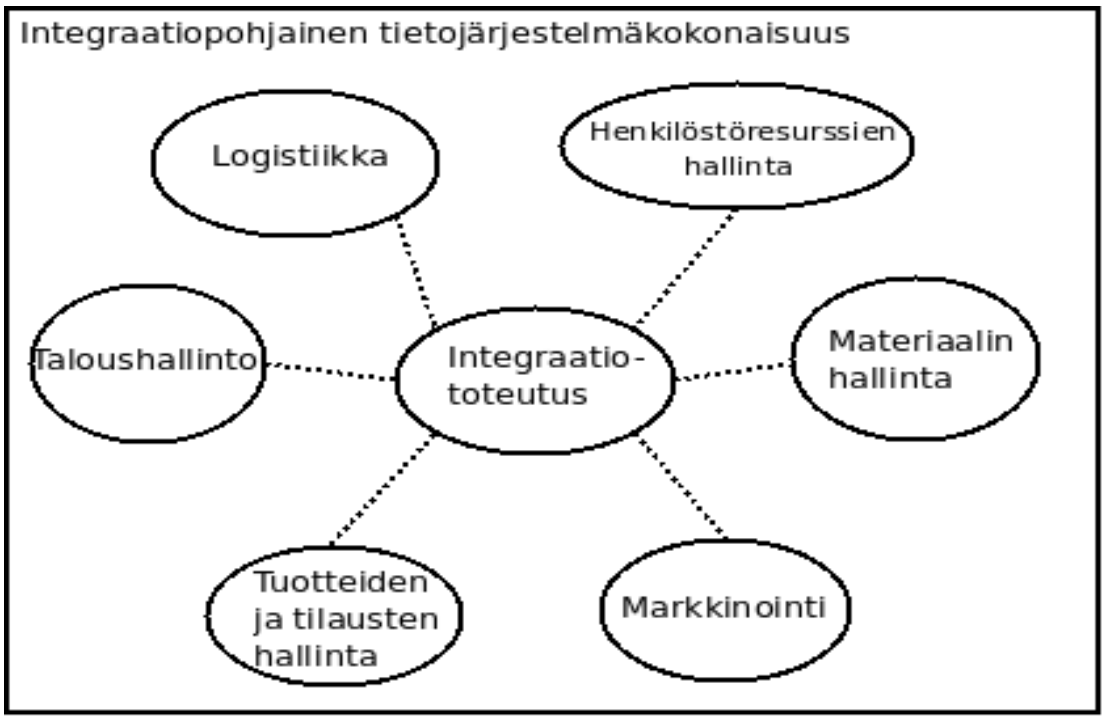
Tässä opinnäytetyössä kehitettiin Oikeat Oliot Oy:n operatiivista projektityöskentelyä ja siihen olennaisena osana kuuluvaa tuntienkirjausmenettelyä. Tuntien kirjaaminen selkiytettiin integroimalla Agilefant-projektinhallintatyökalu yrityksen Tukki-tuntikirjanpitojärjestelmään. Integraatio mahdollistaa Agilefantiin tehtyjen tuntikirjausten viennin automatisoidusti myös Tukkiin, jolloin poistuu tarve kirjata tunteja kahteen eri järjestelmään. Agilefant on suomalainen avoimen lähdekoodin projektinhallintatyökalu, jota yritys hyödyntää ketterän Scrum-prosessin mukaisissa ohjelmistokehitysprojekteissa. Integraation mahdollistamiseksi Agilefantiin toteutettiin SOA-arkkitehtuurin mukainen ReSTful Web Services -palvelurajapinta.

Työssä pyrittiin kiinnittämään erityishuomiota integraatiokokonaisuuden suunnitteluun. Kirja ”Enterprise Integration Patterns” (Hohpe, Gregor, Woolf, Bobby, 2009) tarjoaa joukon organisaatioiden järjestelmäintegraatioissa hyödynnettäviä, ns. ”viestinvälitykseen” (engl. Messaging) soveltuvia ratkaisumalleja. Työssä käytetty SOA-arkkitehtuuri pyrkii tarjoamaan joustavan ja laajalti hyödynnettävän ratkaisumallin järjestelmäintegraatioiden tunnettuihin haasteisiin. Käytännössä SOA määrittelee joukon suunnittelussa sovellettavia periaatteita, jotka mahdollistavat toisilleen ennestään tuntemattomien ohjelmistojen vuorovaikuttamisen.

2 Organisaatioiden järjestelmäintegraatio

Organisaation järjestelmäintegraatiolla tarkoitetaan organisaation käytössä olevien tietojärjestelmien ja tietovarastojen välistä rajoittamatonta tiedon jakamista. Harvat yritysohjelmistot toimivat tänä päivänä itsenäisesti ympäristössä, jossa niillä ei ole tarvetta kommunikoida muiden ympäröivien järjestelmien kanssa. Sen sijaan on tyypillistä, että eri organisaatioiden tietojärjestelmät koostuvat useista tiettyä työrutiinia palvelevista ohjelmistoista, jotka vuorovaikuttavat keskenään halutulla tavalla.

Kuvassa 1 on hahmoteltu liiketoimintaa harjoittavan kuvitteellisen organisaation tietojärjestelmäkokonaisuus, joka muodostuu yksittäisiä osa-alueita palvelevista pienemmistä järjestelmistä. Integraation avulla järjestelmät on yhdistetty kommunikoimaan keskenään halutulla tavalla. Esimerkiksi taloushallinnon järjestelmälle voi olla hyötyä henkilöstöresursseja hallinnoivan järjestelmän sisältämästä, yrityksen työntekijöiden käyttöasteeseen liittyvästä tiedosta. Koska integraatioita voidaan toteuttaa lukuisilla eri tekniikoilla ei kuva ota kantaa tekniseen toteutustapaan.



Kuva 1. Malliesitys useista eri ohjelmistoista muodostetusta tietojärjestelmäkokonaisuudesta.

Lisäksi eri organisaatioiden liiketoimintaprosessit ja toimintaympäristöt elävät jatkuvassa muutoksessa. Tämä aiheuttaa luonnollisesti haasteita yritettäessä pitää tietojärjestelmät ajan tasalla vallitsevien käytäntöjen kanssa. Hyvin suunnitellulla järjestelmäintegraatiolla voidaan tehokkaasti vastata syntyneisiin haasteisiin. Onnistuneella järjestelmäintegraatiolla voidaan myös pidentää käytettävän järjestelmäkokonaisuuden käyttöikää, jolloin siihen tehdyn investoinnin kannattavuus paranee.

2.1 Organisaatioiden järjestelmäintegraation haasteet ja hyödyt

Organisaatiossa toteutettavalla järjestelmäintegraatiolla voidaan saavuttaa huomattavaa hyötyä, mutta niiden toteuttaminen on yhtä lailla erittäin haastavaa. Organisaatiokohtaisten erikoisvaatimusten lisäksi integraatioiden toteuttamisessa on omat tekniset erityispiirteensä, jotka täytyy ottaa huomioon.

Ensinnäkin täytyy muistaa, että tietoverkot ovat haavoittuvia. Jos tietoverkon lisäksi otetaan huomioon integraation piiriin kuuluvat yksittäiset laitteistokokonaisuudet ja integroituvaan ohjelmistoon vaikuttavat muut ohjelmistot, huomataan, että yksittäisen viestin siirtoon käytettävä tiedonsiirtoväylä on hyvin altis mitä moninaisimmille häiriöille.

Erityishuomiota kannattaa myös kiinnittää integraation toteuttavien ohjelmistojen erilaisuuteen. Ohjelmistot on usein toteutettu eri ohjelmointikielillä, jolloin niiden tapa käsitellä tietoa eroaa aina toisistaan jollain tavalla, jolloin esimerkiksi tiedon eri koodausformaatit voivat aiheuttaa ongelmia. Myös ohjelmistojen alustariippuvuudet on otettava huomioon, sillä ohjelmistot käyttävät aina jossain määrin esimerkiksi käyttöjärjestelmäkohtaisia käyttöjärjestelmän tarjoamia palveluita.

Ehkä merkittävin huomioon otettava seikka on ymmärtää jatkuva muutos, jonka keskellä eri organisaatioiden järjestelmäintegraatiot elävät. Suunnittelun kannalta on merkittävää minimoida integraation piiriin kuuluvien tietojärjestelmien väliset riippuvuudet. On kestämätön tilanne, jos yhteen järjestelmään tehty muutos heijastuu välittömästi myös muihin järjestelmiin. Tämän takia järjestelmien yhdistäminen kannattaa toteuttaa ns. ”kevyen sidoksen” (engl. loose coupling) periaatteella (Hophe &

Woolf 2009, xxix).

2.2 Järjestelmäintegraatiototeutuksen tekniset lähestymistavat

Tietotekninen ala elää ehkä nopeimmin etenevän teknologisen muutoksen kourissa, jossa erilaisten teknisten menetelmien tiheä kiertokulku on jo muodostunut alan ominaispiirteeksi. Tästä johtuen myös järjestelmäintegraatioita on kautta aikojen toteutettu mitä erilaisimmilla teknologioilla (mm. CORBA). Tästä huolimatta asiantuntijat ovat määritelleet järjestelmäintegraation tekniselle toteutustavalle neljä erilaista teknologioista riippumatonta lähestymistapaa:

1. *File transfer (suom. tiedoston siirto)* – Lähetettävä tieto kirjoitetaan tiedostoon, jota toinen sovellus lukee. Sovelluksilla täytyy olla yhteinen sopimus tiedoston käsittelyyn liittyvistä seikoista, kuten esimerkiksi käytetystä merkistökoodauksesta.
2. *Shared database (suom. jaettu tietokanta)* – Järjestelmät käyttävät yhteistä tietokantaa, jolloin niiden välisiä suoria riippuvuussuhteita ei ole.
3. *Remote Procedure Invocation (suom. etäfunktiokutsu)* – Järjestelmä toteuttaa toiminnallisuuden, johon sallitaan pääsy myös järjestelmän ulkopuolelta.
4. *Messaging (suom. viestinvälitys)* – Järjestelmät luovat yhteiseen käytettyyn viestikanavaan viestejä, josta niitä voidaan asynkronisesti myös vastaanottaa.

On täysin tapauskohtaista, mikä mainituista lähestymistavoista soveltuu parhaiten suunniteltuun järjestelmäintegraatioon. Niillä kaikilla on omat vahvuutensa ja heikkoutensa. Lähestymistavan hyödyntäminen ei ole ehdotonta, eli niitä voidaan vapaasti soveltaa keskenään niin, että eri osat integraatiosta käyttävät sitä lähestymistapaa, joka siihen parhaiten soveltuu (Hophe & Woolf 2009, xxx).

2.3 Messaging-tyyppinen järjestelmäintegraatiomalli

Koska opinnäytetyössä toteutettua järjestelmäintegraatiota lähestyttiin Messaging-tyyppistä lähestymistapaa käyttäen, käsitellään sitä myös tässä yhteydessä tavallista tarkemmin. Messaging-tyyppinen integraatiomalli voidaan ymmärtää helposti, kun ajatellaan puhelinliikenteen perustoimintamallia. Tavallisen puhelun onnistuminen perusedellytys on, että puhelun molemmat osapuolet ovat saman aikaisesti käytettävissä, jolloin tiedon siirtyminen puhujalta puhujalle voi tapahtua.

Aikakriittisyys tekee puhelun onnistumisesta haastavaa, mikäli osapuolet ovat kiireisiä. Puhelinvastaajat poistavat reaaliaikaisen puhelun edellyttämän aikakriittisyyden. Viestin lähettäjä eli tässä tapauksessa soittaja saa viestiketjun toteutumisen kannalta aina halutun viestin välitettyä. Viesti joko vastaanotetaan reaaliajassa normaalisti toteutuneen puhelun muodossa, tai myöhemmin haluttuna ajankohtana nauhoitetun puhelun muodossa. Asynkronisuus siis lisää huomattavasti integraatiomallin käytettävyyttä, mutta samalla se asettaa myös omat vaatimuksensa integroitaville järjestelmille (Hophe & Woolf 2009, xxx).

Vertailtaessa eri lähestymistapojen ominaispiirteitä on tarpeellista tunnistaa niillä saavutettavat edut. Messaging-tyyppisellä integraatiomallilla saavutetaan seuraavan kaltaisia etuja verrattuna muihin lähestymistapoihin (Hophe & Woolf 2009, xxxiii):

- *Hajautettu kommunikaatio sovellusten välillä* – Esimerkiksi viestittely tavalla, jossa olioita muutetaan merkkijonoksi, tarjoaa järjestelmille luotettavan tavan kommunikoida tietoverkon välityksellä.
- *Ohjelmistoalustojen ja kielten integroitavuus* – Integroitavat järjestelmät ovat usein vuosien saatossa yksittäisten kehittäjäryhmien suunnittelemia. Teknisten eroavaisuuksia lisäksi mallin avulla mahdollistetaan eri kieltä käyttävien järjestelmien integrointi. Yhteensovittamisessa hyödynnetään yleensä erillistä sovellusta.
- *Asynkroninen kommunikaatio* – Malli mahdollistaa järjestelmien toiminnan ns. ”send - and - forget”-periaatteella. Eli kun lähettävä järjestelmä on lähettänyt viestin ja kun lähetys on suoritettu onnistuneesti, on kommunikaatio sen osalta suoritettu.
- *Riippumattomuus ajastukselle* – Kommunikoivat osapuolet voivat lähettää viestejä välittämättä vastaanottavan osapuolen nopeudesta vastaanottaa niitä.
- *Viestien puskurointi ruuhkatilanteessa* – Vältetään tietyn järjestelmän osan ylikuormittuminen tai mahdollinen kaatuminen puskuroimalla siihen osoitettuja viestejä.
- *Kommunikaation luotettavuus* – Esimerkiksi RPC-malliin verrattuna kommunikaatio on luotettavampaa, koska viestijärjestelmä (engl. Messaging

System) tallentaa viestin, ennen sen lähettämistä vastaanottajalle. Tällä menettelyllä varaudutaan vastaanottavan järjestelmän mahdolliseen toimintahäiriöön tai ruuhkatilanteeseen. Tallennettua viestiä välitetään vastaanottajalle, kunnes viestin välittäminen onnistuu.

- *Yhteydetön kommunikaatio* – Esimerkiksi osa kannettavista päätelaitteista on pääosan toiminta-ajastaan yhteydettömässä tilassa. Messaging-malli tukee mainiosti esimerkiksi laitteiden synkronointia silloin, kuin niillä on datayhteys käytettävissä.
- *Järjestelmien välittäminen (engl. Mediation)* – Viestijärjestelmä esittäytyy integroiduille järjestelmille välittäjänä, jonka valikoimasta ne voivat valita palvelut, joihin haluavat integroitua. Mikäli yksittäiseen järjestelmään ilmaantuu häiriö, ei sen palauduttuaan tarvitse muodostaa kaikkia integraatioyhteyksiä uudestaan, vaan riittää, kun se muodostaa yhteyden viestijärjestelmään.
- *Säikeiden hallinta (engl. Thread management)* – Säikeiden hallinnalla tarkoitetaan sitä, että kommunikoivan järjestelmän ei tarvitse jäädä odottamaan toiselta järjestelmältä saapuvaa vastausta, vaan viestittely voidaan toteuttaa ns. callback-menettelyä käyttäen, jolloin yksi järjestelmän säie jää odottamaan vastausta viestiin ja muu osa järjestelmästä jatkaa normaalisti tehtävän suorittamista.

Messaging-malli ei useista hyötynäkökulmistaan huolimatta ole ongelmaton ratkaisu, vaan myös sillä on heikkoutensa, jotka paljolti johtuvat juuri asynkronisuudesta (Hophe & Woolf 2009, xxxvi):

- *Monimutkaisesti ohjelmoitavissa* – Asynkronisuus pakottaa sovelluskehittäjät toteuttamaan ei tapahtumakriittisiä sovelluksia, jotka osaavat ottaa huomioon viestijärjestelmästä aiheutuvia viestien satunnaisia lähetysaikoja.
- *Vaiheistus ongelmat* – Joskus asynkronisuudelle on omat rajoituksensa. Jos lähetetyn viestin täytyy saavuttaa kohde esimerkiksi saman vuorokauden aikana, kuin se on lähetetty, tuo se lisähaasteetta viestijärjestelmän toteuttamiselle.
- *Synkroniset viestit* – Synkronisten ja asynkronisten viestien käsittely samassa viestijärjestelmässä on haastavaa.

- *Suorituskyky* – Viestien käsittely viestijärjestelmässä aiheuttaa aina viivettä kommunikaatioon.

2.4 Organisaatioiden järjestelmäintegraatiolle aiheuttamat erikoisvaatimukset

Organisaatioissa toteutetuilla järjestelmäintegraatioilla voidaan siis päivittää ja jatkokehittää jo olemassa olevia tietojärjestelmiä, mutta ne soveltuvat myös hyvin suurien järjestelmäkokonaisuuksien toteuttamiseen. On likimain mahdoton ajatus toteuttaa yhtä suurta järjestelmää, joka kattaisi kaikki kuviteltavat liiketoiminnalliset, taloushallinnolliset ym. prosessit. Tästä johtuen ajaudutaan usein tilanteeseen, jossa on syntynyt tarve kahden tai useamman yrityksen prosesseja tukevan järjestelmän väliseen kommunikaatioon. Ei myöskään pidä vähätellä järjestelmäintegraation tuomaa mahdollisuutta valita kuhunkin tehtävään parhaiten soveltuva järjestelmä yhden raskaan ja jälkikäteen vaikeasti muunneltavan järjestelmän sijaan.

Organisaatioiden järjestelmäintegraatiot ovat erittäin haastavia toteuttaa. Ohjelmistontoimittajilla on tarjolla valmiita tuotekokonaisuuksia, joilla voidaan hyvin kattavasti hallita aikaisemmin esitettyjä teknisiä haasteita. Ne ovat kuitenkin valitettavasti vain murto-osa vallitsevista ongelmista, sillä esimerkiksi liike-elämän omat vallitsevat lainalaisuudet sekä organisaatioissa alati vallitseva politikointi vaikuttavat yllättävän paljon teknisten seikkojen kehittymiseen.

Integraation onnistuminen kaikkien osapuolten kannalta edullisimmalla tavalla edellyttää laadukasta kommunikaatiota. Yksittäisten ohjelmistojen sekä laajempien järjestelmäkokonaisuuksien ongelmana on usein se, että ne määrittävät loppujen lopuksi ihmisten ja organisaatioiden toimintaprosesseja, kun alun perin lähtökohtana on ollut kehittää ne tukemaan vallitsevia käytäntöjä. On siis ensiarvoisen tärkeää lisätä kommunikaatiota kaikkien sidosryhmien kesken, jotta todella voidaan saada selville integraatiolla tavoiteltavat asiat.

Integraatiototeutukset johtavat usein tilanteeseen, jossa toteutetun järjestelmäkokonaisuuden toimintavarmuus on kokonaisuuden kannalta ratkaisevan tärkeää. Pahimmassa tapauksessa järjestelmän kaatuminen voi lamauttaa koko

organisaation. Sen takia on tärkeää suunnitella integraatio niin, ettei yksittäisen järjestelmän osan kaatuminen aiheuta kokonaisuuden kannalta kriittistä uhkaa.

Ohjelmistokehitystä varjostaa myös voimakas pyrkimys omaleimaisuuteen, joka ilmenee toteutettujen ratkaisuiden suojaamisena lainsäädännöllisin ja teknisin keinoin. Tämä aiheuttaa sen, että integraatioiden parissa työskentelevät tahot eivät pysty vaikuttamaan osallisten järjestelmien toimintaan kuin tietyissä rajoissa. Niin sanotut ”suljetut ratkaisut” aiheuttavat näin ollen paljon päänvaivaa, kun toteutuksessa joudutaan kiertämään havaittuja ongelman lähteitä sen sijaan, että vaikutettaisiin suoraan ongelmia aiheuttavaan ohjelmistoon.

Vaikka järjestelmäintegraatioita on toteutettu jo vuosikymmenten ajan, kärsii myös tämä ohjelmistotekniikan osa-alue standardien puutteesta. Muutamien vahvojen standardien (XML ja Web Service) lisäksi integraatioita toteutetaan vieläkin mitä moninaisimmilla tekniikoilla, jolloin vakiintuneet käytännöt ja niiden tuoma kokonaisetu jäävät syntymättä.

Näiden järjestelmien ylläpito on myös näin ollen erittäin haastavaa, puhumattakaan jatkokehityksestä. Sen hallittavuus esimerkiksi yksittäisen virheen etsimisen kannalta järjestelmässä, joka voi koostua kymmenistä tai jopa sadoista ohjelmistoista, on erittäin haastavaa, jollei mahdotonta. Tästä huolimatta on itsestään selvää, että eri liiketoimintaprosessien muuttuessa säilyy myös tarve tietojärjestelmien muutokselle.

2.5 Integraatiotyypit

Kirja ”Enterprise Integration Patterns” kerää yhteen organisaatioiden järjestelmäintegraatioon tuotettua kirjallista tietoa sekä eri projekteista saatuja käytännön kokemuksia. Se ei pyri sanelemaan eksakteja menetelmiä eikä käsitteitä, vaan ennemminkin kokoaa yhteen jo tunnettuja tosiasioita käsiteltävästä aiheesta. Vaikka standardien ja menetelmien olemattomuus on johtanut mitä erilaisimpiin tapoihin toteuttaa organisaatioiden järjestelmäintegraatioita, voidaan ne kokemusperäisesti siitä huolimatta jaotella karkeasti kuuteen eri kategoriaan (Hophe & Woolf 2009, 5).

- *Tietoportaali (engl. Information portals)* – Useiden eri järjestelmien käyttö

keskitetään yhteen – näytön pieniin osiin jakavaan – portaali -sovellukseen.

- *Tiedon toisinne (engl. Data replication)* – Useilla järjestelmillä voi olla monta kopiota samasta datasta. Tällä integraatiotyypillä huolehditaan yhteen tietolähteeseen tehdyn muutoksen heijastaminen myös muihin kopioihin.
- *Jaetut liiketoiminnot (engl. Shared business functions)* – Useiden eri järjestelmien käyttämät toiminnot jaetaan sen sijaan, että ylläpidetään samasta palvelusta päällekkäisiä toteutuksia.
- *SOA (Service Oriented Architechture)* – SOA-arkkitehtuurin mukaisella ratkaisulla on kaksi määrittävää tekijää, joiden mukaan se tarjoaa ulospäin selvän hakemiston saatavilla olevista palveluista ja selkeän määrityksen siitä, kuinka integraation piiriin kuuluviin palveluihin liitytään.
- *Jaettu liiketoimintaprosessi (engl. Shared business process)* – Yksittäinen liiketoiminto voi heijastua helposti myös kymmeniin muihin järjestelmiin. Esim. transaktioiden toteutumisen takia tähän tarkoitukseen voidaan toteuttaa erillinen ohjelmistokomponentti.
- *Liiketoimintaintegraatio (engl. Business to business integration)* – Edelliset tyypit käsittelevät organisaation sisällä tapahtuvaa integraatiota. Näiden lisäksi toteutetaan myös paljon erilaisten organisaatioiden välisiä integraatioita, joihin voidaan soveltaa edellä lueteltuja integraatiotyyppejä. Internetin yli siirrettävä tieto aiheuttaa integraation suunnittelun kannalta lisähuolenaiheita esimerkiksi tiedon mahdollisesta joutumisesta vääriin käsiin.

3 ReST (Representational State Transfer) -arkkitehtuuri

Agilefantiin laadittu SOA-arkkitehtuurin mukainen palvelurajapinta toteutettiin ReST-arkkitehtuurin mukaisesti. ReST on moderni, hajautettuihin hypermediajärjestelmiin kehitetty arkkitehtuurityyli. Roy Fielding esitteli ReSTin väitöskirjassaan ”Architectural styles and the design of network-based software architechtures”, (University of California, 2000). Arkkitehtuurin nimi ilmentää arkkitehtuurin toimintaperiaatetta:

- *Representational (esityksellinen)* – Verkkopalveluissa on aina kyse hypermediaresurssista, joka esitetään tavalla tai toisella.
- *State* – Asiakkaan näkemällä resurssilla on aina tila. Web -pohjaisessa

kanssakäymisessä on pohjimmiltaan aina kyse resurssin tilan päivittämisestä.

- *Transfer* – Resurssin tilaa päivitetään aina siirtämällä dataa asiakkaan ja palvelimen kesken. (Fielding, Dissertation 2000).

3.1 ReST-arkkitehtuurin mukaisen palvelun rajoitukset (engl. constraints)

Arkkitehtuurin ensisijainen tarkoitus on mahdollistaa toisistaan erillään kehittyvien itsenäisten järjestelmien vuorovaikuttaminen. Resursseihin ja niiden tilan päivityksiin perustuvan perusideologian tueksi on määritelty joukko rajoituksia, joilla pyritään varmistamaan palveluiden universaali yhteensopivuus:

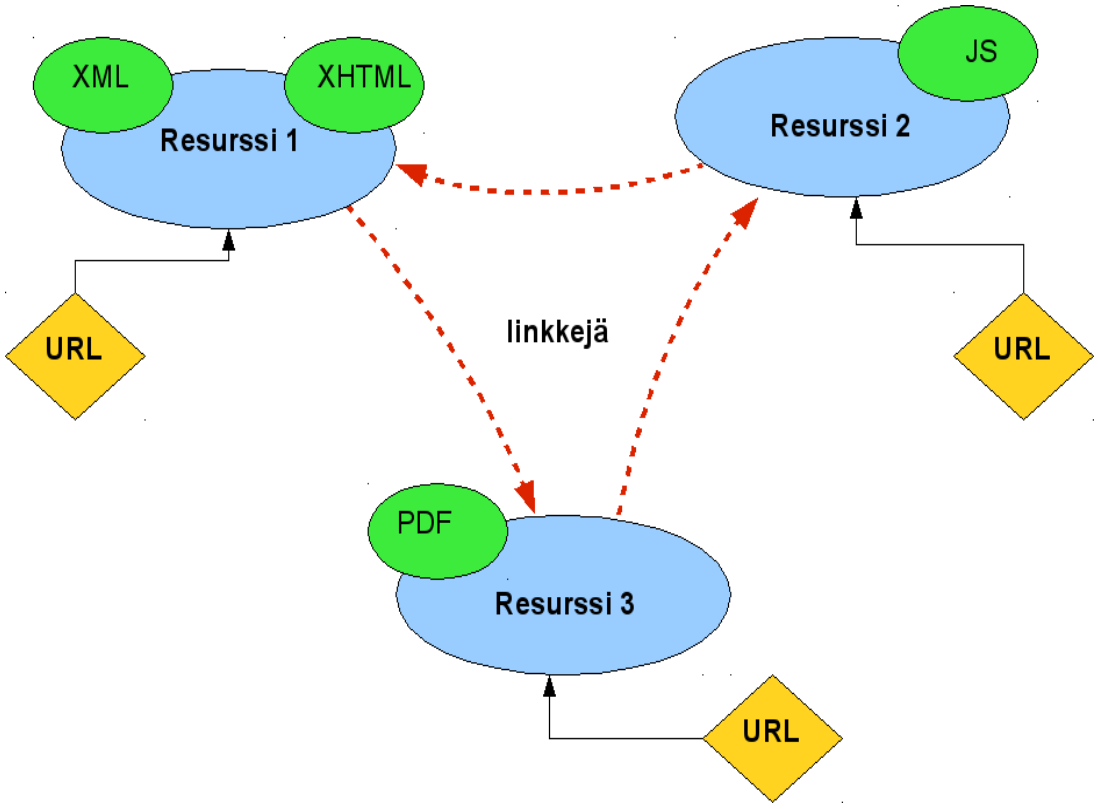
- ReST-palvelun tila ja toiminnallisuus on jaettu resursseihin (esimerkiksi staattisiin html-tiedostoihin).
- Jokaista resurssia voidaan osoittaa hypermedialinkkien käyttämän yleisen syntaksin avulla (esim. ”http://www.[palvelu].fi/[resurssi]).
- Resursseja käsitellään aina esitysten kautta.
- Kaikilla palveluilla tulee olla yhdenmukainen rajapinta (yleensä käytetään HTTP-protokollan mukaisia get-, post-, update- ja delete -metodeita).
- Viestien tulee olla selkeitä ja itseään selittäviä.
- Hypermedia toimii palvelun ”moottorina”, eli palvelimelta saatavan vastauksen tulee aina sisältää tieto siitä, mitä voidaan tehdä seuraavaksi. Näin sovellusta voidaan ajatella tilakoneena, jossa resurssit edustavat tiloja ja linkit tilasiirtymiä.

ReST-palvelut toteutetaan pääasiallisesti edellä mainittua HTTP-tiedonsiirtoprotokollaa käyttäen, jolloin yksittäisiä resursseja käsitellään protokollan mukaisilla get-, post-, put- ja delete -metodeilla. Standardin mukaiset metodit tarjoavat semantiikaltaan selkeän rajapinnan resurssien käsittelyyn (Fielding, Dissertation 2000):

- GET-metodilla luetaan (ts. kopioidaan) resurssi,
- POST-metodilla päivitetään resurssi,
- PUT-metodilla luodaan (ts. kirjoitetaan yli) resurssi,

- DELETE-metodilla tuhotaan resurssi.

Kuvassa 2 mallinnetaan ReST-arkkitehtuurin peruskonseptia. Resurssit (1, 2 ja 3) kuvaavat asioita, joita palvelun käyttäjät tarvitsevat (esimerkiksi projekti-, tuote- tai käyttäjälistaus). Verkossa julkaistavaan yksittäiseen resurssiin viitataan yksikäsitteisellä URL (Unified Resource Locator):lla. Kun resurssi ladataan palvelimelta, sen tulee myös sisältää linkit muihin mahdollisiin resursseihin. Kuvassa linkkien avulla tapahtuvia resurssien välisiä siirtymiä kuvataan punaisilla nuolilla. Lisäksi yksittäinen hypermediaresurssi sisältää tiedon sen esitysmuodosta (kuvassa vihreällä).



Kuva 2. Kuvaus ReST-arkkitehtuurin toiminnallisesta peruskonseptista.

3.2 ReST-resurssien kuvaaminen WADL-kielellä

ReST-arkkitehtuurin mukaisia web-resursseja ja palveluita kuvataan yleisimmin WADL-kielellä (Web Application Description Language). WADL on XML-pohjainen merkintäkieli, jolla voidaan kuvata seuraavat kriteerit täyttävä verkkoresurssi:

- Pohjautuu Web-arkkitehtuuriin ja -infrastruktuuriin (HTTP).
- On riippumaton ohjelmointikielestä, sekä -alustasta.
- Mahdollistaa yhdistämisen muiden Web- ja työpöytäsovellusten kanssa.
- Palvelut on nimetty johdonmukaisesti.

Koodiesimerkissä 1 esitetään yksinkertaisen ReST-palvelun WADL-kuvaus. WADL-spesifikaation mukaisen yksittäisen WADL-kuvauksen täytyy sisältää verkkopalvelusta seuraavat tiedot (kohdat 1, 2 ja 4 merkitty esimerkkiin):

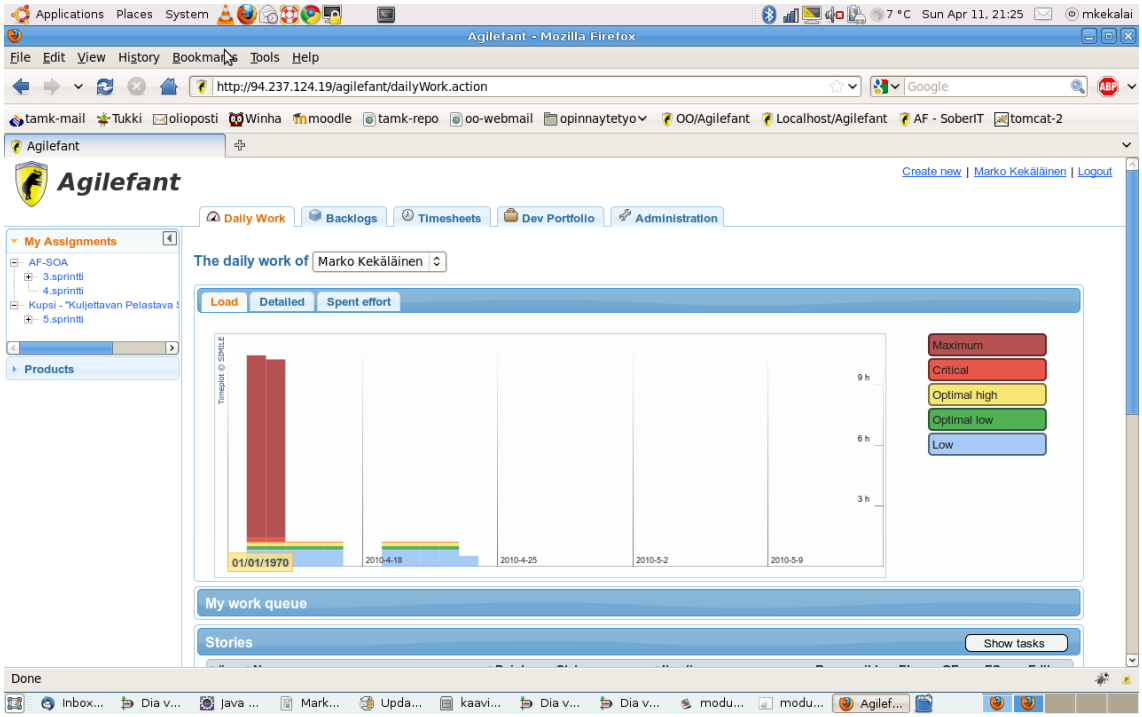
1. Resurssijoukon, esimerkiksi luettelomaisen esityksen tarjotuista resursseista.
2. Resurssien suhteet, esimerkiksi mahdolliset viittaukset tai järjestykseen liittyvät lisätiedot.
3. Operaatiot, esimerkiksi tuetut HTTP-operaatiot, niiden syötteet, vasteet sekä tuetut formaatit.
4. Resurssin kuvaustavat, kuten esimerkiksi tuetut MIME-tyypit tai mahdolliset skeemat.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <application xmlns="http://research.sun.com/wadl/2006/10">
    <doc xmlns:jersey="http://jersey.dev.java.net/"
jersey:generatedBy="Jersey: 0.10-ea-SNAPSHOT 08/27/2008/>
          <resources base="http://localhost:9998/">
        1.  <resource path="/helloworld">
         3.   <method name="GET" id="getClichedMessage">
               <response>
              4.<representation mediaType="text/plain"/>
                </response>
               </method>
             </resource>
          </resources>
   </application>
```

Koodiesimerkki 1. Yksittäisen ReST-palvelun WSDL-esimerkkikuvaus.

4 Järjestelmäintegraation suunnittelu

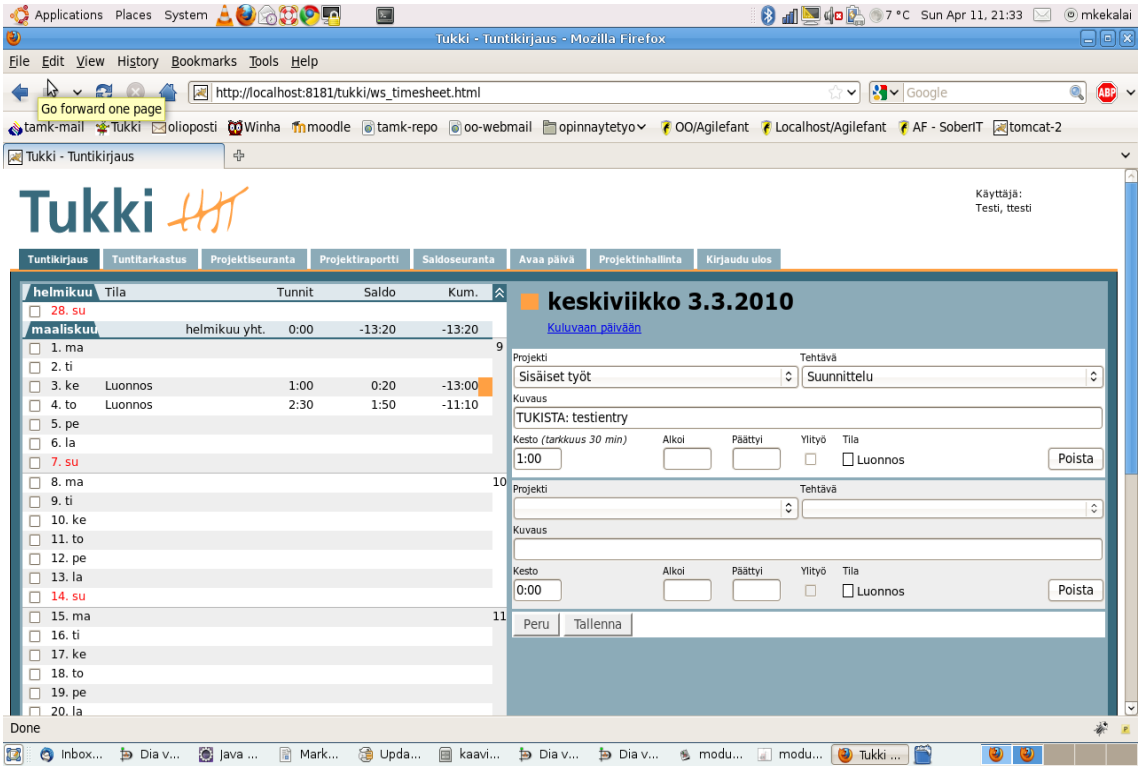
Integraation päätavoite oli mahdollistaa Agilefant-projektinhallintatyökaluun (kuvassa 3) tehtävien tuntikirjausten vienti automatisoidusti myös toimeksiantajayrityksen Tukki-tuntikirjanpitojärjestelmään (kuvassa 4). Yksittäisen yrityksen toiminnan kannalta tuntikirjanpitojärjestelmä on yksi tärkeimmistä järjestelmistä ja tästä johtuen se on myös yksi toimintakriittisimpiä. Yrityksen taloushallinnon perustana tuntikirjanpitojärjestelmän toiminnalla ja sen tietosisällön oikeellisuudella on ratkaiseva merkitys. Tämä tosiasia oli otettava huomioon myös toteutetun tuntien massasiirto-ominaisuuden suunnittelussa.



Kuva 3. Agilefant-projektinhallintatyökalun käyttäjäkohtainen päänäkymä.

Viestiketjun kriittisin osa tiedon mahdollisen vääristymisen kannalta on erillinen viestijärjestelmäohjelmisto, jossa suoritetaan kaikki siirrettävän tiedon väliprosessointi. Tähän aiheeseen palataan tarkemmin viestijärjestelmän suunnittelua käsittelevässä luvussa 4.2.

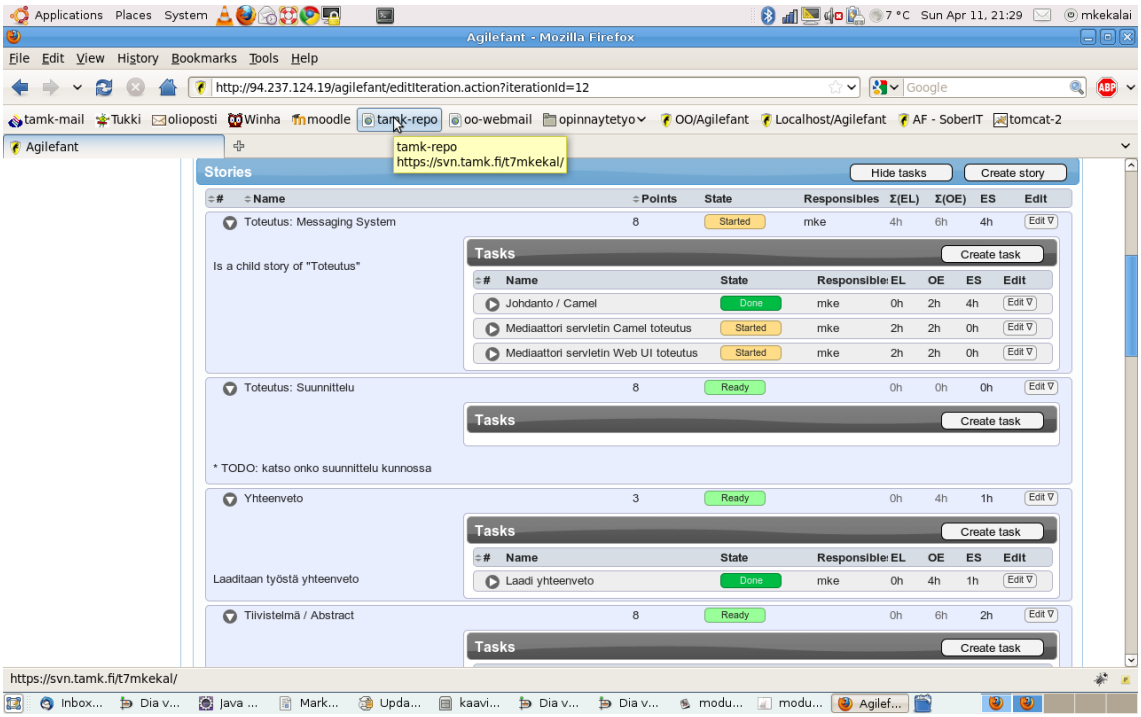
Toinen suunnittelun kannalta merkittävä asia oli mahdollisiin häiriötilanteisiin reagointi. Java EE -ohjelmistoalusta itsessään on hyvin stabiili, mutta palvelinlaitteiston elinikä on aina rajallinen. Palvelimen alasajosta seuraa aina myös ajattavien ohjelmien alasajo, joka voidaan mieltää yksittäisen ohjelmiston kannalta häiriötilanteeksi. Onnistuneiden massasiirtojen kirjoittaminen lokitiedostoon on yksinkertainen mutta riittävä ratkaisu tämän ongelman ratkaisemiseksi. Kokonaisuuden kannalta merkitsevä tieto on se, onko massasiirto suoritettu vai ei.



Kuva 4. Tukki-tuntikirjanpitojärjestelmän tuntikirjausnäkymä.

4.1 Agilefantin SOA-rajapinnan suunnittelu

Agilefant on operatiivinen projektinhallintatyökalu, joka tukee ns. ketterien menetelmien mukaisia ohjelmistokehitysprosesseja. Scrum-prosessi perustuu pelkistäen yksittäiseltä tuotteelta toivottujen priorisoitujen ominaisuuksien toteuttamiseen. Yksittäiset ominaisuudet puretaan toteutusprosessissa pieniksi tehtäviksi, joille on helpompaa tehdä aika-arvioita. Kuvassa 5 esitetään Agilefantiin tehdyn projektin tehtävänäkymää, josta ilmenee listattuja ominaisuuksia (engl. stories) ja ominaisuuksien alle määriteltyjä tehtäviä. Projektin toteutusvaiheessa yksittäisille tehtäville kirjataan tuntitoteutumia.



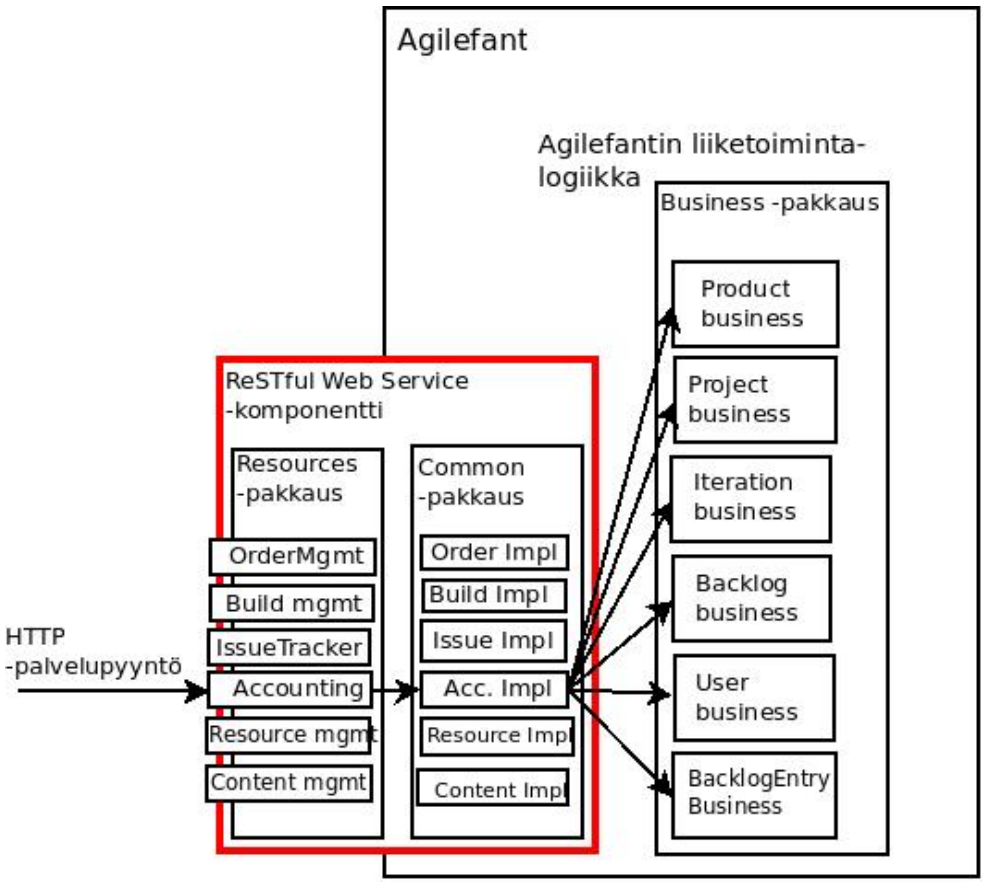
Kuva 5. Agilefantiin tehdyn projektin alaisten tarinoiden ja tehtävien listaus.

Rajapintakomponentin suunnittelun perustaksi lähdettiin selvittämään ohjelmistoprojektien eri sidosryhmien tarpeita. Projekteissa on aina niiden luonteesta riippuen erilaisia sidosryhmiä, joiden tarpeet eroavat toisistaan. Tämän takia rajapintakomponentin suunnittelun lähtökohtana pidettiin yleisesti tunnettuja, yritysten liiketoiminnan kannalta oleellisia tukitoimintoja (esim. taloushallinto, tilausten hallinta ym. (kuva 1, sivu 2)).

Erilaisten liiketoimintaa harjoittavien organisaatioiden tukitoiminnoissa on paljon yhteneväisyyksiä. Järjestelmäintegraation kannalta on olennaista tunnistaa tukitoimintojen välisiä yleistettävissä olevia tarve suhteita. Yleistyksiä suunnittelussa hyödyntämällä voidaan toteuttaa ohjelmakomponentteja, joita voidaan mahdollisesti myös uusiokäyttää. Näin ollen myös rajapintakomponentin suunnittelussa keskeinen teema oli pyrkiä toteuttamaan yleiskäyttöinen ohjelmakomponentti, jota voidaan mahdollisesti hyödyntää myös tulevaisuudessa.

Rajapintakomponentin arkkitehtuuri tukee edellä mainittua pyrkimystä. Kuvassa 6 on esitetty toteutetun ReSTful Web Service -rajapintakomponentin arkkitehtuuri ja sen vuorovaikuttaminen Agilefantin sovelluslogiikan kanssa. Rajapinnan toiminta on pyritty

toteuttamaan edellisessä kappaleessa mainittuja tukitoimintoja ja tarpeita silmällä pitäen. Kuvasta nähdään, kuinka rajapinnan modulaarinen rakenne tukee ajattelumallia. Rajapinnan tarjoamat palvelut (esim. kirjanpidolle ja tilausten hallinnalle) voidaan mieltää erillisinä moduuleina ja näin ollen on myös luontevaa, että niitä edustavat erilliset Java-luokat.



Kuva 6. Agilefantin ReSTful Web Service -rajapinnan arkkitehtuuri.

Rajapintakomponentti (kuvassa punaisella) koostuu kahdesta Java-pakkauksesta. Komponentin sisällä vasemmalla puolella esitetyn Resources-pakkauksen luokat muodostavat varsinaisen palvelurajapinnan, jossa HTTP-palvelupyynnöt käsitellään. Oikeanpuoleisen Common-pakkauksen sisältämät vastaavasti nimetyt Java-luokat sisältävät varsinaisen toimintalogiikan, joka koostaa XML-muotoisen vastausdokumentin hyödyntämällä Agilefantin liiketoimintalogiikan tarjoamia palveluita. Agilefantin liiketoimintalogiikka on myös koottu omaan Java-pakkaukseen, jolloin rajapintatoteutus pysyy myös sen suhteen selkeänä.

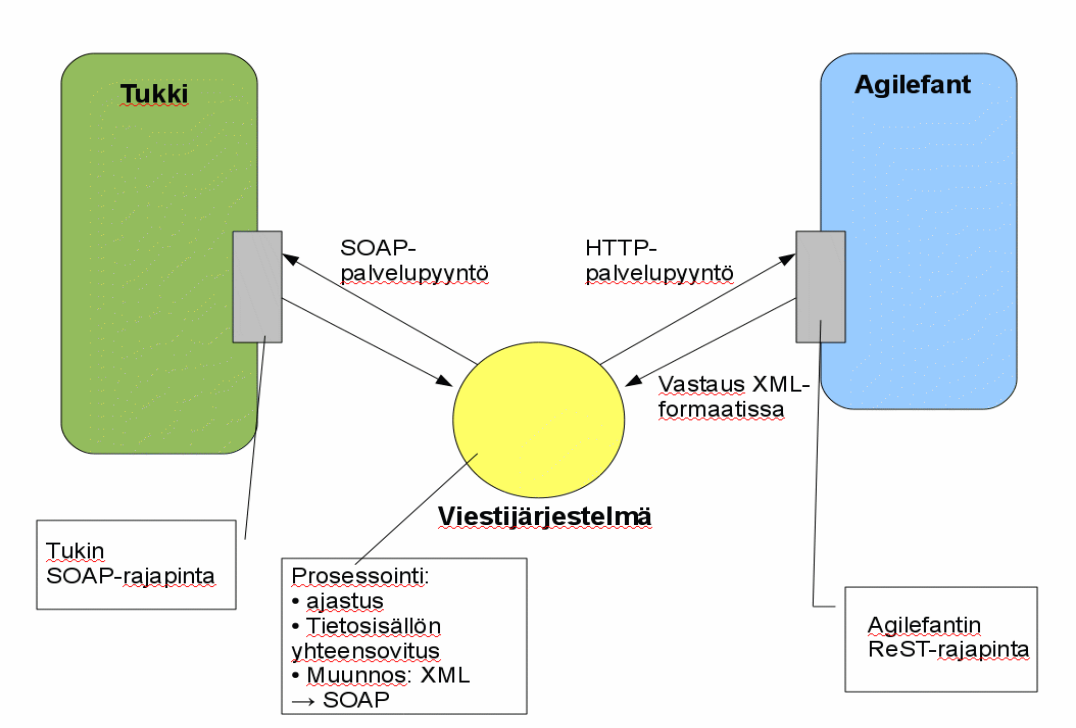
Yksittäinen Resources-pakkauksen luokka tarjoaa tietyn valikoiman palveluita. Esimerkiksi kirjanpidon tarpeisiin toteutettu Accounting-luokka sisältää muun muassa seuraavat palvelut:

- tuntitapahtumien haku käyttäjän ja ajanjakson perusteella
- tuntitapahtumien haku käyttäjän ja päivän perusteella
- tuntitapahtumien haku projektin ja päivän perusteella

4.2 Viestijärjestelmän suunnittelu

Messaging-tyyppisessä integraatiomallissa olennainen integraation osa on ns. ”viestijärjestelmä” (engl. Messaging System). Viestijärjestelmä on edellytys sille, että integraatiototeutuksessa saavutetaan kaikki integraatiomallin hyödyt. Viestijärjestelmä huolehtii käytännössä viestien välittämisestä integraation piirissä olevien sovellusten kesken.

Kuvasta 7 voidaan havaita viestijärjestelmän rooli integraatiossa. Kokonaisuuden kannalta komponentin rooli on kiistaton, sillä se tekee aloitteen viestien vaihdosta, välittää viestit sekä lisäksi suorittaa järjestelmien kommunikaation mahdollistavan väliprosessoinnin.



Kuva 7. Integraation kokonaisarkkitehtuuri.

Kahden erilaisen ohjelmiston integroinnissa on aina omat yksityiskohtaiset vaatimuksensa. Tässä tapauksessa yksi suunnittelun haasteista oli erilaisten dataformaattien väliset muunnokset. Agilefantin palvelurajapinta palauttaa pyydettävän tietosisällön XML-formaatissa. Tukki-tuntikirjanpitojärjestelmästä löytyy valmis Web Service -liityntä, joka edellyttää SOAP (Simple Object Access Protocol) -protokollan mukaista viestittelyä. Näiden kahden formaatin välisen muunnoksen mahdollistamiseksi viestijärjestelmään toteutettiin ns. Message Translator -mallin (Hophe & Woolf 2009, 85) mukainen käännösominaisuus.

Formaattien välisen muunnoksen lisäksi viestijärjestelmätoteutuksessa täytyi ottaa huomioon Agilefantin ja Tukin tietosisältöjen rakenteellinen ja käsitteellinen yhteensovitus. Järjestelmien käsittelemä tietosisältö on yksittäisen tuntikirjauksen osalta yhdenmukainen. Ilmenneet ongelmat liittyivät koodiesimerkissä 2 esitettyihin Agilefantin ja Tukin tietorakenteiden käsitteellisiin eroavaisuuksiin.

Tukin tietomallissa yksittäisen tuntikirjauksen omistaa aina projektille kuuluva tehtävä. Tietomalli tukee lisäksi tehtävälle lisättäviä alitehtäviä. Alitehtäviä voidaan lisätä myös alitehtäville, jolloin rakenteesta voi tulla rajoittamattoman monikerroksinen. Agilefantin tietomalli käyttää yksiselitteisen *tehtävä* -termin sijaan Scrum-prosessin mukaisia *tarinoita* ja *tehtäviä*. Tarinalla tarkoitetaan pelkistetysti sanoen toteutettavan tuotteen yksittäistä ominaisuutta ja tehtävällä ominaisuuden toteuttamiseksi tehtävää selkeästi hahmotettavaa osatehtävää.

```
Tukin tietomalli:
→ projekti
    → tehtävä
        → alitehtävä (voidaan määrittää rajoittamaton määrä alitehtäviä)
            → tuntikirjaus
Agilefantin tietomalli:
→ projekti
    → tarina (voidaan määrittää rajoittamaton määrä alitarinoita)
        → tehtävä (ei pakollinen)
            → tuntikirjaus
```

Koodiesimerkki 2. Tukin ja Agilefantin käyttämät tuntikirjauksen käsitemallit.

Tuntikirjanpitojärjestelmä ei asettanut teknisiä rajoitteita Agilefantissa ilmennettävän mallin mukaisten tuntikirjausten viemiselle Tukkiin. Tästä oli hyötyä integraatiossa toteutetun sovelletun mallin rakentamisessa. Mallissa ongelmalliset Agilefantin tarinat ja alitarinat sovitetaan rakenteellisena esityksenä Tukissa käytettyihin tehtäviin ja alitehtäviin. Koska Agilefantissa tehtävälle ei voida määrittää alitehtävää, syntyy sovitetusta mallista helposti ymmärrettävä. Tehtävärakenteen alin alitehtävä on näin ollen tehtävä ja muut mahdolliset tehtävät tarinoita. Sovellettu malli on hahmoteltu koodiesimerkissä 3.

```
→ Tukki: projekti = Agilefant: projekti
  → Tukki: tehtävä = Agilefant: tarina
    → Tukki: alitehtävä = Agilefant: alitarina
      → Tukki: alitehtävä = Agilefant: tehtävä
        → Tukki: tuntitapahtuma = Agilefant: tuntitapahtuma
```

Koodiesimerkki 3. Integraatiossa käytetty sovellettu käsitemalli.

Tuntitapahtumien vienti Agilefantista Tukkiin toteutettiin ns. peilaamalla, jolla tässä yhteydessä tarkoitetaan Agilefantiin tehtyjen tuntikirjausten kopioimista tuntikirjanpitojärjestelmään muuttumattomina. Kuitenkin niin, ettei samasta tuntitapahtumasta luoda useaa ilmentymää. Käytännössä tämä tarkoittaa sitä, ettei esimerkiksi päällekkäisillä tai liian useasti toistuvilla massasiirroilla voida aiheuttaa Tukissa tietosisällön vääristymistä.

Järjestelmäintegraation viestinvälityksestä ja viestien väliprosessoinneista muodostuva toiminnallinen kokonaisuus määritellään viestijärjestelmään ohjelmallisesti tehtävässä reitityskonfiguraatiossa (koodiesimerkki 9, sivu 33). Käytännössä viestijärjestelmä välittää laaditun konfiguraation mukaiset palvelupyynnöt - tietyn aikaraamin sisällä (< 10 sec.) - satunnaisella ajan hetkellä. Satunnaista viivettä järjestelmään aiheuttaa viestijärjestelmässä ja palvelurajapinnoissa suoritettava tiedon prosessointi. Koska integraatiossa ei tarvittu erillistä viestien tahdistamiseen liittyvää ominaisuutta, voitiin viestijärjestelmän reititystoiminnot toteuttaa yksinkertaisen Message Channel -periaateen mukaisesti.

Suunnittelun viimeinen merkittävä haaste liittyi viestien lähetysmekanismiin sekä viestien jaksottamiseen. Tuntitoteutumatietojen sopivaksi päivitystiheydeksi sovittiin määrittelyvaiheessa vuorokausi. Aloite tietojen massasiirrolle synnytetään viestijärjestelmään konfiguroidulla ajastetulla toiminnolla. Ajastuskonfiguraation mukaisesti viestijärjestelmä suorittaa reitityskonfiguraation mukaisen massasiirron järjestelmien välillä hyödyntäen Agilefantin ja Tukin palvelurajapintoja.

5 Agilefantin SOA-rajapintatoteutus

Projektin ensimmäinen työvaihe oli toteuttaa Agilefant-projektinhallintatyökaluun SOA-rajapinta, jolla mahdollistettiin Agilefantin tietosisällön hyödyntäminen myös ulkopuolisissa järjestelmissä. Rajapinnan ensimmäinen versio ei mahdollista tiedon viemistä Agilefantiin. Web Service -tekniikka on ollut jo vuosien ajan vallitseva toteutustapa SOA-ratkaisuille. Vasta viime vuosina myös ReST-arkkitehtuurin mukaiset ns. ReSTful Web Service -toteutukset ovat yleistyneet merkittävästi.

Kahta edellä mainittua toteutustapaa vertailtaessa täytyy muistaa, että Web Service on W3C (World Wide Web Consortium):n määrittelemä ohjelmistojärjestelmä, jonka protokollakenttä koostuu kolmesta XML-pohjaisesta komponentista (SOAP, WSDL ja UDDI (Universal Description Discovery and Integration)), kun taas ReST on teknologiariippumaton arkkitehtuuriratkaisu hajautettujen hypermediajärjestelmien toteuttamiseksi.

Agilefant on toteutukseltaan moninainen ja moderni Java EE -palvelinsovellus. Sovelluksen tietokantakerros hyödyntää Hibernaten tarjoamaa relaatiotietokantojen oliomallinnusominaisuutta. Agilefantin sisäisiä olioriippuvuuksia kontrolloidaan Spring Frameworkin tarjoamalla DI (Dependency Injection) -toiminnolla. Käyttöliittymäkerros on toteutettu Struts2-sovelluskehystä hyödyntäen. Näin ollen SOA-rajapinnan toteutuksessa oli huomioitava toiminnallinen konteksti sekä eri sovelluskehysten edellyttämän arkkitehtuurin säilyttäminen.

Hibernaten ja Spring Frameworkin ominaisuudet tarjoavat erittäin monipuolisen työkalukokoelman laadukkaan web-sovelluksen toteuttamiseksi, mutta niiden oikeanlainen käyttö edellyttää myös syvää teknistä perehtymistä. Mainitut teknologiat

asettivat rajapintatoteutukselle omat haasteensa, sillä esimerkiksi Springin IoC-ominaisuus edellyttää luokkien ja olioiden konfiguroimista sovelluksen kontekstiin konfiguraatiotiedostojen tai Java-annotaatioiden avulla.

5.1 ReSTful Web Service -palvelun toteutus

Java-palvelinsovelluksen ollessa kyseessä, on eri tarkoituksiin laadittujen avoimen lähdekoodin kirjastojen määrä merkittävä. Rajapintakomponentin ohjelmalogiikan toteuttamisen lisäksi toinen haastava tehtävä onkin parhaiten haluttuun tarkoitukseen soveltuvan kirjaston valitseminen. ReSTful Web Service -komponentin toteuttamisessa hyödynnettiin Jersey-kirjastoa, joka on Sun Microsystemsin JAX-RS (Java Api for ReSTful Web Services) kirjastolle laadittu avoimen lähdekoodin referenssitoteutus.

Jersey sisältää ominaisuudet niin varsinaisen ReST-rajapinnan toteuttamiseen, kuin myös HTTP-palvelukutsujen monipuoliseen käsittelyyn. Jersey tukee myös Springiä, jolloin sen integrointi Springiä hyödyntävään Agilefantiin voitiin tehdä vaivattomasti servlet-sovelluksen web.xml -konfiguraatiotiedostossa (koodiesimerkki 4, sivu 23).

Konfiguraatiossa ladataan ylätason Jersey-luokka (SpringServlet), joka konfiguroidaan erillisen konfiguraatioluokan (resourceConfigClass) avulla toteuttamaan laaditun rajapintapakkauksen (fi.oikeatoliot.agilefant.ws.rs.frontend) mukaista palvelurajapintaa. Servlet-mapping -tagien sisällä kohdennetaan kaikki sovellukselle polkuun ”/rs/” osoitetut palvelukutsut ReST-rajapintaan.

```
<servlet>
   <servlet-name>Jersey Spring</servlet-name>
   <servlet-class>
com.sun.jersey.spi.spring.container.servlet.SpringServlet
</servlet-class>
   <init-param>
      <param-name>
com.sun.jersey.config.property.resourceConfigClass </param-
name>
    <param-
value>com.sun.jersey.api.core.PackagesResourceConfig</param-
value>
  </init-param>
  <init-param>
      <param-
name>com.sun.jersey.config.property.packages</param-name>
      <param-value>fi.oikeatoliot.agilefant.ws.rs.frontend
</param-value>
  </init-param>
</servlet>

<servlet-mapping>
   <servlet-name>Jersey Spring</servlet-name>
   <url-pattern>/rs/*</url-pattern>
</servlet-mapping>
```

Koodiesimerkki 4. Jersey-komponentin integrointi Spring-sovellukseen web.xml -servletkonfiguraatiossa.

Toteutetun ReSTful Web Servicen palvelurajapintaa edustavat kuvassa 6 (sivu 17) esitetyt Resources-luokat. Koodiesimerkissä 5 kuvataan pelkistetyn Resource-luokan ja luokan yhden metodin rakenne. Käytännössä kyse on yksinkertaisista Javan POJO (Plain Old Java Object) -luokista, joiden metodit ovat palvelun toteuttavia get- ja set-funktioita. Lisäksi kannattaa huomioida modernit Java-annotaatiot (@Path, @Produces ym.), joilla esimerkissä konfiguroidaan luokan palveluiden URI:t ja MIME -tyypit.

```
@Path("/accounting")
@Component
@Scope("request")
public class AccountingResource {

    @Inject private AccountingResult accountingResult;

    @GET
    @Path("/projectBacklogHourEntries")
    @Produces("application/xml")
    public String getProjectBacklogHourEntries(
            @QueryParam("projectId") String projectId )
    {
     return
accountingResult.getProjectBacklogHourEntries(projectId);
    }
}
```

Koodiesimerkki 5. ReSTful Web Service-rajapintaluokka ja luokan jäsenfunktio.

Spring-tuen lisäksi Jerseystä löytyy myös muita hyödyllisiä lisäominaisuuksia. Esimerkiksi palvelun kuvausten käsittelyä varten Jersey tarjoaa valmiin ominaisuuden, jonka avulla voidaan joko tuottaa WADL-tiedoston avulla rajapintaa edustavat Java-luokat tai sen sijaan tuottaa toteutetuista luokista rajapinnan WADL-kuvaus.

5.2 XML-dokumentin tuottaminen Agilefantin tietosisällöstä

Kun palvelurajapintaan lähetetään palvelupyyntö, rajapinnan toteuttava luokka välittää kutsun rajapinnan varsinaiselle toimintalogiikalle, joka koostaa pyyntöä vastaavan tietosisällön XML-dokumenttiin hyödyntämällä Agilefantin liiketoimintalogiikkaa. Dokumenttien muodostamisessa hyödynnettiin avoimen lähdekoodin Dom4j-kirjastoa. Dom4j on XML-dokumenttien käsittelyyn kehitetty Java-kirjasto.
Agilefant-toteutus muodostuu useasta - eri tehtävää palvelemaan toteutetusta - ohjelmakerroksesta. Sovelluksen liiketoimintalogiikka (engl. Business Logic Layer) tarjoaa kattavan ohjelmointirajapinnan sovelluksen tietosisällön turvalliseen käsittelyyn.

Kommunikointi liiketoimintalogiikan kautta on esimerkiksi suoraan tietokantaan kohdistuvien kyselyiden sijaan suositeltavaa, koska sovelluslogiikan Hibernatella ja Spring DAO (Data Access Object):lla toteutettu relaatiotietokannan käsittely takaa transaktioiden toteutumisen. Toinen merkittävä hyöty on DAO-luokkien mahdollistama viite-eheyksien toteutuminen.

Agilefantin liiketoimintalogiikka sisältää kattavat palvelut kaikkien sen sisältämien, yksittäisen ohjelmistoprojektin kannalta olennaisten tietokokonaisuuksien muodostamiseksi. Palveluiden avulla ReST-rajapinnassa voidaan muodostaa halutun kaltainen XML-dokumentti. Koodiesimerkissä 6 (sivulla 26) on kuvattu rajapinnan sovelluslogiikan metodi, joka koostaa Agilefantin sisältämät projektit XML-dokumenttiin. Dokumentin rakentaminen sisältää seuraavat vaiheet (merkitty koodiesimerkkiin):

1. Luodaan Document-olio,
2. Luodaan juurielementti ja lisätään se dokumenttiin,
3. Hyödynnetään Agilefantin liiketoimintalogiikkan palveluita halutun tiedon lisäämiseksi dokumenttiin omana elementtinä (esim. projektin id-numero).

Dokumentin rakenne ja tietosisältö voidaan vapaasti määrittää hyödyntämällä elementtejä sekä attribuutteja. Ainoastaan dokumentti ja dokumentin juurielementti ovat pakollisia objekteja. Koodiesimerkissä 7 (sivulla 27) on esitetty esimerkkifunktiossa muodostettu XML-dokumentti, joka sisältää Agilefantiin luodut projektit.

```
public String getProjects() {
  1. Document document = DocumentHelper.createDocument();
  2. Element root = document.addElement("AgilefantResult");
     Element projects = root.addElement("Projects");
     Collection<Project> AFProjects =
projectBusiness.retrieveAll();
     Iterator<Project> projectIterator =
AFProjects.iterator();
     while(projectIterator.hasNext()) {
      Project AFProject = projectIterator.next();
   3. Element projectId = projects.addElement("ProjectId");
      projectId.addText(Integer.valueOf(AFProject.getId()).toS
tring());
      Element projectName =
projects.addElement("ProjectName");
      projectName.addText(AFProject.getName());
      Element description =
projects.addElement("Description");
      description.addText(AFProject.getDescription());
    }
   return document.asXML();
  }
```

Koodiesimerkki 6. Sovelluslogiikan yksittäinen metodi, jossa työstetään XML-dokumentti Agilefantin projekteista.

```
<AgilefantResult>
    <Projects>
          <ProjectId>2</ProjectId>
          <ProjectName>Agilefant-testi</ProjectName>
          <Description>Viestirajapinta</Description>
          <ProjectId>6</ProjectId>
          <ProjectName>KuPSi</ProjectName>
          <Description/>
          <ProjectId>11</ProjectId>
          <ProjectName>Testiprojekti</ProjectName>
          <Description/>
      </Projects>
</AgilefantResult>
```

Koodiesimerkki 7. Agilefant-projektit sisältävä XML-dokumentti.

6 Viestijärjestelmäsovelluksen (engl. Mediator) toteutus

Toiminnallisen kokonaisuuden ja järjestelmäintegraation jatkokehitettävyyden kannalta merkittävin osio oli erillisen viestijärjestelmäohjelmiston toteuttaminen. Niin sanottu mediaattori toteutettiin Java EE -standardin mukaisilla Java-servleteillä. Sovelluksen toimintalogiikan toteuttamisessa hyödynnettiin avoimen lähdekoodin Apache Camel -integraatiosovelluskehystä.

Apache Camel tarjoaa kattavan sovelluskehyksen järjestelmäintegraatiototeutuksille. Camelin helppo käytettävyys perustuu yleisimmistä tiedonsiirtoprotokollista ja dataformaateista tehtyihin korkean tason abstraktioihin. Camel sisältää yli 70 komponenttia, joiden avulla se suoriutuu erittäin kattavasti eri tiedonsiirtoprotokollia ja dataformaatteja käsittävistä viestinvälitystapahtumista.

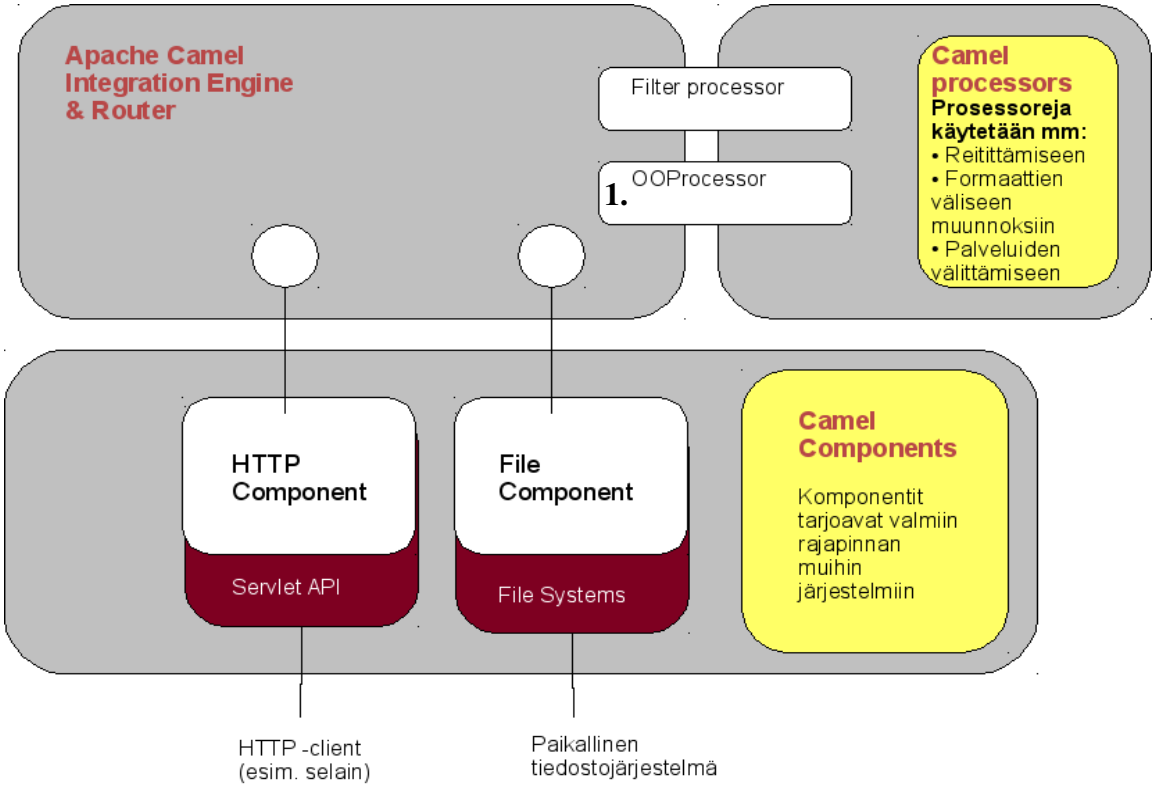
6.1 Apache Camelin hyödyntäminen viestijärjestelmäsovelluksessa

Apache Camel tarjoaa valmiin ja helppokäyttöisen sovelluskehyksen järjestelmien integrointiin. Camelin ydintoiminnallisuus muodostuu reititys- / välityskomponentista, joka toteuttaa viestien välittämistä laaditun konfiguraation mukaisesti. Reitittämisen lisäksi integraatioon saadaan lisätoiminnallisuutta erillisillä prosessoreilla, joilla voidaan esimerkiksi suorittaa dataformaattien välisiä muunnoksia.

Apache Camelin reititystä ja välitystä toteuttavan ydinkomponentin toiminta on helppo käsittää, kun tutustuu sovelluskehyksen keskeisiin komponentteihin ja käsitteisiin, jotka on esitetty seuraavalla sivulla kuvassa 8:

- Camel Components (suom. komponentti) – Komponenttien avulla luodaan halutunlaisia päätepisteitä (engl. Endpoint).
- Camel Endpoints – Päätepiste kuvaa yksittäisen viestin lähtö- tai päätepistettä.
- Camel Processors (suom. prosessori) – Prosessoreiden avulla suoritetaan viestittämisessä väliprosessointia.

Prosessori- ja komponenttiluokista periyttämällä voidaan Cameliin helposti toteuttaa myös omia komponentteja. Kuvaan 8 on piirretty integraatiossa toteutettu erillinen prosessori (**1.**), jolla suoritetaan edellä mainittujen käsitemallien yhteensovittaminen. Toteutetut komponentit täytyy muistaa lisätä sovelluksen integraatiokontekstiin joko DSL- tai Spring XML -konfiguraatiossa. Apache Camelista löytyy omat komponenttinsa molemmille integraatiossa käytetyille tiedonsiirtoprotokollille (HTTP ja SOAP), joita voitiin hyödyntää reitityskonfiguraation toteuttamisessa.



Kuva 8. Apache Camel -integraatiosovelluskehyksen arkkitehtuuri.

- **Java:**

```
from("file:data/inbox").to("file:data/outbox");
```

- **Spring XML:**

```
<route>
      <from uri="file:data/inbox" />
      <to uri="file:data/outbox" />
</route>
```

- **Scala:**

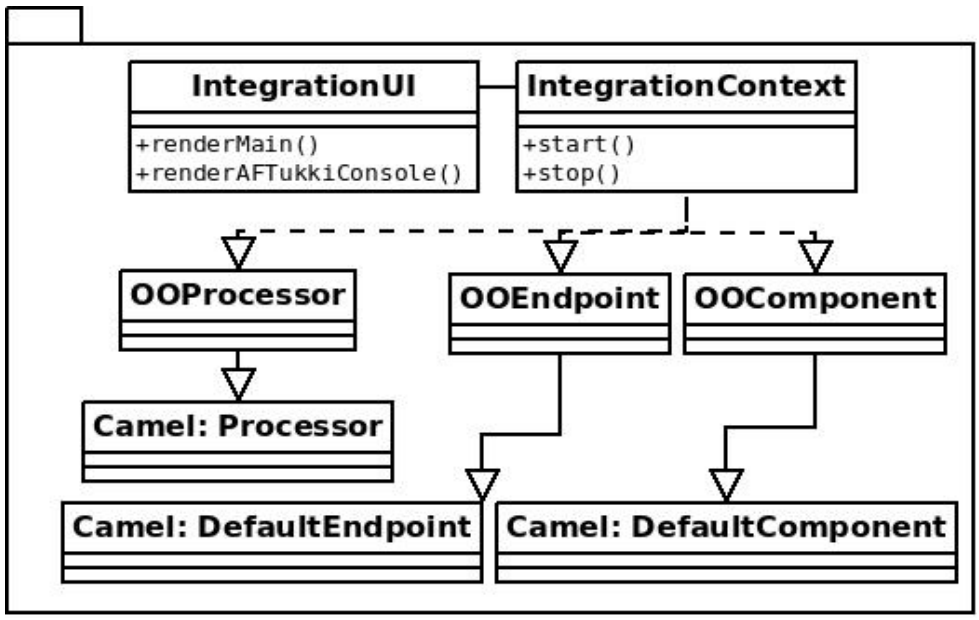
```
from "file:data/inbox" → "file:data/outbox"
```

Koodiesimerkki 8. Esimerkit Camelin DSL-kielen Java-, Spring XML - ja Scala-konfiguraatioista.

Camelin ominaisuuksia konfiguroidaan sen omalla DSL (Domain Specific Language) -kielellä, josta löytyy tuki Java- ja Scala-ohjelmointikielille sekä Springin XML -konfiguraatiolle. Springin avulla reitityskonfiguraatio voidaan helposti alustaa halutuilla toiminnoilla ja parametreilla. Lisäksi DSL:n avulla voidaan myös dynaamisesti muokata reitityskonfiguraatiota. Konfigurointimenetelmät on esitelty edellisen sivun koodiesimerkissä 8, jossa reititetään inbox-kansion sisältö outbox-kansioon.

6.2 Viestijärjestelmäsovellus

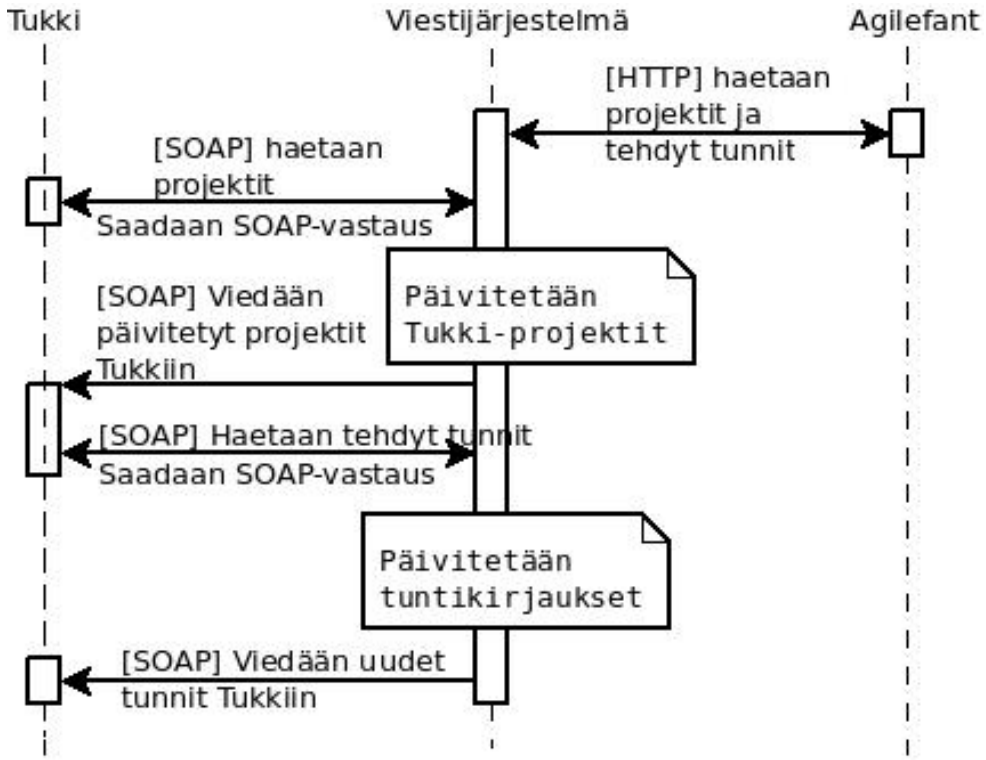
Camelin toiminnallisen konseptin mukaisesti toteutetun viestijärjestelmäsovelluksen arkkitehtuuri muodostui selkeäksi ja modulaariseksi. Integraatiosovelluskehyksen tarjoama fyysinen ja toiminnallinen kokonaisuus helpottaa myös viestijärjestelmän jatkokehitystä. Kuvassa 9 on esitetty toteutetun sovelluksen rakenne luokkakaavioesityksenä. Sovellus muodostuu kahdesta ylätason Java-servlet -luokasta, joista toinen hallinnoi reititys- ja integraatiokokonaisuutta ja toinen toteuttaa html-käyttöliittymän. Agilefant-Tukki -integraatiota varten toteutettiin lisäksi erillinen komponentti (”OOComponent”), päätepiste (”OOEndpoint”) sekä prosessoriluokka (”OOProcessor”), joihin voitiin ja voidaan myös jatkossa luontevasti kehittää kyseisen integraation tarvitsemia lisäominaisuuksia.



Kuva 9. Viestijärjestelmän luokkakaavio.

Yksittäinen integraatiossa toteutettava tuntikirjausten vienti järjestelmästä toiseen on monivaiheinen prosessi. Seuraavassa sekvenssikaaviossa (kuva 10) on esitetty prosessin vaiheet yksityiskohtaisesti:

1. Viestijärjestelmä hakee Agilefantista kaikki projektit, tehtävät ja niille kuluvan vuorokauden aikana tehdyt tuntikirjaukset.
2. Viestijärjestelmä hakee Tukista kaikki projektit ja tehtävät.
3. Tukin projektit ja tehtävät päivitetään, jos Agilefantin projekteihin on tehty muutoksia tai lisäyksiä.
4. Päivitetyt projektit ja tehtävät viedään Tukkiin.
5. Viestijärjestelmä hakee Tukista tehdyt tuntikirjaukset.
6. Tukin tuntitapahtumat päivitetään, sekä tehdään mahdolliset lisäykset ja poistot.
7. Päivitetyt tuntitapahtumat viedään Tukkiin.



Kuva 10. Sekvenssikaavioesitys tuntitapahtumien siirtoon tarvittavista operaatioista.

Camelin DSL-konfiguraatiossa voidaan yksittäisiin komponentteihin ja prosessoreihin viitata suoraan URI:lla. Lisäksi päätepisteitä voidaan yhdistellä hyvin joustavasti, jolloin voidaan vähäisellä koodimäärällä toteuttaa hyvinkin vaativia viestinvälitysrutiineita. Edellisessä kappaleessa kuvattu ja integraatiossa toteutettu 7-osainen reitys- ja viestinvälitysoperaatio voitiin käytännössä ohjelmoida kolmella ohjelmarivillä koodiesimerkin 9 mukaisella tavalla.

Koodiesimerkistä ilmenevän helppokäyttöisen syntaksin lisäksi kannattaa kiinnittää huomioida käytettäviin päätepisteisiin (esim. ”AgilefantContent”, **1.**), jotka on toteutettu erillisellä integraatiota varten toteutetulla päätepisteluokalla (”OOEndpoint”) sekä erilliseen prosessoriin (”OOProcessor”, **2.**), jonka avulla suoritetaan tiedon yhteensovittaminen. Prosessori-luokan prosessointi perustuu täysin standardin java.util-pakkauksen työkaluilla suoritettaviin merkkijonojen vertailu- ja muokkausoperaatioihin.

```
public class OOIntegrationContext extends HttpServlet {

CamelContext context = new DefaultCamelContext();
private OOEndpoint AgilefantContent;
private OOEndpoint TukkiGetProjects;
private OOEndpoint TukkiSetProjects;
private OOEndpoint TukkiGetHourEntries;
private OOEndpoint TukkiSetHourEntries;
private OOProcessor OOProcessor = null;
String AgilefantContent = null;
String TukkiProjects = null;
String TukkiHourEntries =  null;

/* Reititysketju toistetaan konfiguraation mukaisesti kerran
vuorokaudessa */

/* Haetaan Agilefantin ReST-rajapinnasta projektit ja
tuntitapahtumat merkkijonoon */                          1.
from(”timer:daily?”fixedRate=true&period=86400000”).to(Agilefa
ntContent).to(AgilefantContent);

/** Haetaan Tukin projektit merkkijonoon, suoritetaan
prosessorilla yhteensovitus ja viedään päivitetyt projektit
Tukkiin */                                          2.
from(TukkiGetProjects).to(TukkiProjects).process(OOProcessor).
to(TukkiSetProjects);

/** Haetaan Tukin tuntitapahtumat merkkijonoon, suoritetaan
prosessorilla yhteensovitus ja viedään päivitetyt
tuntitapahtumat Tukkiin */
from(TukkiGetHourEntries).to(TukkiHourEntries).process(OOProce
ssor).to(TukkiSetHourEntries);
}
```

Koodiesimerkki 9. Camel DSL:n avulla toteutettu integraation reitityskonfiguraatio.

7 Yhteenveto

Suurimmat haasteet työn toteuttamisen osalta liittyivät ennakkoon osaltani melko tuntemattomaan Java EE -ohjelmistoalustaan. Modernit Java EE -websovellukset koostuvat useaa eri tarkoitusta varten laaditusta ohjelmakerroksesta. Kokonaisuus on usein hyvin monimutkainen ja kehittäjän kannalta jo pienenkin lisäominaisuuden toteuttaminen vaatii käytettyjen tekniikoiden vahvaa perusosaamista.

Työssä saavutetun Java EE -osaamisen lisäksi työn mielenkiintoisinta antia oli valitun Messaging-integraatiomallin ja Apache Camelin avulla toteutettu kevyt ja joustava järjestelmäintegraatio. Camel ja integraatioita varten laaditut ratkaisumallit tarjoavat yhdessä erittäin kattavan työkalupakin haastaviinkin järjestelmäintegraatioihin. Vaihtoehtoinen ratkaisu olisi esimerkiksi ollut valita jokin massiivinen valmistyökalu. Ennalta on hyvin vaikea arvioida, mikä ratkaisu osoittautuu pitkällä tähtäimellä parhaaksi.

Toteutettiinpa integraatio tavalla tai toisella, on joka tapauksessa tärkeää pyrkiä huolehtimaan ratkaisun jatkokehitettävyydestä, jolla taas voidaan suoraan vaikuttaa toteutetun ratkaisun elinkaaren pituuteen. Mitä kauemmin integraatiototeutus soveltuu tietojärjestelmäkokonaisuuden hallintaan ja kehittämiseen, sitä paremmin katetaan myös siihen investoidut kokonaiskustannukset.

Työn aihe oli erittäin ajankohtainen ja mielenkiintoinen. Tietojärjestelmiin kohdistuvat muutospaineet eivät suinkaan vähene ajan saatossa, jolloin saavutetuista taidoista ja tiedosta on varmasti hyötyä tulevaisuudessa.

8 Lähdeluettelo

Painetut lähteet

1. Hohpe, Gregor & Woolf Bobby, Addison Wesley 2009, Enterprise Integration Patterns
2. Schwaber, Ken, Beedle, Mike, Prentice Hall 2002, Agile Software Development with Scrum
3. Ibsen, Claus, Anstey, Jonathan, Zbarcea, Hadrian, Manning 2009, Camel In Action (Early Access Edition)

Sähköiset lähteet

1. Roy Thomas Fielding, Dissertation 2000. Architechtural Styles and the Design of Network-based Software Arhictechtures. Saatavissa: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
2. W3C, 2009. WADL Specification. Saatavissa: http://www.w3.org/Submission/wadl/