Web service -palvelurajapinnan arkkitehtuurisuunnittelu toiminnanohjausjärjestelmään

Pekka Rantanen

Opinnäytetyö
Huhtikuu 2012
Tietojärjestelmäosaamisen koulutusohjelma, ylempi AMK
Tampereen ammattikorkeakoulu

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojärjestelmäosaamisen koulutusohjelma, ylempi AMK

RANTANEN, PEKKA: Web service -palvelurajapinnan arkkitehtuurisuunnittelu toiminnanohjausjärjestelmään

Opinnäytetyö 101 sivua
Huhtikuu 2012

Yritysjärjestelmien pääasiallinen tarkoitus on tukea yrityksen liiketoimintaa mahdollisimman tehokkaasti. Yrityksen toiminnot ja prosessit vaikuttavat toisiinsa suoraan tai välillisesti, joten myös niitä tukemaan käytettävien järjestelmien tulee pystyä toimimaan vuorovaikutuksessa toistensa kanssa. Järjestelmien liitettävyys osaksi isompaa liiketoimintaa tukevaa tietojärjestelmäkokonaisuutta on yrityksen kilpailukykyyn merkittävästi vaikuttava tekijä.

Tämän opinnäytetyön lähtökohta oli Digia Oyj:n oma Enterprise-toiminnanohjausjärjestelmä ja siihen liittyvä tarve parantaa sovelluksen liitettävyyttä ulkoisiin järjestelmiin ja päätelaitteisiin. Liitettävyyden parantaminen tarkoittaa käytännössä toiminnanohjausjärjestelmään asiakasympäristöissä liittyvien integraatioiden helpottamista sekä sovellusteknisesti että laite- ja henkilöresursointien osalta. Lisäksi tavoitteena oli työskentelyn tehostaminen teknisestä ratkaisusta seuraavan toimintatapojen muutoksen avulla ja potentiaalisten projektiresurssien määrän kasvattaminen yleisellä tasolla.

Opinnäytetyöhön liittynyt kehittämisprojekti aloitettiin loppuvuodesta 2010 tehdyllä toteutettavuusdemolla, jolla konkretisoitiin mahdollisuus rajapinnan toteuttamiseen. Varsinainen kehitysprojekti alkoi vaatimusmäärittelyllä, jossa asetettiin hankkeelle konkreettiset tavoitteet ja rajat. Tavoiteasetannan perusteella tutkittiin mahdollisia ratkaisumalleja kirjallisuuskartoituksella haetun teoriatiedon pohjalta. Teoriatiedon sekä aiemmin integraatioista kertyneen kokemuksen perusteella tehtiin rajapinnan arkkitehtuurisuunnitelma. Arvioidun ja hyväksytyn arkkitehtuurisuunnitelman perusteella toteutettiin Web service -standardin mukainen integraatiorajapinta Digia Enterprise -toiminnanohjausjärjestelmään.

Opinnäytetyönä suunniteltua ja toteutettua rajapintaa on jo käytetty onnistuneesti osana muita kehityshankkeita mahdollistamaan toiminnanohjauksellisen tiedon ja liiketoimintalogiikan hyödyntämisen teknologia riippumattomasti osana uudenlaisia Digia Enterprise -toiminnanohjausjärjestelmään liittyviä konsepteja. Lyhyellä aikavälillä ja teknisesti arvioiden opinnäytetyön lopputulos on onnistunut.

Todellisen onnistumisen arviointi vaatii toteutetun teknisen ratkaisun hyödyntämistä useissa asiakasprojekteissa ilman alkuperäisten toteuttajien panosta. Tällöin voidaan luotettavasti arvioida rajapinnan hyödyllisyyttä kaupallisessa mielessä sekä jalkauttamisen onnistumista käytettyjen resurssien perusteella.

Asiasanat: web service, toiminnanohjaus, progress openedge, sovellusarkkitehtuuri, arkkitehtuurisuunnittelu, toiminnalliset vaatimukset, ei-toiminnalliset vaatimukset

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Information System Competence, Master's Degree

RANTANEN, PEKKA: Web Service Architecture Design for ERP System

Master's thesis 101 pages
April 2012

The main purpose of enterprise systems is to support a company's business as efficiently as possible. The company's business processes and functions interact with each other directly or indirectly. This interaction is required also from the various systems used to support business. The ability to connect a system to a larger information system as an active player has a significant impact on a company's competitiveness.

The basis for this Master's thesis is Digia Plc's own Enterprise Resource Planning (ERP) system and the need to improve the application's connectivity with external systems and devices. Improving connectivity basically means facilitating different customer-related integration needs by providing technical solutions as well as solutions to different hardware and personnel resourcing challenges. Additional objects of the work were to improve efficiency when working with integrations and to increase the number of potential project resources in general.

The development project associated with the Master's thesis was originally initiated in late 2010 by designing and implementing a proof-of-concept. The actual development project began with requirements specification, which set up the tangible targets and boundaries for the work. Possible solution models were studied by collecting theoretical information through literature analysis. Architectural design was created on the basis of theoretical information and knowledge gained from previous integration cases. On the basis of the architectural plan a functioning web service standards compliant interface was implemented for the ERP system.

The designed and implemented web service interface has been used successfully in other development projects to share ERP system data and business logic. These projects have created new technologically independent concepts and functional applications to use with Digia's Enterprise ERP system. The outcome of the thesis work has been successful when evaluating over the short term and based on technical facts only.

The real assessment of the thesis outcome needs at least several customer implementations done without the aid of the original designers. After that, a more reliable assessment of commercial usefulness and the success of deployment can be done.

Key words: web service, enterprise resource planning, progress openedge, software architecture, software architecture design, functional requirements, non-functional requirements

SISÄLLYS

1 JOHDANTO

Nykyisessä tietoyhteiskunnassa ulkopuolisista yhteyksistä eristettyjen järjestelmien käyttö ei ole enää liiketoiminnallisesti kannattavaa tai järkevää. Yrityksen liiketoiminnan tehokkuus riippuu siitä, kuinka hyvin se on pystynyt hiomaan omaa toimintaansa vastaamaan asiakkaiden vaatimuksiin. Yrityksen tietojärjestelmien tehtävä on tukea liiketoimintaa.

Samalla tavalla, kuin yrityksen toiminnot ja prosessit vuorovaikuttavat suoraan tai välillisesti toistensa kanssa pitäisi myös tietojärjestelmien pystyä kommunikoimaan keskenään ja vaikuttamaan toisiinsa. Tämän pitäisi olla mahdollista niin yrityksen sisäisiin kuin ulkoisiinkin prosesseihin ja toimintoihin liittyvien tietojärjestelmien osalta.

Voidaan sanoa, että yrityksen järjestelmien liitettävyys osaksi isompaa tietojärjestelmäkokonaisuutta on kilpailukykyyn merkittävästi vaikuttava tekijä. Järjestelmä, joka ei voi vastaanottaa tietoa ulkopuolelta tai tuottaa tietoa ulkopuolisten järjestelmien käyttöön, ei nykyajattelun mukaan voi tehokkaasti osallistua yrityksen liiketoimintaprosesseihin. Yrityksillä on kuitenkin vielä käytössään paljon tällaisia toisiinsa liittämättömiä järjestelmiä, joiden tehokasta käyttöikää olisi muuten vielä runsaasti jäljellä. Lisäongelmia tilanteeseen tuo vielä se, että järjestelmähankintoja tehtäessä ei välttämättä koskaan ole otettu liitettävyyttä huomioon. Tällöin yrityksen järjestelmät voivat olla jo teknisellä tasolla keskenään yhteensopimattomia.

Tätä tilannetta helpottamaan on olemassa palvelukeskeiseen arkkitehtuuriin perustuva ja suurien tietojärjestelmätoimijoiden hyväksymä teknologiastandardien joukko. Tätä teknologiastandardien joukkoa voidaan kutsua yhdellä termillä web service -standardiksi. Termillä tarkoitetaan korkean tason menetelmää, jossa on sovittu yleisistä toimintatavoista, eikä se ota kantaa yksityiskohtiin. Web service -palvelut mahdollistavat tietojärjestelmäintegraatioiden ja hajautettujen järjestelmien toteuttamisen myös teknisesti keskenään sopimattomien järjestelmien välillä.

Tämä opinnäytetyö käsittelee Digian Enterprise -toiminnanohjausjärjestelmän liitettävyyden parantamista. Parannettu liitettävyys syntyy tuottamalla toiminnanohjausjärjestelmään palvelukeskeisen arkkitehtuurin pääperiaatteet täyttävä web service –standardeja noudattava rajapinta. Käytännössä kyseessä on fyysinen rajapinta, jota hyödyntäen toiminnanohjausjärjestelmää käyttävät asiakkaat voivat liittää muita järjestelmiä osaksi toiminnanohjausta sekä tuottaa toiminnanohjausjärjestelmästä jalostettua tietoa ulkopuolisin järjestelmiin, kuten vaikkapa raportointijärjestelmiin.

Opinnäytetyön lopputuloksena on kattava arkkitehtuurikuvaus kehittämistehtävänä toteutettavasta fyysisestä rajapinnasta. Toiminnanohjausjärjestelmän standardoidun rajapintaa hyödyntäen asiakkaiden on helpompi liittää tietojärjestelmiä toisiinsa ja näin mukautua paremmin liiketoiminnan vaihtuviin vaatimuksiin ja tarpeisiin.

1.1 Taustaa

Digia Enterprise -toiminnanohjausjärjestelmä sisältää yli 15 vuoden kehityksen tuloksena valtavan määrän liiketoimintalogiikkaa. Standardia ja natiivia tapaa tämän logiikan hyödyntämiseksi ulkopuolisissa järjestelmissä tai päätelaitteissa ei ole. Enterprise-toiminnanohjausjärjestelmän liittäminen osaksi suurempaa tietojärjestelmäkokonaisuutta on aiemmin ollut asiakas- ja tapauskohtaisten ratkaisujen sekä tietyn Digian toimittaman integraatioteknologian varassa.

Toimivia malleja toiminnanohjausjärjestelmän hyödyntämiseen ulkopuolisia välineitä käyttäen on siis kuitenkin olemassa. Tästä huolimatta asiakasprojekteissa tehdään erilaisia ja eritasoisia ratkaisuja toiminnanohjauksellisen tiedon ja liiketoimintalogiikan hyödyntämiseksi varsinaisen toiminnanohjausjärjestelmän ulkopuolella.

Asiakasprojekteissa tehtävät ratkaisut eivät välttämättä ole laadullisesti riittäviä, ne eivät myöskään skaalaudu tai mukaudu asiakkaiden liiketoiminnantarpeiden

mukaisesti. Yksinkertaisesti sanottuna osa tehdyistä ratkaisuista ei ole liiketoiminnan kannalta tarkasteltuna ole kestäviä.

Projektien resursointi aiheuttaa myös ongelmia tilanteissa, joissa järjestelmien välisiä integraatioita pitäisi toteuttaa järjestelmän ulkopuolisilla välineillä. Käytettävissä olevia integraatiovälineiden osaajia on rajallinen määrä. Sellaisia integraatiovälineiden osaajia, joiden osaamiseen sisältyy myös toiminnanohjauksellista osaamista yleisesti tai nimenomaisesti osaamista tietystä toiminnanohjausjärjestelmästä, on käytettävissä hyvin vähän.

1.2 Tavoitteet ja rajaukset

Työn tavoitteena on kehittää toimeksiantajan toiminnanohjausjärjestelmässä olevan tiedon ja liiketoimintalogiikan käyttömahdollisuuksia järjestelmän ulkopuolella. Toisin sanoen tavoitteena on parantaa toiminnanohjausjärjestelmän liitettävyyttä asiakkaan tietojärjestelmäympäristöissä teknisen ratkaisun avulla. Konkreettisesti liitettävyyden parantuminen tarkoittaa:

- asiakasprojekteissa toteutettavien järjestelmien välisten integraatioiden helpottumista
- toiminnanohjauksellisen tiedon ja liiketoimintalogiikan hyödyntämisen helpottumista järjestelmän ulkopuolella
- potentiaalisten projektiresurssien määrään kasvattamista poistamalla teknisiä rajoitteita
- projekteissa integraatiotehtävissä työskentelevien henkilöiden ajankäytön tehostamista tuottamalla parempia työskentelytapoja ja välineitä heidän käyttöönsä

Tarkoituksena on suunnitella ja toteuttaa toiminnanohjausjärjestelmään fyysinen rajapinta ohjeistoineen ja malleineen asetettujen tavoitteiden saavuttamiseksi. Tuotettavaa ratkaisua tullaan käyttämään asiakasprojekteissa järjestelmän ulkoisten integraatiotarpeiden ratkaisemiseksi. Käytännössä tarkoituksena on tehdä arkkitehtuurisuunnitelma, jonka pohjalta toteutetaan Progress OpenEdge -teknologialla standardi ja dokumentoitu web service –rajapinta. Toteutuksen

todentamiseksi tullaan toteuttamaan myös muutamia mallipalveluita sekä kaksi erillistä eri teknologioilla toimivaa asiakassovellusta.

Opinnäytetyö on rajattu koskemaan nimenomaan tavoitteena olevan rajapinnan arkkitehtuurisuunnittelua. Opinnäytetyö ei sisällä kuvausta varsinaisesta rajapinnan toteutuksesta. Lopputulosta arvioidaan kuitenkin kokonaisuutena, jossa on huomioitu arkkitehtuurin lisäksi rajapinnan fyysinen toteutus sekä projektin aikana toteutetut asiakassovellukset.

1.3 Tutkimusmenetelmistä

Opinnäytetyö painottuu laadulliseen tutkimukseen. Laadullisessa tutkimuksessa ei välttämättä aseteta tarkkaa tutkimusongelmaa etukäteen. Tästä huolimatta olen kuitenkin muotoillut tätä työtä koskevan tutkimusongelman seuraavaan lauseeseen:

”Kuinka Digia Enterprise toiminnanohjausjärjestelmään toteutetaan standardeja noudattava, tietoturvallinen ja natiivi integraatiorajapinta kustannustehokkaan järjestelmä- ja päätelaiteintegraation saavuttamiseksi?”

Tutkimusongelmaa ei ole asetettu liian tarkalla tasolla, jotta se ei rajoittaisi varsinaista ratkaisua tai ratkaisun löytämiseksi käytettäviä metodeja. Se on kuitenkin riittävän tarkka ja yksiselitteinen, jotta sekä kehittämistehtävän että opinnäytetyön sisältö pysyy määrätyissä rajoissa.

Teoriatietoa tutkittavasta aihealueesta, varsinaisesta ongelmasta sekä ratkaisusta haettiin kirjallisuuskartoituksella. Kirjallisuuskartoituksen avulla saatiin hankittua tietoperusta varsinaiselle tutkimukselle ja samalla muodostettiin myös pohja käytännön ratkaisulle.

Opinnäytetyön tutkimusote on pääasiassa konstruktiivinen. Lukan (2001) mukaan konstruktiivisen tutkimuksen lähtökohta on olennaisen käytännön ongelman olemassaolo ja lopputuloksena ongelman ratkaisevan uuden konstruktion

luominen. Opinnäytetyössä keskitytään nimenomaan olennaisen käytännön ongelman tutkimiseen ja konkreettisen ratkaisun kehittämiseen. Ratkaistava käytännön ongelma on esitetty tutkimusongelman muodossa. Ratkaisun muodostavia konstruktioita voivat olla esimerkiksi tuotteet, toimintamallit, tietojärjestelmät tai sovellukset (Lukka 2001). Opinnäytetyössä esiteltävä konkreettinen ratkaisu, eli uusi konstruktio, on web service –rajapinta osaksi Digia Enterprise toiminnanohjausjärjestelmää.

Lukan (2001) mukaan konstruktiivisen tutkimusotteen ideaalinen tulos on, että tosielämän ongelma ratkaistaan implementoidulla uudella konstruktiolla, ja tämä ongelmanratkaisu-prosessi tuottaa suuren kontribuution sekä käytännön että teorian näkökulmasta. Opinnäytetyön käytännön hyödyt ovat selkeät, mutta teoriatason hyödyt eivät niinkään. Periaatteessa opinnäytetyön kontribuutio teorian näkökulmasta on käytännön ratkaisun myötä lisääntyneen tiedon määrä. Uuden tiedon yleishyödyllisyys on kuitenkin rajallista, koska hyödyntäminen voi tapahtua ainoastaan työn toimeksiantajalla sisäisesti.

1.4 Toimeksiantaja

Digia Oyj on Suomesta lähtöisin oleva kansainvälistä liiketoimintaa harjoittava ohjelmistoratkaisu- ja palveluyhtiö. Yritys on perustettu vuonna 2005 Sysopen Oyj:n ja Digia Oy:n yhdistymisen myötä. Tämän jälkeen yritys on kasvanut usein yritysostoin yhdeksi Suomen suurimmista ohjelmistotaloista.

Yrityksen liiketoiminta perustuu yritysjärjestelmiin sekä mobiili- ja käyttäjäkokemuspalveluihin ja -ratkaisuihin. Yhtiön asiakkaita ovat eri toimialojen yritykset ja yhteisöt. Keskeisimmät toimialat ovat finanssi, julkishallinto, kauppa ja palvelut sekä telekommunikaatio. Toiminnanohjausliiketoimintaan Digialla on myös oma Digia Enterprise -toiminnanohjausjärjestelmä.

Digian strategiset painopistealueet, eli liiketoimintayksiköt, vuoden 2011 vuosikertomuksen mukaan ovat:

- asiakaskohtaiset ratkaisut ja palvelut (Solutions & Services –liiketoimintayksikkö)
- toimialakohtaiset monistettavat ohjelmistoratkaisut (Industry Verticals -liiketoimintayksikkö)
- kansainvälinen ohjelmistoliiketoiminta (International Products –liiketoimintayksikkö)
- uudet markkina-alueet (New Market Areas -liiketoimintayksikkö)

Digian missio vuoden 2011 vuosikertomuksen mukaan on

”Luomme oivaltavilla ratkaisuillamme onnistumisia ihmisten ja yhteisöjen jokaiseen päivään”.

Digialla on toimipisteitä Suomessa, Venäjällä, Kiinassa, Ruotsissa, Norjassa ja USA:ssa. Digian Vuoden 2011 (2012) vuosikertomuksen mukaan Digian liikevaihto oli n. 121 miljoonaa euroa ja henkilöstömäärä oli tilikaudella keskimäärin 1453 henkilöä.

2 TOIMINNANOHJAUS

Toiminnanohjaus (Enterprise Resource Planning, ERP) on käsitteenä laaja ja se koskee yleensä kaikkia yrityksen keskeisiä liiketoiminnan osa-alueita. Aihepiirin laajuudesta ja toisistaan poikkeavista liiketoimintatavoista johtuen käsitettä ei todennäköisesti ole määritelty yksiselitteisesti. Toiminnanohjausta voidaan kuitenkin avata liiketoiminnan osa-alueiden ja prosessien avulla.

Liiketoiminnan eroista riippumatta yrityksillä on yleensä samanlaisia toiminta-alueita. Näitä toiminnan osa-alueita ovat esimerkiksi myynti ja markkinointi, toimitusketjun hallinta, talousohjaus sekä henkilöstöhallinto. Perinteisesti näitä osa-alueita on käsitelty ja johdettu erillisinä sekä itsenäisinä kokonaisuuksina. Ne ovat kuitenkin toisistaan riippuvaisia ja ne toimivat vuorovaikutuksessa keskenään. (Monk & Wagner 2009, 2 – 3.)

Yrityksen liiketoimintaprosessit ovat kokoelma, tai ketju, toimintoja jotka syötteen saatuaan tuottavat sisäistä tai ulkoista asiakasta hyödyttävän lopputuloksen. Prosessit käsittävät yleensä toimintoja useista yrityksen toiminnan osa-alueista. Prosessien toimivuus riippuu niiden sisältämien toimintojen sisäisen tiedon käsittelyn sekä niiden keskinäisen vuorovaikutuksen tehokkuudesta. (Monk & Wagner 2009, 3 – 4.)

Monkin ja Wagnerin (2009, 1 – 5, 15) mukaan yrityksen toiminnan ohjaaminen tarkoittaa:

- yksittäisten toimintojen yhdistämistä prosesseiksi tehokkuuden ja kilpailukyvyn parantamiseksi
- muodostettujen prosessien, niiden syötteiden sekä tulosten seuraamista ja ohjaamista
- yksittäisten toimintojen analysointia ja niihin vaikuttamista

Yrityksen hallinta ja ohjaaminen vaatii tarkkaa ja ajan tasalla olevaa tietoa. Yrityksen toiminnanohjauksen tukemisessa tarvittavan tiedon keräämiseen, tuottamiseen ja analysointiin käytetään tietojärjestelmiä. Käytettävät tietojärjestel-

mät ovat yleensä tähän tarkoitukseen erityisesti tarkoitettuja toiminnanohjaussovelluksia (ERP-sovellus, ERP software).

2.1 Toiminnanohjaussovelluksista

Toiminnanohjaussovellus standardoi yrityksen tietoa ja liiketoimintaprosesseja. Ohjelmisto muuntaa ja kokoaa yrityksen tapahtumiin liittyvää tietoa yli organisaation sisäisten rajojen käyttökelpoiseksi informaatioksi mahdollistaen sekä tiedon että toiminnan keskitetyn analysoinnin. Näin koko yritykseen liittyvä tapahtumatason tieto saadaan hyödylliseen muotoon liiketoimintapäätösten tukemiseksi. (Parthasarathy 2007, 2; Magal & Word 2009, 15, 33 – 34.)

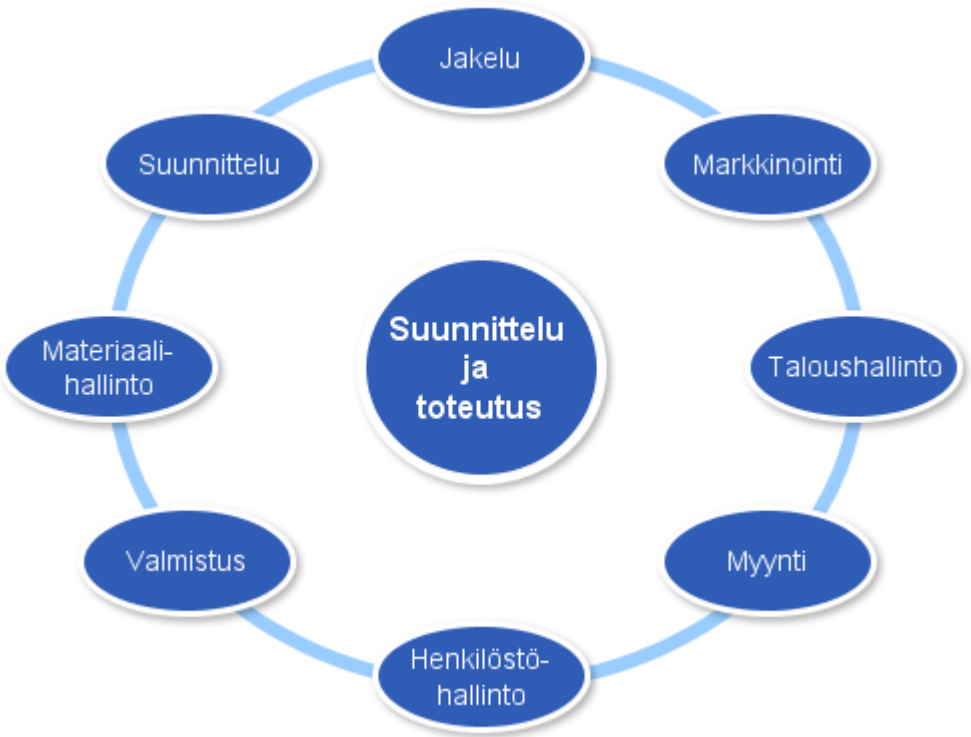
Toiminnanohjausjärjestelmän suorat hyödyt liittyvät parantuvaan tehokkuuteen, tiedon jalostamiseen päätöksenteon tukemiseksi sekä nopeampaan reagointiin sidosryhmätasolla. Välillisiä hyötyjä toiminnanohjausjärjestelmästä ovat esimerkiksi parantuva asiakastyytyväisyys, laatukustannusten pieneneminen, resurssien käytön tehostuminen, nopeampi ja parempi päätöksentekoprosessi. (Parthasarathy 2007, 3.)

Monkin ja Wagnerin (2009, 33 – 34) mukaan toiminnanohjausjärjestelmien hyödyntäminen voi vähentää yrityksen kuluja merkittävästi ja lisäksi tehostaa yrityksen kokonaistehokkuutta. Hyötyjä toiminnanohjausjärjestelmistä saadaan Monkin ja Wagnerin (2009, 33 – 34) mukaan kansainvälisten integraatioiden helpottumisella, tietojen ylläpidon keskittämisellä sekä toimien hallinnoimisen mahdollistamisella.

Toiminnanohjausjärjestelmien päätarkoituksena on kuitenkin jo tapahtuneiden asioiden tallentaminen. Koska toiminnan painopiste on menneessä eikä tulevaisuudessa, niin järjestelmät eivät juuri analysoi liiketoimintatilanteita tuottaakseen tai tarjotakseen tietoja tulevista toimintavaihtoehdoista. Toiminnanohjausjärjestelmät tarjoavat mahdollisuuden monimutkaisten ja hienostuneiden työnkulkujen ja prosessien toteuttamiseen, mutta ovat yleensä liian jäykkiä, jotta

näitä työnkulkuja ja prosesseja voitaisiin muokata tarveperusteisesti uusien liike-toimintamahdollisuuksien mukaisesti. (Parthasarathy 2007, 5.)

Parthasarathyn (2007, 6) mukaan toiminnanohjausjärjestelmä kokoaa yrityksen toiminnan kannalta tärkeät toiminnot yhteen kattavaan tietojärjestelmään tai tietokantaan tiedon keskitettyä keräämistä ja jakamista varten. Kuviossa 1 esitetään toiminnanohjausjärjestelmän yleisiä moduuleja Parthasarathya (2007, 6) mukaillen.



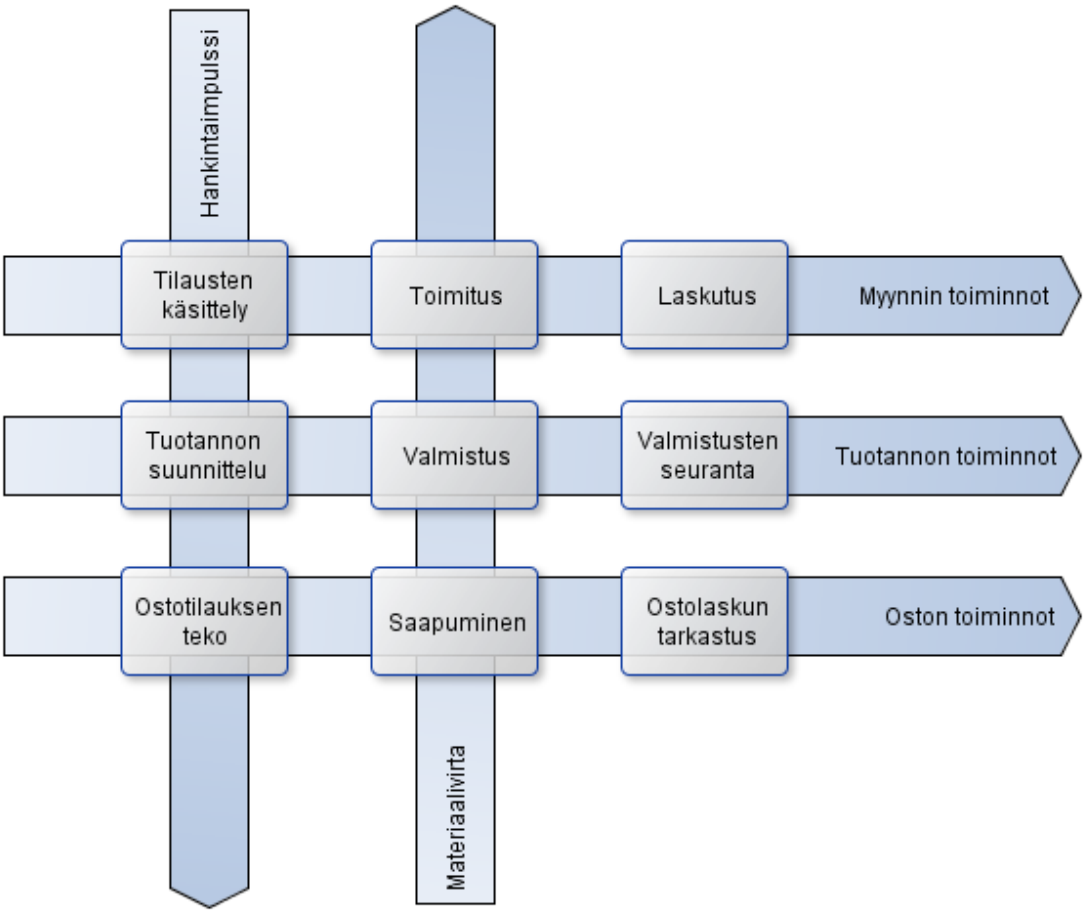
KUVIO 1. Toiminnanohjausjärjestelmän yleiset moduulit (mukaillen Parthasarathy, 2007, 6)

2.2 Digia Enterprise ERP

Digia Enterprise -toiminnanohjausjärjestelmä on tarkoitettu yrityksen päivittäisen operatiivisen toiminnan hoitamiseen. Vahvimmin tuettuina liiketoiminta-alueina ovat teollisuus ja tukkukauppa. Toiminnanohjausjärjestelmä skaalautuu yrityksen koon, maantieteellisen sijainnin sekä kielellisten tarpeiden mukaan. (Digia Enterprise 4.6 tuotekuvaus 2011, 15.)

Digia Enterprise toiminnanohjausjärjestelmä mahdollistaa yrityksen rutiinitöiden sujuvuuden, toiminnallisten tavoitteiden saavuttamista sekä parantaa yrityksen tiedonkulkua. Järjestelmä tarjoaa kattavat ominaisuudet ja toiminnot yrityksen toimintojen tukemiseksi. Lisäksi järjestelmä on suunniteltu tukemaan yrityksen muuttuvaa liiketoimintaympäristöä. (Digia Enterprise 4.6 tuotekuvaus 2011, 16 – 20.)

Kuviossa 2 kuvataan Enterprise ERP toiminnanohjausjärjestelmän pääprosessit laatikoina. Vaakasuuntaiset nuolet kuvaavat yrityksen toimintaa myynti-, osto- ja tuotantoprosessien kannalta. Pystysuuntaiset nuolet kuvaavat toimintoja, jotka vaikuttavat koko yrityksen toimintaan myynti-, tuotanto- ja osto-prosessien kautta. (Digia Enterprise 4.6 tuotekuvaus 2011, 12.)



KUVIO 2. Enterprise ERP toiminnanohjausjärjestelmän prosessit (mukaillen Digia Enterprise 4.6 tuotekuvaus 2011, 12)

Järjestelmänä Digia Enterprise -toiminnanohjausjärjestelmä kattaa yrityksen tärkeimmät toiminnot. Järjestelmään on myös mahdollista liittää uusia toimintoja ja ominaisuuksia tukemaan asiakkaan yksilöllisiä liiketoiminnan tarpeita. Enterprise-toiminnanohjausjärjestelmä on myös suunniteltu siten, että se voidaan sovittaa ohjaustapoja ja parametrointia muuttamalla erilaisiin liiketoimintaperiaatteisiin sopivaksi. (Digia Enterprise 4.6 tuotekuvaus 2011, 19 – 20.)

3 PROGRESS OPENEDGE -SOVELLUSKEHITYSALUSTA

Progress Software on USA:sta kotoisin oleva globaali ohjelmistoyritys, joka tarjoaa ohjelmistoratkaisuja hyvin laaja-alaisesti yritysten käyttöön sekä sovellus- ja ratkaisukehitykseen että teknologia alustaksi eri ratkaisuille. Yrityksen tuotteita ja teknologiaa käytetään yli 140 000 organisaatiossa ja yli 180 maassa. USA:n Fortune 100 listan yrityksistä 88 prosenttia käyttää Progress teknologiaan perustuvia sovelluksia ja järjestelmiä. (Progress Software, Who We Are.)

Progress Openedge on sovelluskehitysalusta dynaamisten, liiketoimintaprosesseihin perustuvien sovellusten kehittämiseen. OpenEdge-sovelluskehitysalusta mahdollistaa tietoturvallisen sovellusten käyttöönoton ja käytön yleisimmillä alustoilla ja päätelaitteilla. (Progress Software, OpenEdge 11.0.)

3.1 OpenEdge ABL

ABL (Advanced Business Language) on korkean tason proseduraalinen ohjelmointikieli. ABL-kielen avulla on mahdollista rakentaa kaikki sovelluksen tarvitsemat osa-alueet käyttöliittymästä liiketoimintalogiikkaan ja tietokantatapahtumiin asti. Itse sovelluksen rakentamisen lisäksi ABL-kielen mahdollistaa työkalujen ja apuvälineiden rakentamisen sovelluskehityksen avuksi ja kehitettyjen sovellusten ylläpitämiseen. (Openedge Getting Started: ABL Essentials 2009, 1-1.)

3.2 OpenEdge Appserver

AppServer on OpenEdge-sovelluskehitysalustan ydin, jonka avulla voidaan tukea monitasoarkkitehtuuria tuottamalla hajautettuja sovelluspalveluita. AppServer-teknologia mahdollistaa palveluiden tuottamisen sekä ABL-pohjaisille että muihin teknologioihin pohjautuville asiakkaille (client). AppServer on siis sovelluspalvelimen komponentti, joka suorittaa ABL-proseduureja vastauksena tieto-

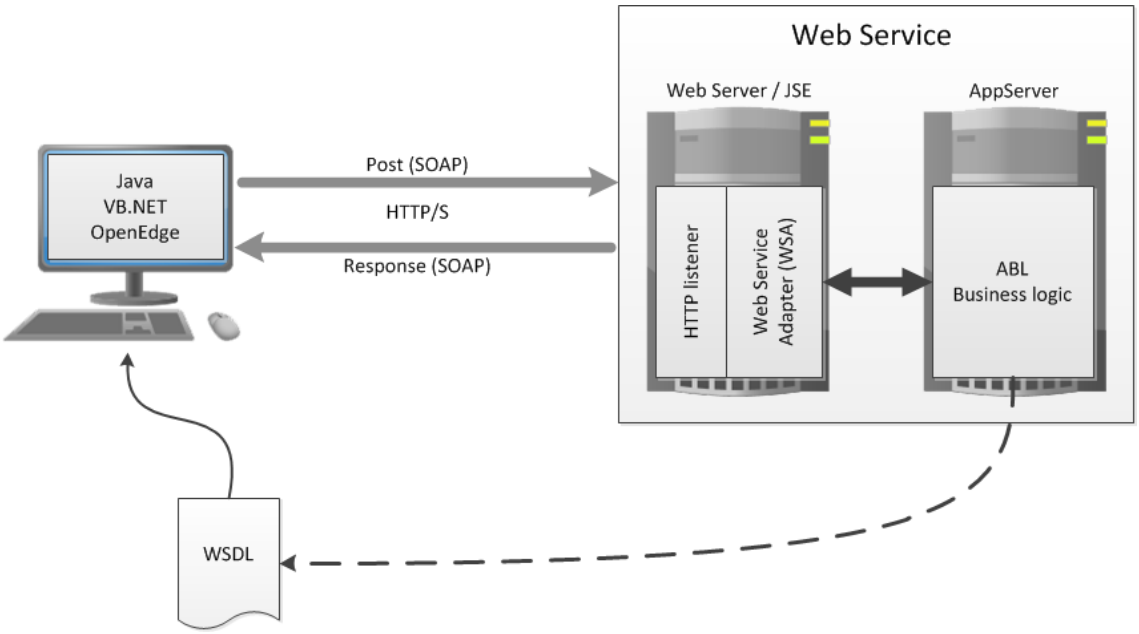
verkon ylitse tehtyihin palvelupyyntöihin. Suurin etu AppServer teknologian hyödyntämisestä on sovellusten osittaminen ja hajauttaminen. (Openedge Getting Started: Application and integration Services 2009, 2-4, 3-1, 3-2, 3-5.)

3.3 OpenEdge Webspeed

WebSpeed on sovellusalusta, joka sisältää sekä kehitysvälineet että tuotantokäyttöön tarkoitetun ajoympäristön. WebSpeed mahdollistaa yleisiä www-tekniikoita hyödyntävien sovellusten toteuttamisen ja ajamisen useille päätelaitteille ja alustoille. Webspeed-sovellusalustaa voidaan käyttää tuottamaan sisältöä ihmisille ja järjestelmille www-sivuina tai määrämuotoisia esitystapoja, kuten xml-formaattia, hyödyntäen. (Openedge Getting Started: Webspeed Essentials 2009, 1-1.)

3.4 OpenEdge Web Services

Progress Softwaren suosittelema tapa tuottaa palveluita OpenEdge-ympäristössä on toteuttaa palvelun logiikka ja toiminnot ABL-kielellä ja julkaista itse palvelu AppServer-alustalla. Mallilla voidaan tuottaa SOAP-pohjaisia palveluita sekä palveluiden hyödyntämiseen tarvittavat konekieliset kuvaukset rajapinnasta ja sen sisältämistä toiminnoista (WSDL-kuvaukset), toisin sanoen toteuttaa SOAP RPC -tyyppisen web service –rajapinnan palveluineen. Tässä mallissa edustapalvelimella sijaitseva Web Service Adapteri (WSA) mahdollistaa web service -palveluiden käyttäjän (client) yhteyden Appserver-pohjaiseen palveluun SOAP-standardin mukaista rajapintaa hyödyntäen. Tätä arkkitehtuuria kutsutaan OpenEdge Open Client -malliksi. Kuviossa 3 esitetään Progress OpenEdgen WSA-pohjainen web service -arkkitehtuuri (Openedge Getting Started: Application and integration Services 2009, 6-15.)



KUVIO 3. OpenEdge web services -arkkitehtuuri (Mukaillen Openedge Getting Started: Application and integration Services 2009, 6-15)

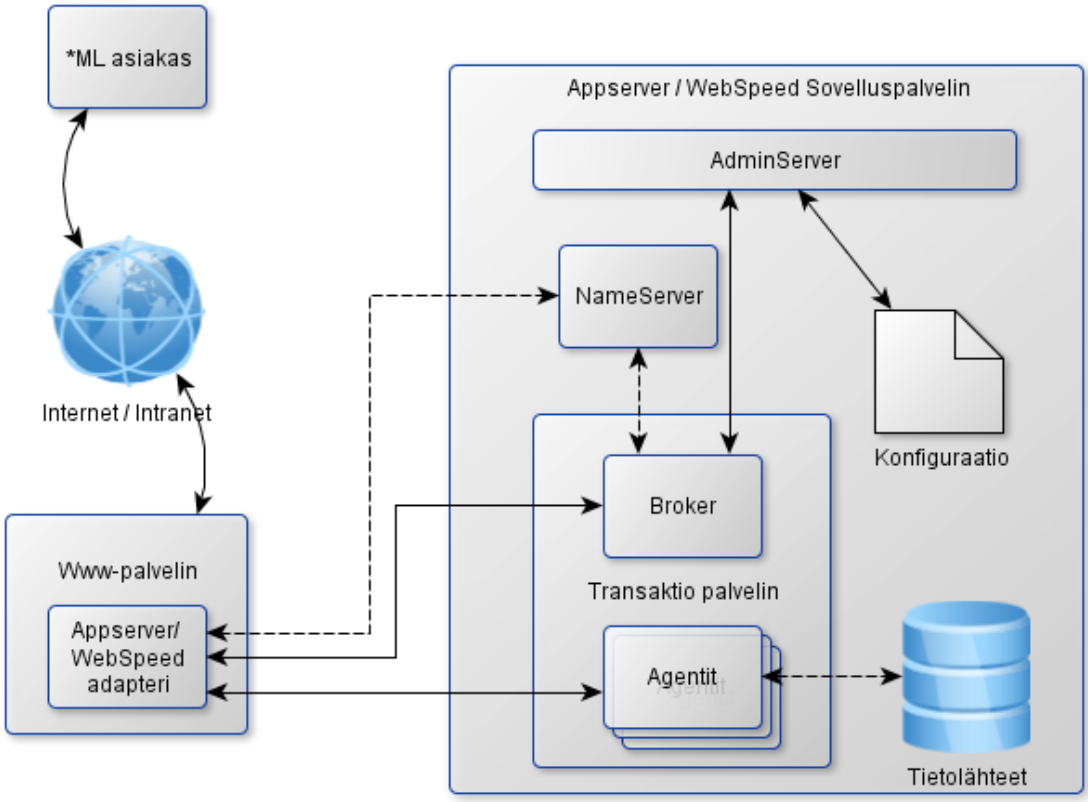
Suositeltu tapa ei kuitenkaan ole ainoa tapa palveluiden tuottamiseen. Appserver-alustan päälle on mahdollista rakentaa edustajärjestelmiä ja rajapintoja myös muilla teknologioilla. Näitä teknologioita ovat Java ja .Net. Malli perustuu edellä kuvattuun SOAP-pohjaisia palveluita tuottavaan arkkitehtuuriin. Erona SOAP-pohjaiseen toimintamalliin on se, että Java tai .Net sovellus kommunikoi AppServerillä olevan palvelun kanssa vastaavalla teknologialla tuotettua natiivia proxy- tai liittymäsovellusta hyödyntäen. Varsinainen web service -liittymä voidaan toteuttaa valitulla teknologialla. (Openedge Getting Started: Application and integration Services 2009, 4-1 – 4-2.)

AppServer-pohjaisten palveluiden lisäksi on OpenEdge sovelluskehitysalustalla mahdollista tuottaa web service –pohjaisia palveluja myös WebSpeed teknologiaa ja arkkitehtuuria hyödyntäen. OpenEdge WebSpeed on perinteisesti selaimella käytettävien www-sovellusten tuottamiseen tarkoitettu teknologia. WebSpeedissä on myös olemassa kaikki web service -palveluiden tuottamiseen tarvittavat ominaisuudet. WebSpeed-teknologialla voidaan tuottaa sekä RESTful, SOAP RPC sekä REST-RPC hybridi -tyyppisiä web service –rajapintoja ja palveluita.

3.5 Appserver ja Webspeed arkkitehtuurit

OpenEdge Appserver ja WebSpeed Broker -arkkitehtuurit koostuvat fyysisellä tasolla sovelluspalvelimesta ja edustapalvelimesta. Sovelluspalvelimella sijaitsevat Appserver tai Webspeed -prosessit vastaavat asiakkaan (client) palvelupyyntöjen suorittamisesta. Edustapalvelimella, yleensä www-palvelin, toimiva adapteri vastaanottaa palvelupyynnöt ja välittää ne käsiteltäväksi sovelluspalvelimelle. (Openedge Getting Started: Webspeed Essentials 2009, 1-2; Openedge Getting Started: Application and integration Services 2009, 2-4 – 2-5.)

Kuviossa 4 on esitetty Appserver ja Webspeed -sovellusympäristön tekninen ympäristö ja arkkitehtuuriin liittyvien komponenttien sijainti eri fyysisillä resursseilla. Katkoviivalla esitetyt nuolet kuvaavat yhteyksiä, joita ei jokaisessa ympäristössä välittämättä ole.



KUVIO 4. Openedge Webspeed sovellusympäristön tekninen arkkitehtuuri (Mukaillen Openedge Getting Started: Webspeed Essentials 2009, 1-2)

Www-palvelimella toimivan adapterin (Appserver Internet Adapter tai Webspeed messenger) tarkoituksena on kuunnella www-palvelimelle tulevia sovelluspalvelimelle tarkoitettuja palvelupyyntöjä ja välittää ne edelleen sovelluspalvelimelle. NameServer hallinnoi ja ylläpitää tietoja olemassa olevista Appserver tai WebSpeed -palveluista, eli brokereista. Nameserver voi ohjata Messengeriltä tulevan palvelupyynnön sitä tukevalle brokerille tarjoten näin mahdollisuuden skaalautumiseen sekä komponenttien hajauttamiseen useille fyysisille resursseille. NameServer ei ole pakollinen osa sovelluspalvelin ympäristöä. (Openedge Getting Started: Webspeed Essentials 2009, 1-2 – 1-4; Openedge Getting Started: Application and integration Services 2009, 2-4 – 2-5.)

Broker hallinnoi agentti-prosesseja ja ylläpitää niiden statustietoja. Broker-prosessi ohjaa messengeriltä tulevat palvelupyynnöt vapaille agenteille suoritettavaksi. Broker voi myös käynnistää ja sammuttaa agentteja tarpeen mukaan riippuen sovelluksen kuormituksesta. Agentit ovat varsinaisia työn tekeviä prosesseja. Agentti vastaanottaa palvelupyynnön, suorittaa siinä mainitun ohjelman, suorittaa ohjelmaa vastaavat tietokantatapahtumat sekä muodostaa asiakkaalle vastaukseksi esimerkiksi html-sivun. (Openedge Getting Started: Webspeed Essentials 2009, 1-2 – 1-4; Openedge Getting Started: Application and integration Services 2009, 2-4 – 2-5.)

4 ARKKITEHTUURI

Arkkitehtuuri on yleisnäkymä järjestelmään. Siinä esitetään pääkomponentit ja niiden toiminta suhteessa koko järjestelmään sekä niiden keskinäinen vuorovaikutus. (Achimugu, Babajide, Gambo, Oluwagbemi, Oluwaranti 2010, 933). Tämän perusteella arkkitehtuuria voi ajatella karttana, jossa paikat (kylät, kaupungit jne.) ovat komponentteja, paikkojen sijainti kartalla esittää niiden suhdetta toisiinsa ja koko järjestelmään sekä paikkojen väliset liikenneyhteydet (tiet, rautatiet, lentoreitit jne.) esittävät niiden keskinäistä vuorovaikutusta.

Arkkitehtuuri on suunnitelma sovelluksen kehittämiseen ja se kertoo kaikille sidosryhmille, miten järjestelmän pääasiallisesti tulisi toimia. Sen esitysmuoto on pelkistetty ja se keskittyy järjestelmän elementtien toimintaan ja vuorovaikutukseen. Arkkitehtuuri toimii siltana liiketoiminnallisten ja teknisten tavoitteiden ja vaatimusten sekä ohjelmiston välillä. Se ohjaa ohjelmistokehitystä koko sen elinkaaren ajan. (Achimugu ym. 2010, 933, 938.)

Jokainen ohjelmisto perustuu arkkitehtuuriin. Menestyvän ja tehokkaan ohjelmiston takana on yleensä myös hyvin suunniteltu arkkitehtuuri. (Achimugu ym. 2010, 933.)

4.1 Mitä on sovellusarkkitehtuuri?

Gortonin (2006, 2) mukaan sovellusarkkitehtuurin määrittelemiseksi ei ole olemassa yksiselitteistä ja yleisesti hyväksyttyä kuvausta. Tämä johtuu osittain siitä, että sovellusarkkitehtuuri on todella laaja aihealue, johon eri tahoilla on olemassa joukko erilaisia näkemyksiä. Lisäksi asiaan vaikuttaa se, että tietoinen arkkitehtuurisuunnittelu on kohtuullisen nuori asia.

IEEE:n (2000, 3), eli Institute of Electrical and Electronics Engineersin arkkitehtuuria käsittelevän standardin 1471-2000 mukaan arkkitehtuuri on järjestelmän perustavanlaatuinen rakenne, jota ilmennetään sen osien, osien keskinäisten

suhteiden, osien ja kokonaisuuden välisten suhteiden sekä sen suunnittelua ja kehitystä ohjaavien periaatteiden kautta. Samansuuntaisen tulkinnan sovellusarkkitehtuuriin antavat myös Rozanski ja Woods (2011, 12). Heidän mukaansa järjestelmän arkkitehtuuri on joukko keskeisiä järjestelmän ympäristöön sidonnaisia käsitteitä tai ominaisuuksia, joita kuvataan elementtien, suhteiden sekä suunnittelussa ja kehittämisessä käytettyjen periaatteiden avulla.

Yksinkertaisemmin arkkitehtuuria voi kuvata kokonaisuuden jakamiseksi järkeviin toisiinsa yhteydessä oleviin osiin. Osiin jakaminen sallii ja mahdollistaa ihmisten ja ihmisryhmien tuottavan työskentelyn, jonka tarkoituksena on yhteisen kokonaisuuden aikaan saamineni. Kokonaisuus on yleensä suurempi, kuin mitä yksittäinen ihminen tai ihmisryhmä olisi itsenäisesti pystynyt saamaan aikaan. Mitä suurempi ja monimutkaisempi järjestelmä on kyseessä, sen kriittisemmässä roolissa jakaminen on. (Clements, Bachmann, Bass, Garlan, Ivers, Little, Merson, Nord, Stafford 2010, 1 – 2.)

4.2 Palvelukeskeinen arkkitehtuuri

Yritykset ja organisaatiot käyttävät teknologioiltaan ja laitealustoiltaan eroavia järjestelmiä ja sovelluksia tukemaan liiketoimintaansa. Näiden joukossa on myös liiketoimintakriittisiä järjestelmiä ja sovelluksia. Tällaisessa ympäristössä toimiessa järjestelmien ja sovellusten korvaaminen kokonaan tai osittain sekä niiden siirtäminen toisille alustoille on liian riskialtista sekä kustannussyistä mahdotonta. (Gorton 2006, 219.)

Palvelukeskeisen arkkitehtuurin (Service Oriented Architecture, SOA) keskeisiä käsitteitä ovat palvelupohjainen toiminta- ja suunnittelumalli sekä eri teknologioista ja sovellusalustoista johtuvien käytännön yhteensopivuusongelmien konkreettinen ratkominen. Käytännönläheinen yhteentoimivuuden ja yhteensopivuuden painottaminen palvelukeskeisessä arkkitehtuurissa on seurausta nykyaikaisen liiketoiminnan monimuotoisuudesta ja sen tosiasian ymmärtämisestä että tämä monimuotoisuus ei jatkossa tule vähenemään. (Gorton 2006, 218 – 219.)

Teoreettisesti ajateltuna palvelukeskeinen arkkitehtuuri tarkoittaa kokoelmaa itsenäisiä toisiensa kanssa kommunikoivia palveluita. Kommunikointi voi tarkoittaa yksinkertaista tietojen välitystä tai se voi tarkoittaa useamman palvelun yhdessä koordinoimaa aktiviteettia. Palveluiden välinen kommunikointi mahdollistetaan web service -rajapinnoilla. (Barry 2003, 19.)

Barryn (2003, 84 – 85) mukaan palvelukeskeisen arkkitehtuurin tavoitteita ovat:

- hyväksyä ja hyväksikäyttää alan standardeja
- hyväksikäyttää kaupallisia valmisohjelmistoja mahdollisimman paljon
- mahdollistaa vanhojen sovellusten hyödyntäminen standardeja rajapintoja hyödyntäen
- datan sisällön ja rakenteen normalisointi sovellusten välisessä tiedonsiirrossa yhteensopivuuden takaamiseksi

4.2.1 Hyödyt

Monkin ja Wagnerin (2009, 224) mukaan palvelukeskeisen arkkitehtuurin suurimmat hyödyt liittyvät uusien sovelluksien toimintaympäristöön liittämisen helppouteen sekä avoimien standardien hyödyntämiseen. Palvelukeskeisen arkkitehtuurin hyödyntäminen vaikuttaa yrityksen kykyyn vastata liiketoiminnan muutoksiin, helpottaa ja nopeuttaa järjestelmien ja sovellusten välisiä integraatioita sekä mahdollistaa komponenttien uudelleenkäytön avulla säästöjä ajassa ja kustannuksissa.

Perinteisesti ajateltuna liiketoiminnan muuttuvat vaatimukset ovat olleet hyvin hankalia yritysten tietohallinto- ja it-organisaatioille. Palvelukeskeisen arkkitehtuurin pääasiallinen hyöty on sen sopivuus monimutkaisen ja osittain hallitsemattomankin liiketoiminnan muuttuviin tarpeisiin. Kunnolla suunniteltuna lähestymistapa mahdollistaa yritykselle ketterän ja kustannustehokkaan tavan uudistaa tarpeen mukaan tietojärjestelmiä ja niiden osia. (Barry 2003, 85 – 86.)

Barryn (2003, 85 – 86) mukaan palvelukeskeinen arkkitehtuurin hyötyjä osana yrityksen kokonaisarkkitehtuuria ovat:

- tietohallinto- ja IT-organisaatio voivat reagoida herkemmin liiketoiminta-lähtöisiin tarpeisiin
- kehitys- ja ylläpitokustannusten pieneneminen käyttäen kaupallisia valmisohjelmistoja
- kustannussäästöt järjestelmien välisissä integraatioissa standardeja web service -rajapintoja hyödyntäen
- mahdollisuus ottaa mukaan myös pieniä sisäisiä ja ulkoisia toimijoita, kuten yrityksiä, osaksi tiedonvälitystä

Edellä mainituista hyödyistä huolimatta ehkä merkittävin palvelukeskeisen arkkitehtuurin mukanaan tuoma etu on kuitenkin palvelujen, toisin sanoen ohjelmakoodin, uudelleen käytettävyys. (Monk & Wagner 2009, 224.)

4.2.2 Haasteet

Palvelukeskeisen arkkitehtuurin käyttöönotto tai hyödyntäminen ei useinkaan ole helppoa ja saattaa osoittautua ennakoitua monimutkaisemmiksi. Lisäksi palvelukeskeisen arkkitehtuurin hyödyntämisen taloudellisten hyötyjen arviointi on vaikeaa, esimerkiksi rahalliselle sijoitukselle on hankala arvioida tuottoastetta tai takaisinmaksuaikaa. (Monk & Wagner 2009, 224.)

Barryn (2003, 100 – 101) mukaan palvelukeskeisen arkkitehtuurin käyttöönottoon liittyy aina muutoksia ja muutostilanteessa ihmisten normaali reaktio on sen vastustaminen. Ihmisten muutosvastarinta saattaakin olla suurin yksittäinen este palvelukeskeisen arkkitehtuurin käyttöönotolle. Muutosvastarintaan vaikuttavia tekijöitä voivat olla ymmärryksen ja koulutuksen puute, yksittäisten vaikutusvaltaa omaavien henkilöiden asenne sekä huoli omasta työpaikasta.

4.2.3 Palvelukeskeisen arkkitehtuurin hyödyntäminen

Palvelukeskeisen arkkitehtuurin hyödyntäminen tapahtuu yrityksissä vaiheittain. Barryn (2003, 96 – 97) mukaan vaiheet ovat seuraavat:

- web service –rajapintojen ja palveluiden kokeileminen
- nykyisten järjestelmien mukauttaminen käyttämään web service –palveluita
- sisäisten järjestelmien keskinäisten riippuvuuksien poistaminen
- sisäisen palvelukeskeisen arkkitehtuurin perustaminen
- ulkoisten palveluiden tuominen osaksi yrityksen palvelukeskeistä arkkitehtuuria

Palvelukeskeisen arkkitehtuurin hyödyntäminen alkaa yleensä hyvin yksinkertaisista ja lyhyissä projekteissa tehtävistä kokeiluista. Näihin projekteihin osallistujat oppivat tutkimalla ja kokeilemalla konkreettisia asioita palvelukeskeisestä arkkitehtuurista ja web service -palveluista. Oppimisen ja kokeilujen lopputuloksena syntyy parhaita käytäntöjä (best practices) tukemaan palvelukeskeisen arkkitehtuurin käyttöönottoa yrityksessä. Kokeilujen jälkeen alkaa olemassa olevien järjestelmien sovittaminen osaksi web service –palveluita. (Barry 2003, 96.)

Varsinainen palvelukeskeisen arkkitehtuurin suunnittelu alkaa sisäisten järjestelmien keskinäisten riippuvuuksien kartoittamisella. Kartoituksen perusteella pyritään poistamaan arkkitehtuurin kannalta rajoittavat riippuvuudet ja minimoimaan sellaisten riippuvuuksien vaikutus, joita ei voida poistaa. Riippuvuuksien poistaminen mahdollistaa yrityksen sisäisen palvelukeskeisen arkkitehtuurin vakiinnuttamisen. Sisäisen palvelukeskeisen arkkitehtuurin vakiinnuttamisen jälkeen sitä voidaan laajentaa ottamalla mukaan yrityksen ulkopuolisia palveluita tai ottamalla käyttöön sitä tukevia uusia järjestelmiä tai sovelluksia. (Barry 2003, 97.)

Palvelukeskeisen arkkitehtuurin käyttöönotto synnyttää uusia kustannuksia niissä vaiheissa, joissa opitaan uutta ja keskitytään olemassa olevien järjestelmien kanssa toimimiseen. Arkkitehtuurin vakiinnuttamisesta alkaa kulujen aleneminen. Aluksi tämä johtuu ylläpitokustannusten pienenemisestä yleisen integraatioteknologian käyttöönotosta johtuen. Pitemmällä aikavälillä tarkasteltuna kustannussäästöjä syntyy erityisesti ohjelmistojen räätälöinneistä aiheutuneiden

kehityskulujen pienenemisestä sekä valmiiden arkkitehtuurin kanssa yhteensopivien ohjelmistojen hyödyntämisestä. (Barry 2003, 97.)

4.3 Web Service

W3C:n määritelmän mukaan (2004) web service –termillä tarkoitetaan järjestelmiä, jotka mahdollistavat yhteensopivan koneiden välisen vuorovaikutuksen tietoverkon ylitse. Tätä määritelmää voidaan tarkentaa siten, että web service on yksittäinen selkeästi määritelty, omavarainen ja riippumaton palvelu (Barry 2003, 19). Tämän perusteella voidaan ajatella, että järjestelmä voi tarjota useita palveluita.

Toisen määrityksen mukaan web service –termillä tarkoitetaan integraatioteknologian standardeihin perustuvia palvelukeskeisen arkkitehtuurin tarpeisiin vastaavia palveluita. Web service –pohjaisten palveluiden suunnittelua ja toteuttamista ohjaa yksinkertaisuus ja yhteentoimivuus, jotka toimivat myös niiden keskeisinä lähtökohtina. (Gorton 2006, 225.)

Web service –termillä tarkoitetaan standardia ja luotettavaa tapaa toteuttaa useiden järjestelmien välistä tietojen vaihtoa. Web service –pohjaiset ratkaisut mahdollistavat yhdistelmäsovelluksia olemassa olevien järjestelmien päälle ilman että olemassa olevia järjestelmiä tarvitsee korvata. Web service -ratkaisuja hyödyntäen tietojen vaihto on nopeaa ja osajärjestelmiä voidaan vaihtaa ja skaalata tarpeen mukaan. (Magal & Word 2009, 33.)

4.3.1 Hyödyt

Web service –pohjaisen teknologian hyödyt yrityksille kasvavat vähitellen. Web service -pohjaisia palveluita ja toimintoja voidaan käyttää olemassa olevien sovelluksien ja järjestelmien käytön parantamiseen. Organisaation ulkopuolisten tahojen tarjoamia palveluita voidaan hyödyntää omissa tietojärjestelmissä niiden toiminnallisuuksien laajentamiseen. Omien sovelluksien ja järjestelmien

perusrakenteiden muuttaminen tarjoamaan ja hyödyntämään palveluita on inkrementaalinen prosessi, jossa hyödyt yritykselle kasvavat prosessin edetessä. (Barry 2003, 62, 73; Abeysinghe 2008, 16.)

Hyötyjä web service –pohjaisista ratkaisuista löytyy sekä palveluiden ja toimintojen kuluttajille että tuottajille. Hyödyt voidaan jaotella karkeasti liiketoiminnallisiin ja teknisiin hyötyihin, joskaan rajaus ei aina ole näin selkeä. Kopackin ja Pottsin (2003, 21 – 31) mukaan web service –pohjaisten palveluiden ja toimintojen mahdollistamia asioita ovat:

- liiketoiminnan kustannussäästöt
- uusien liiketoimintamallien ja ansaintalogiikoiden hyödyntäminen
- ohjelmistokehityksen kustannussäästöt sekä kehityksen nopeutuminen
- nopeampi reagointi uusiin liiketoimintamahdollisuuksiin
- vanhojen järjestelmien integrointi sekä parempi järjestelmien välinen yhteensopivuus
- asiakkaiden ja yhteistyökumppaneiden integroiminen osaksi liiketoimintaympäristöä
- luotujen palveluiden ja toimintojen uudelleen käyttäminen

Suurin osa edellä mainituista hyödyistä vaikuttaa toisiinsa tai on toisistaan riippuvaisia. Esimerkiksi palvelujen uudelleen käyttäminen vaikuttaa suoraan ohjelmistokehityksen kustannuksiin ja kehitykseen käytettävään aikaan vähentävästi. Järjestelmien väliset integraatiot, ovat ne sitten sisäisiä tai ulkoisia, vaikuttavat liiketoiminnan kustannuksiin vähentävästi sekä antavat mahdollisuuden reagoida nopeammin liiketoiminnassa eteen tuleviin uusiin mahdollisuuksiin.

4.3.2 Haasteet

Mikään tekninen ratkaisu tai teknologia ei ole täydellinen. Vaikka web service -pohjaiset palvelut auttavat monenlaisten ongelmien ratkomisessa, tuovat ne myös mukanaan omat ongelmansa ja haasteensa. Kopackin ja Pottsin (2003, 34) sekä Gortonin (2006, 232) mukaan osa ongelmista johtuu web ce -palveluille ominaisesta teknologisesta pohjasta ja sen puutteista, osa johtuu

taas itse web service -teknologioiden määrityksistä ja osa siitä, että teknologioita ja ratkaisuja käytetään sellaisissa ympäristöissä ja tapauksissa, joihin niitä ei ole tarkoitettu.

Kopackin ja Pottsin (2003, 34 – 35) mukaan keskeisimpiä web ce -pohjaisiin palveluihin liittyviä haasteita ovat:

- saavutettavuus
- vaatimuksiin ja tarpeisiin vastaaminen
- rajapintojen staattisuus
- suoritusten luotettavuus

Internetissä ei yksikään sivusto tai www-sovellus ole 100 prosenttisesti saavutettavissa. Näin ollen myöskään samaan teknologiaan pohjautuvat web service –pohjaiset palvelut eivät voi olla aina saavutettavissa. Saavutettavuuteen voivat vaikuttaa hyvin monet tekijät aina käyttäjän ja palvelun välisestä verkkoyhteydestä palveluntarjoajien teknisiin ongelmiin asti. Tästä johtuen web ce -teknologiaa hyödyntäviin sovelluksiin täytyy rakentaa mekanismeja kutsujen uudelleen yrittämiseen ja virhetilanteiden käsittelemiseen. (Kopack & Potts 2003, 34.)

Vaatimuksiin ja tarpeisiin vastaaminen sekä rajapintojen staattisuus luovat myös omat ongelmansa. Aina löytyy jokin palvelua tai sen käyttöä tarvitseva taho, jonka tarpeet eivät osu yksiin toteutetun palvelun tai muiden sitä käyttävien tahojen kanssa. Samaan ongelmaan liittyy myös jo luodun ja käytössä olevan palvelun rajapinnan staattisuus. Käytössä olevan palvelun rajapinnan muuttaminen on aina riskialtista. Riippuen tehtävistä muutoksesta pitäisi todennäköisesti muuttaa myös kaikkia tai suurta osaa palvelua käyttävistä asiakkaista (clienteista). Web service –teknologioiden hyödyntämisen pääasiallinen tarkoitus on kuitenkin luoda yleiskäyttöisiä, isolle joukolle sopivia palveluja, joiden on tarkoitus pysyä kohtuullisen muuttumattomina. (Kopack & Potts 2003, 34.)

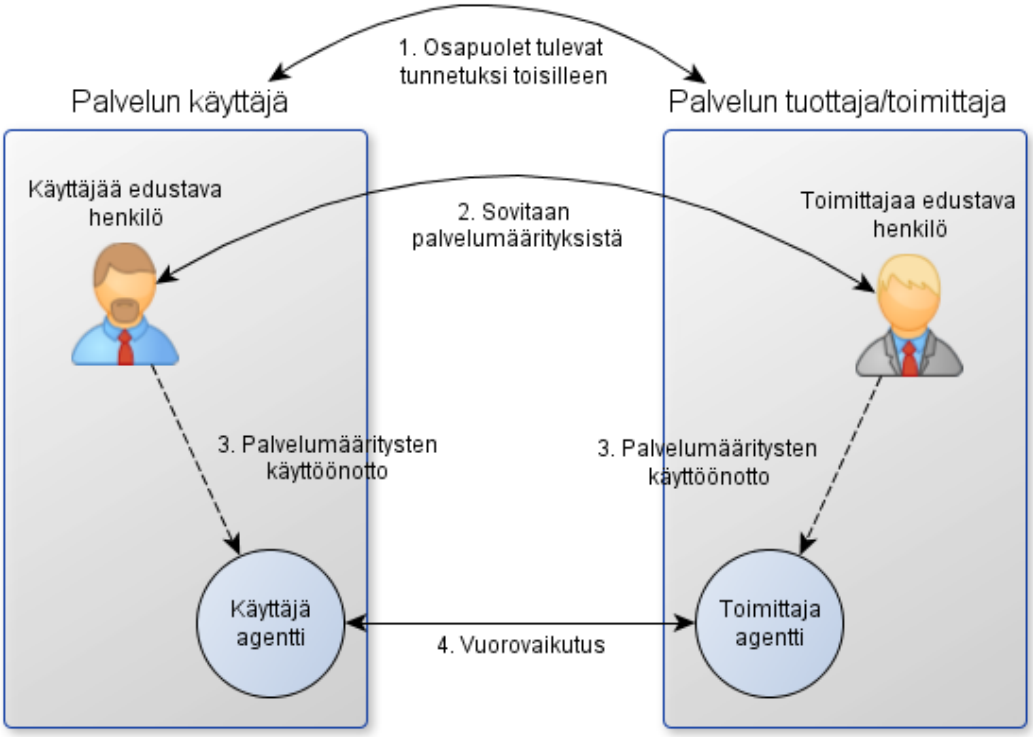
Web service –pohjaisia ratkaisuja käytettäessä myös suoritusten luotettavuus ja suorituskyky saattavat olla kyseenalaisia. Http-protokolla itsessään on epäluotettava protokolla, koska se ei takaa yhteyden onnistumista, tiedon välittä-

mistä tai vastausta lähetettyyn kutsuun. Web service -palveluita käytettäessä täytyy asiakkaan itse huolehtia tiedon välittymisen ja käsittelyn varmistamisesta erillisillä mekanismeilla. (Kopack & Potts 2003, 35.)

4.3.3 Web service -palveluiden hyödyntäminen

Web service –pohjaisten palveluiden hyödyntämiseen on olemassa useita tapoja. W3C:n (2004) mukaan yleisellä tasolla seurataan kuitenkin seuraavia kuvion 5 mukaisia vaiheita:

1. osapuolet tulevat toisilleen tunnetuiksi
2. osapuolet sopivat jollain menettelyllä palvelun käyttöä ja osapuolten välistä vuorovaikutusta ohjaavista palvelumäärityksistä
3. osapuolet ottavat käyttöön sovitut palvelumääritykset
4. palvelun käyttäminen aloitetaan, eli osapuolet ovat vuorovaikutuksessa toistensa kanssa jonkin sovitun toiminnon suorittamiseksi



KUVIO 5. Yleinen prosessi Web service –pohjaisen palvelun käyttöönottoon (mukaillen W3C 2004)

Ensimmäisessä vaiheessa käyttäjällä on tarve saada jokin tietty palvelu tai toiminto suoritettua. Käyttäjällä ei kuitenkaan välttämättä ole tietoa palvelun tarjoajasta. käyttäjä voi etsiä ja löytää palveluita esimerkiksi palvelurekistereistä, kuten UDDI (Universal Description Discovery and Integration), tai yleisistä hakupalveluista, kuten Google. Palvelun käyttäjä ja toimittaja ovat yleensä sopimussuhteessa toisiinsa palvelun käyttämiseen liittyen, sopimussuhde voi perustua vaikkapa käyttöehtojen hyväksyntään.

Sopivan palvelun löydyttyä täytyy molempien osapuolien, eli käyttäjän ja toimittajan, sopia palvelumäärityksistä. Jos palvelu on löytynyt automaattista prosessia hyödyntäen, löytyvät yleensä myös tarvittavat palvelumääritykset automaattisesti. Valmiin palvelun osalta palvelumääritykset käyttäjää varten ovat todennäköisesti saatavissa määrämuotoisesti, vaikka palvelu olisi löytynyt manuaalisestikin. Muissa tapauksissa osapuolten välillä käydään neuvottelu, jossa sovitaan palvelun muodosta ja sisällöstä.

Palvelumääritysten sopimisen jälkeen molempien osapuolien täytyy käyttöönottaa sovitut palvelumääritykset. Valmispalvelun osalta vain käyttäjä joutuu ottamaan palvelumääritykset käyttöön. Kun molemmat osapuolet ovat ottaneet palvelumääritykset käyttöönsä voi vuorovaikutus käyttäjän ja palvelun välillä alkaa.

4.4 Kilpailevat web service -arkkitehtuurit

Yhteistä kaikille web service –teknologioille on http-protokollan käyttäminen liikennöinnissä palvelun kuluttajan ja tarjoajan välillä. Http-protokollan päällä voidaan vaihtaa tietoa useilla tavoilla, kuten xml, json, html tai binääri-muotoisin dokumentein. Web service -teknologiat eroavat kuitenkin toisistaan sekä arkkitehtuurin että käytännön toiminnan osalta. Richardsonin ym. (2007, 4, 8) mukaan Web service -teknologioita voidaan ryhmitellä sen mukaan, miten ne vastaavat kahteen perustavaa laatua olevaan kysymykseen.

Ensimmäinen kysymys on, kuinka palvelun kuluttaja (client) voi välittää tiedon aikomuksistaan palvelulle. Siis kuinka palvelu voi erotella, mitä tietty pyyntö tar-

koittaa, onko se pyyntö tiedon hakemiseksi, jonkin tiedon poistamiseksi tai tallentamiseksi? Tätä sanotaan kutsun metodiksi (method information). (Richardson & Ruby 2007, 8.)

Toinen kysymys käsittelee sitä, kuinka kuluttaja (client) kertoo palvelulle, mitä palvelun tietokokonaisuuden osaa, eli resurssia, kutsu koskee. Jos palvelu ymmärtää kutsun metodin, eli mitä pitää tehdä, niin kuinka palvelu tietää, mille tiedolle metodi täytyy suorittaa? Tätä sanotaan kutsun kohteen rajaukseksi tai rajaamiseksi (scoping information). (Richardson & Ruby 2007, 11 – 12.)

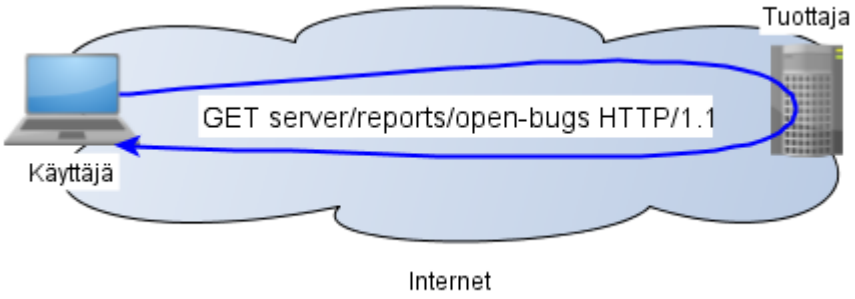
4.4.1 REST

REST määrittelee ja kuvaa arkkitehtuuria, jossa käyttäjille annetaan mahdollisuus yksinkertaisen http-protokollaan ja sen tunnettuihin metodeihin pohjautuvalla rajapinnalla käsitellä ja hyödyntää palvelun tuottajan resursseja. Käyttäjä voi siis lukea, lisätä, päivittää tai poistaa palvelussa olevia käsittelyn mahdollistavia kokonaisuuksia, kuten vaikkapa tuotetietoja. (Wikipedia, Web Services.)

REST-tyyppinen arkkitehtuuri perustuu siis osapuolten, palvelun käyttäjän (client) ja palvelun tuottajan (server), väliseen resursseja käsittelevään vuorovaikutukseen. RESTful arkkitehtuurin mukaisissa web service -palveluissa palvelun kutsun metodina käytetään http-protokollan metodia. Se, mitä palvelun halutaan tekevän, kerrotaan suoraan kutsussa käyttäen metodeja GET (lue), POST (päivitä), PUT (lisää) ja DELETE (poista). Se, mille tietokokonaisuudelle toiminta halutaan kohdistaa, kerrotaan kutsun osoitteessa. Osoite, eli URI (Uniform Resource Identifier), toimii sekä palvelun sijainnin että palvelun sisältävän resurssin globaalina tunnisteena ja näin määrittelee toiminnan kohteen yksiselitteisesti. (Richardson & Ruby 2007, 13; Wikipedia, Representational state transfer.)

Kuviossa 6 näytetään esimerkki http-kutsu RESTful web service -palveluun, joka hakee tietoa määritetystä osoitteesta (GET server/reports/open-bugs HTTP/1.1). Tiedon hakeminen, esimerkiksi poiston tai päivityksen sijaan, mää-

räytyy kutsun alussa olevalla metodilla GET. Se mitä tietoa mistä ja mitä haetaan, määräytyy metodin jälkeen kutsussa määritetyllä URI-komponentilla, eli osoitteella, server/reports/open-bugs. Kutsun viimeinen osa http/1.1 määrää mitä http-protokollan versiota kutsussa käytetään. (Richardson & Ruby 2007, 13.)



KUVIO 6. Esimerkki RESTful web service kutsusta

Tärkeää RESTful web service –pohjaisissa palveluissa on, että jokainen palvelu toimii samojen arkkitehtuurillisten periaatteiden mukaisesti ja hyödyntää http-protokollaa yhtenäisellä tavalla. Tällöin voidaan olla varmoja siitä, että eri palvelut toimivat samalla tavalla ja ovat yhteensopivia keskenään. Näin palvelun käyttäjien (client) tarvitsee opetella ainoastaan yleiset metodit käyttötarkoituksineen ja samalla voidaan välttyä palveluntarjoajien itse määrittämien metodien tuottamilta palveluiden välisiltä yhteensopivuusongelmilta. (Richardson & Ruby 2007, 106 – 107.)

Abeysinghen (2007, 7) mukaan RESTin yksinkertaisuus, helppokäyttöisyys sekä sen laaja kehittäjille tuttujen web-pohjaisten teknologioiden hyödyntäminen ovat vaikuttaneet siihen, että siitä on tullut suosittu, jopa ensisijainen, teknologia web service -palveluiden toteuttamiseen.

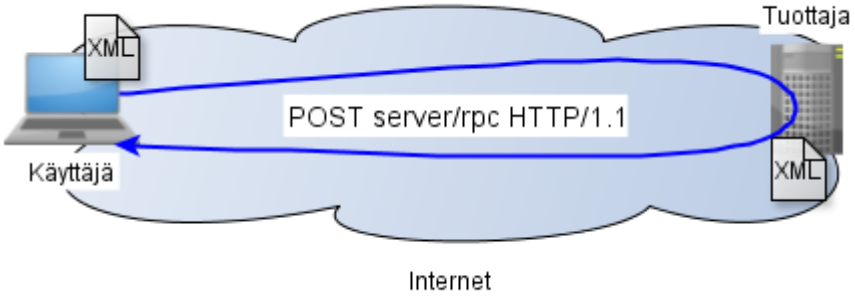
4.4.2 XML RPC

XML RPC -tyyppinen web service on palvelu, joka vastaanottaa ja lähettää xml-pohjaisia sanomia hyödyntäen http-protokollaa. Palvelut määrittelevät sekä me-

todin että kohteen rajauksen, eli resurssin, välitettävän sanoman sisällä. (Richardson & Ruby 2007, 14 – 15.)

Richardsonin ja Rubyn (2007, 14 – 15) mukaan isona erona RESTful arkkitehtuuriin on se, että XML RPC -tyyppiset palvelut eivät noudata mitään yleistä sovittua tapaa toimintojen määrittelemiseksi ja toteuttamiseksi, vaan jokainen palvelu määrittelee oman sanastonsa kutsuttavien palveluiden metodeille. Tästä johtuen eri palveluissa käytettävien metodien kirjo on valtava. Toisin sanoen samanlaisen asian aikaan saamiseksi voi eri palveluissa olla täysin toisistaan poikkeavat ja yhteensopimattomat metodit. Esimerkiksi nimiketietojen noutamiseksi voitaisiin kutsua metodia ”GetItem” yhtä hyvin, kuin metodia ”ShowProduct”.

Kuviossa 7 näytetään esimerkkinä http-kutsu XML RPC -tyyppiseen web service -palveluun, joka välittää palvelulle xml-sanoman käsittelyä varten. (POST server/rpc HTTP/1.1). Kutsusta itsestään ei voida päätellä varsinaista metodia eikä kutsun kohdetta, koska ne on määritelty välitettävän xml-sanoman sisällä. (Richardson & Ruby 2007, 106 – 107.)



KUVIO 7. Esimerkki XML RPC web service –pohjaisen palvelun kutsusta

Richardsonin ja Rubyn (2007, 15) mukaan XML RPC -tyyppisiä palveluita kutsuttaessa huomattavaa on, että varsinainen http-kutsu pysyy yleensä samana, mutta kutsussa välitettävän sanoman sisältö muuttuu kutsuttavan metodin ja käsiteltävän tietosisällön mukaan. Kuviossa 8 on kuvattu esimerkki siitä, millainen kuviossa 7 näytetyn palvelukutsun sisältämä xml-sanoma voisi olla. Kuviossa 8 esitetystä XML-sanomasta voidaan päätellä kutsuttava metodi (GetItemData) sekä tietosisältö, jota kutsu koskee (itemcode 123). Kyseinen XML-

sanoma on yhden palveluntarjoajan määrityksen mukainen, eikä sitä sanomaa voisi todennäköisesti käyttää toisen palvelun yhteydessä.

```
<xml version=”1.0”?>
<call>
<method>GetItemData</method>
<params>
<itemcode>123</itemcode>
</params>
</call>
```

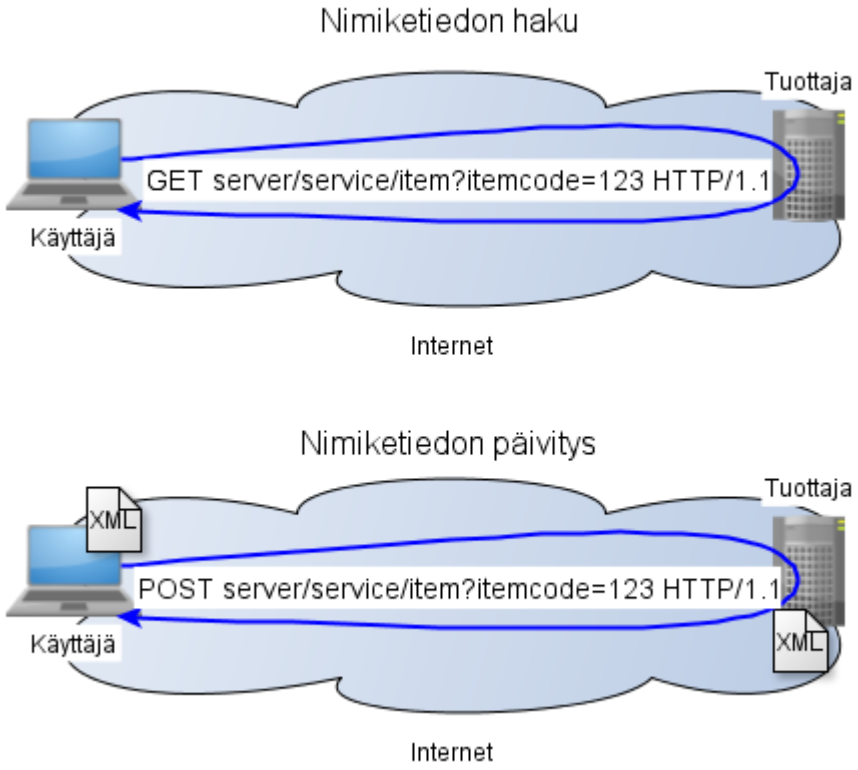
KUVIO 8. Esimerkki XML RPC -kutsun sisältämästä XML-sanomasta

Richardsonin ja Rubyn (2007, 14) mukaan yleinen tapa viestin välitykseen XML RPC-tyyppisissä palveluissa on SOAP (Simple Object Access Protocol). Abeysinghen (2008, 7) mukaan SOAP-pohjaisia web service -palveluita kuitenkin kritisoidaan niiden monimutkaisuuden ja raskauden vuoksi.

4.4.3 REST-RPC hybridit

REST-RPC -hybridit ovat palveluita, jotka sisältävät piirteitä sekä REST että XML RPC –arkkitehtuureista. Richardsonin ja Rubyn (2007, 16 – 18, 21) mukaan tällaisten palveluiden tunnusmerkkejä ovat yleensä REST-tyyppinen resurssien määrittely ja yksilöinti, REST-tyyppinen tietojen haku käyttäen GET-metodia sekä muuten XML-RPC -tyyppinen arkkitehtuuri.

Hybridi-palvelut käyttävät tietojen hakemiseen REST-arkkitehtuurin mukaisesti GET-metodia ja ne yksilöivät myös haettavan resurssin kutsun osoitteessa, siis osana kutsun URI-komponenttia. Muiden toimintojen osalta ne toteuttavat XML RPC tyyppisten palveluiden tapaa määritellä omat toimintonsa ja kutsutapansa. Kuviossa 9 esitetään esimerkki REST-RPC -hybridi palvelun mukaisesta tietojen hausta ja päivittämisestä.



KUVIO 9. Tiedon haku ja päivitys REST-RPC hybridi palvelussa

Richardsonin ja Rubyn (2007, 16 – 17) mukaan tällaisia palveluita tuotetaan yleensä, jos kehittäjillä on paljon kokemusta web-sovelluksista ja vähän kokemusta tai tietoa REST-arkkitehtuurista. Lisäksi he toteavat, että tällaisen arkkitehtuurin tuottaminen ei juuri koskaan ole suunnittelun lähtökohta tai tavoite. Tästä huolimatta voidaan tällaisella toimintamallilla toteuttaa kohtuullisella työmäärällä ja kustannuksilla toimiva web service –rajapinta sellaisia teknologioita hyödyntäviin sovelluksiin, joissa ei tueta REST-arkkitehtuuria.

4.5 SOA + Web service

Palvelukeskeinen arkkitehtuuri (Service-Oriented Architecture) ja web service -standardi ovat pyrkimys tuottaa ratkaisu eri teknologioilla tuotettujen palveluiden yhteensopivuus ja yhteentoimivuus ongelmiin. Samalla niiden avulla pyritään luomaan perusta Internetin laajuisten hajautettujen sovellusten luomiseksi. (Gorton 2006, 218.)

Sekä Monkin ja Wagnerin (2009, 42) että Magalin ja Wordin (2009, 32 – 33) mukaan palvelukeskeinen arkkitehtuuri ja web service -standardi ovat sama asia. SOA termiä käytetään, kun asiasta puhutaan yleisemmällä tasolla ja arkkitehtuurin näkökulmasta. Web service -termiä käytetään puhuttaessa samasta asiasta tarkemmalla ja teknisellä tasolla.

Barryn (2007, 73) mukaan palvelukeskeinen arkkitehtuuri ja web ce -pohjaiset palvelut voidaan nähdä käynnissä olevana tietojärjestelmien ja niiden välisen vuorovaikutuksen kehitysprosessina. Alkuvaiheessa käytetyt yksinkertaiset palvelukeskeistä arkkitehtuuria ja web service –pohjaisia palveluita hyödyntävät ratkaisut kehittyvät ja kasvavat yritysten ja ihmisten oppiessa ja saadessa lisää kokemusta ja käytettävien standardien kehittyessä.

Ovatpa perustelut tai väittämät mitkä tahansa, voidaan yleisesti todeta, että olemme mukana kehitysprosessissa, jossa palvelukeskeisen ajattelun ja teknologiariippumattomien ratkaisujen avulla pyritään minimoimaan ”henkiset” ja tekniset esteet globaalien sovellusten tieltä. Esitellyillä ratkaisuilla pyritään kohti tilaa, jossa sovelluksia voidaan rakentaa liittämällä yhteen tarvittavia palveluita ilman että jokaista tarvitsee tuottaa itse.

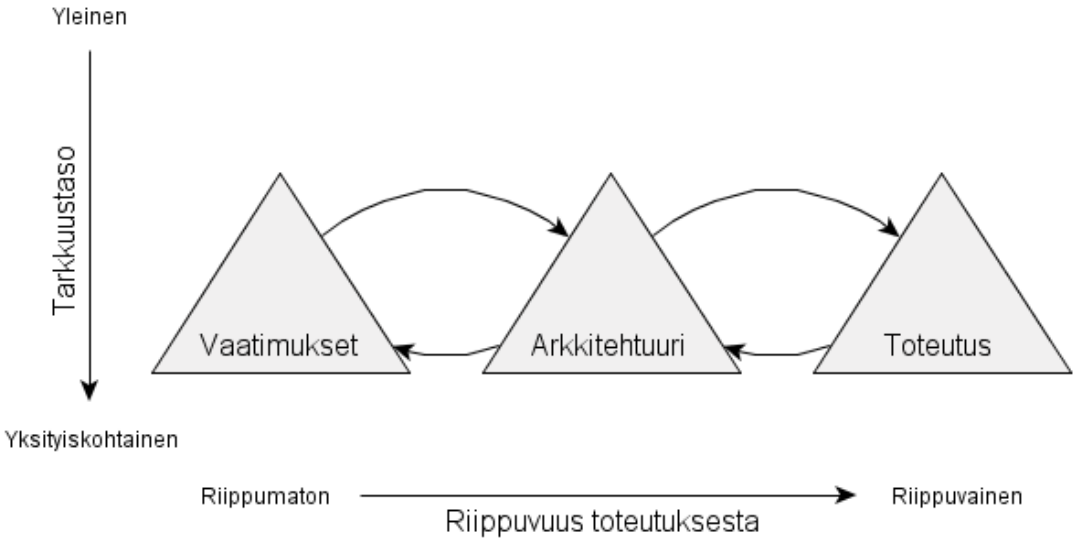
5 ARKKITEHTUURISUUNNITTELU

Arkkitehtuurisuunnittelu on monipuolista työskentelyä, johon liittyy huomattava määrä sosiaalisia toimintoja pelkän ohjelmistosuunnittelun lisäksi. Arkkitehtuuria suunniteltaessa täytyy toimia vuorovaikutuksessa sovellukseen liittyvien sidosryhmien kanssa ja pyrkiä koostamaan ja yhtenäistämään niiden toiminnalliset vaatimuksensa osaksi suunnitelmaa. Koostettujen vaatimusten pohjalta luodaan arkkitehtuuri yhdessä suunnittelijoiden kanssa. Arkkitehtuurityöllä on vaikutusta myös projektisuunnitelmaan, projektin töiden jakamiseen ja arviointiin sekä projektin aikataulutukseen. (Gorton 2006, 91.)

5.1 Arkkitehtuurisuunnitteluprosessi

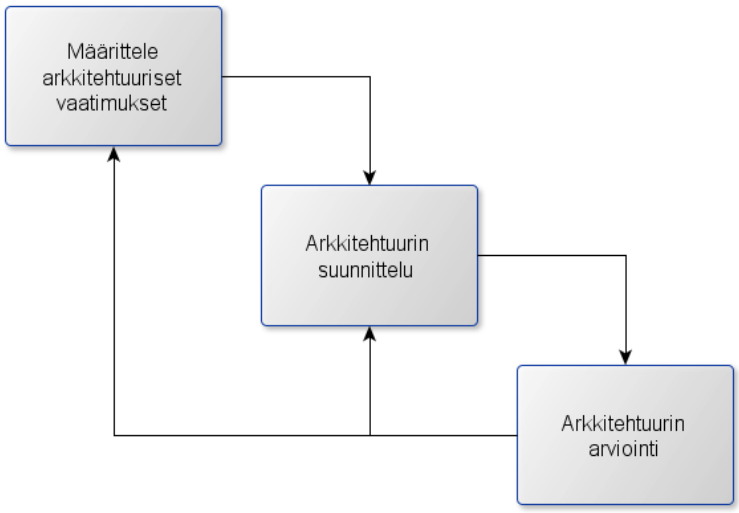
Arkkitehtuuri muodostaa sillan vaatimusten ja suunnittelun väliin sovittaen niiden asettamia ehtoja ja rajoituksia yhdeksi kokonaisuudeksi. Toisin sanoen arkkitehtuurisuunnittelu tapahtuu vaatimusmäärittelyn ja varsinaisen toteutustyön välissä.

Kolmen huipun mallissa (Three Peaks model) kuvataan kolmen sovelluskehityksen vaiheen keskinäistä vuorovaikutusta. Malli on kuvattu kuviossa 10. Kuviossa olevat kolme kolmiota kuvastavat sovelluskehityksen vaiheita ja niiden alaspäin levenevä muoto edustaa sitä, miten yksityiskohdat lisääntyvät ajan kuluessa. Kolmioiden väliset nuolet edustavat vaiheiden välistä vuorovaikutusta ja sitä miten ne ovat yhä enemmän sidoksissa toisiinsa ajan kuluessa ja varsinaisen sovellustyön edetessä. Vaikkakin vaatimusmäärittely, arkkitehtuurisuunnittelu ja varsinainen sovelluskehitys ovat erillisiä sovelluskehityksen vaiheita, niin niillä on siitä huolimatta merkittävä vaikutus toisiinsa. (Rozanski & Woods 2011, 87 – 88.)



KUVIO 10. Sovelluskehityksen "Three Peaks" –malli (mukaillen Rozanski & Woods 2011, 88)

Kuviossa 11 esitellään yksinkertainen kolmitasoinen arkkitehtuurisuunnittelun tueksi tarkoitettu prosessi, jota voidaan käyttää ohjaamaan ja tukemaan arkkitehtuuri-suunnittelua. Ensimmäisessä vaiheessa luodaan vaatimuksien perusteella arkkitehtuurityötä ohjaava malli. Toisessa vaiheessa suunnitellaan arkkitehtuurin muodostavien komponenttien rakenne ja vastuut. Viimeisessä vaiheessa arkkitehtuuria arvioidaan testaamalla sen toimintaa asetettuja vaatimuksia vasten. Esitetty malli on iteratiivinen ja vaiheita käydään lävitse kunnes arkkitehtuuri-suunnitelma toteuttaa asetetut vaatimukset. (Gorton 2006, 91 – 92.)



KUVIO 11. Arkkitehtuurisuunnittelun kolme vaihetta (mukaillen Gorton 2006, 92)

5.2 Arkkitehtuurivaatimusten tunnistaminen, määrittely ja priorisointi

Ennen varsinaisen arkkitehtuurin suunnittelua täytyy olla hyvä ymmärrys niistä vaatimuksista, jotka arkkitehtuurin täytyy toteuttaa. Arkkitehtuurille merkittäviä vaatimuksia ovat järjestelmälle tai sovellukselle asetetut ei-toiminnalliset vaatimukset ja rajoitukset. Muita arkkitehtuurisuunnittelussa huomioitavia asioita voivat olla myös eri sidosryhmien asettamat toiminnalliset vaatimukset. Vaatimusmäärittely tuottaa siis arkkitehtuurisuunnitellulle alustavat puitteet. (Gorton 2006, 92 – 93; Rozanski & Woods 2011, 88.)

Järjestelmälle asetetuista vaatimuksista ja rajoituksista pyritään tunnistamaan sellaiset, jotka asettavat selkeitä tavoitteita ja reunaehtoja tuotettavalle arkkitehtuurille. Tunnistettujen vaatimusten ja rajoitusten osalta määritetään sen varsinainen vaikutus arkkitehtuuriin. Vaatimukset asettavat arkkitehtuurille tavoitteita, joihin pitää pyrkiä ja rajoitteet asettavat ehtoja sille, millä keinoilla tavoitteet voidaan saavuttaa. Rajoitteita ei yleensä pystytä muuttamaan tai poistamaan ja ne on huomioitava arkkitehtuurisuunnittelussa sellaisenaan. (Gorton 2006, 93 – 94.)

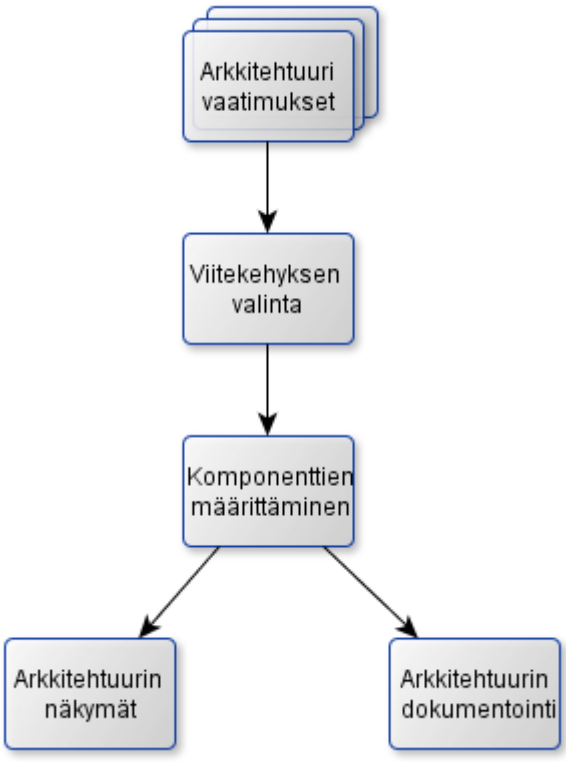
Arkkitehtuurille tunnistetut ja määritetyt vaatimukset eivät useinkaan ole samanarvoisia. Asetettujen vaatimusten osalta on tärkeää tunnistaa niiden merkitys lopulliselle sovellukselle ja järjestää tai ryhmitellä ne tärkeyden mukaan. Järjestämiseen tai ryhmittelyyn voidaan käyttää yksinkertaista asteikkoa 1 (pakollinen), 2, 3 (voisi sisältää) kuvastamaan vaatimuksien tärkeyttä arkkitehtuurille. (Gorton 2006, 94 – 95.)

Ongelmia vaatimusten priorisointiin tuo ristiriidassa toistensa kanssa olevat vaatimukset. Ristiriitatilanteiden selvittämiseen ei ole olemassa yksiselitteistä ja helppoa tapaa, vaan jokainen ristiriita on arvioitava ja ratkaistava tapauskohtaisesti yhdessä vaatimuksiin liittyvien sidosryhmien kanssa. Arkkitehtuurisuunnitteluun kuuluu sellaisten ratkaisumallien etsiminen, jotka mahdollisimman hyvin toteuttavat ristiriidassa olevat vaatimukset. Näiden ratkaisumallien hyödyntämisestä ei saa myöskään seurata negatiivisia vaikutuksia sidosryhmille, arkkitehtuurille tai sovellukselle. (Gorton 2006, 95.)

5.3 Arkkitehtuurin suunnittelu

Arkkitehtuurin suunnittelu ja sen lopputuloksena syntyvän arkkitehtuurisuunnitelman laatu ovat merkityksellisiä kokonaisuuden kannalta. Arkkitehtuurin suunnittelua aloitettaessa ei tiedetä järjestelmän tai sovelluksen kokoa eikä laajuutta, alueiden monimutkaisuutta, suurimpia riskejä tai mahdollisia sidosryhmien välisiä konflikteja. Arkkitehtuurin suunnittelu on tärkein ja samalla haastavin tehtävä koko arkkitehtuuriprosessissa. Hyvän arkkitehtuurin suunnittelu vaatii yleensä vuosien kokemuksen. (Gorton 2006, 95 – 96; Rozanski & Woods 2011, 85.)

Kuviossa 12 kuvataan arkkitehtuurisuunnittelun prosessi ja sen saamat syötteet ja tulosteet. Suunnittelutyön pohjana, eli syötteenä, ovat määritetyt ja priorisoidut vaatimukset ja rajoitukset. Varsinainen arkkitehtuurin suunnittelu koostuu arkkitehtuuri viitekehyksen valinnasta sekä arkkitehtuurin sisältämien komponenttien määrittämisestä. Suunnittelun lopputuloksena, eli tulosteena, tuotetaan arkkitehtuurin kannalta merkitykselliset näkymät kaavioina sekä dokumentoidaan suunnitelma, suunnittelussa tehtyihin keskeisiin valintoihin vaikuttaneet syyt sekä mahdolliset arkkitehtuuriin liittyvät riskit. (Gorton 2006, 96.)



KUVIO 12. Arkkitehtuurin suunnittelu (mukaillen Gorton 2006, 96)

Suurin osa arkkitehtuurisuunnittelusta koskee sovelluksen osioimista järkeviin toisiinsa liittyviin osatekijöihin. Näitä osatekijöitä voivat olla komponentit, moduulit, objektit tai mikä tahansa muu selkeästi tunnistettavissa oleva osa ohjelmistoa tai sovellusta. Järkevä tapa tehdä osiointi perustuu sovellukselle asetettuihin vaatimuksiin ja rajoitteisiin. Suunnittelutyön lopputuloksena syntyvän arkkitehtuurin täytyy vastata niitä erityisiä juuri kyseiselle sovellukselle asetettuja vaateita ja rajoituksia. (Gorton 2006, 3.)

5.3.1 Viitekehyksen valinta

Arkkitehtuurillisen viitekehyksen valinta perustuu keskeisten, prioriteetiltaan tärkeimpien vaatimusten toteuttamiselle. Tunnettujen suunnittelumallien (design pattern) ja viitekehysten (framework) hyödyntäminen minimoi kehityksen epäonnistumisen sopimattoman arkkitehtuurin vuoksi. Nämä mallit ja viitekehykset ovat uudelleen käytettäviä ohjeita merkittävien arkkitehtuurivalintojen tekemisen helpottamiseksi. (Gorton 2006, 5, 97; Clements. ym. 2010, 17.)

Käytännön tasolla suunnittelumallit ja viitekehykset ovat toimintatapoja yleisesti esiintyvien ongelmatapausten ratkaisemiseksi. Ne eivät ole kuitenkaan valmiita konkreettisia ratkaisuja. Ne ovat enemmänkin ohjenuoria oman ratkaisun löytämiseksi tilanteessa, jossa ongelma ja siihen vaikuttavat tekijät tunnetaan.

Tunnetun viitekehyksen tai suunnittelumallin hyödyntämisen edut perustuvat sille, että niiden osalta tiedetään kuinka ne käsittelevät ja toteuttavat eitoiminnallisia vaatimuksia. Jokaiseen suunnittelumalliin liittyy myös omat suunnittelua rajoittavat ja sille vaatimuksia asettavat piirteensä. Suunnittelumalli keskittyy tiettyyn ongelmatilanteeseen ja esittää siihen vakioidun ratkaisun. Yksinkertaisten sovellusten osalta käytetään yleensä yhtä suunnittelumallia viitekehyksenä. Laajat järjestelmät hyödyntävät useita suunnittelumalleja yhdistettynä tavalla, joka parhaiten vastaa järjestelmälle asetettuja vaatimuksia ja rajoituksia. (Gorton 2006, 5, 97; Clements. ym. 2010, 17.)

5.3.2 Komponenttien määrittäminen

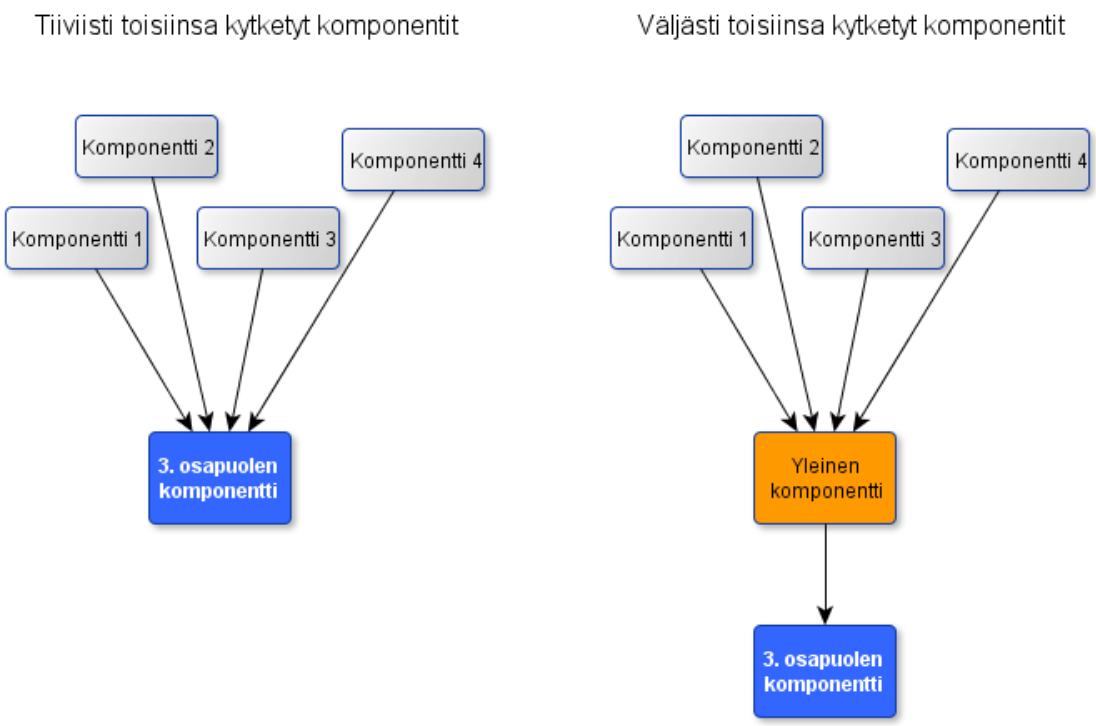
Arkkitehtuurisuunnittelua ohjaavan viitekehyksen, tai useamman, valinnasta alkaa tärkeimpien arkkitehtuurin muodostavien komponenttien, eli sovelluksien osien määrittäminen. Määrittämisen yhteydessä kullekin komponentille määrätään jokin vastuu tai rooli. Vastuulla ja roolilla tarkoitetaan niitä toimintoja, joista kyseisen komponentin tulee suoriutua sovelluksessa. Arkkitehtuuri on sidos, joka saa komponenttien roolien ja vastuiden kokonaisuuden toimimaan johdonmukaisesti. Komponenteista muodostuvan kokonaisuuden kautta arkkitehtuuri vastaa sovellukselle asetettuihin vaatimuksiin. (Gorton 2006, 3 – 4.; Clements. ym. 2010, 2.)

Arkkitehtuurissa kuvatut komponentit ovat ylimmän tason kokonaisuuksia, joita järjestelmä tai sovellus toteuttaa. Gortonin (2006, 106 – 107) mukaan komponenttien määrittämisessä voidaan hyödyntää seuraavia suuntaviivoja:

- komponenttien välisten riippuvuuksien minimointi
- komponentilla pitää olla selkeästi määritetty joukko vastuita, jotka yhdessä toteuttavat loogisen toiminnon
- rajaa ja minimoi riippuvuudet väliohjelmistoihin (middleware) ja valmistuotteisiin (COTS, Commercial-of-the-self)
- suunnittele ja erittele komponenttien rakenne hierarkisesti ulkoisiin ja sisäisiin toimintoihin ja rajapintoihin
- komponenttien välisten kutsujen minimointi

Edellä luetellussa listassa useimmat kohdat keskittyvät eri kokonaisuuksien välisten riippuvuuksien minimoimiseen. Riippuvuuksien minimointi onkin yksi tärkeimmistä rakenteellisista asioista arkkitehtuurisuunnittelussa. Komponenttien välillä on olemassa riippuvuus silloin, jos muutos yhdessä komponentissa pakottaa tekemään muutoksia myös muissa komponenteissa. Tarpeettomien riippuvuuksien poistaminen komponenttien väliltä auttaa rajaamaan muutostarpeet, eivätkä ne pääse leviämään arkkitehtuurin kautta koko sovellukseen. (Gorton 2006, 4.)

Kuviossa 13 esitetään kaksi esimerkkiä siitä kuinka komponenttien välisiä riippuvuuksia voi esiintyä sovelluksessa. Kyseisessä kuviossa vasemmalla puolella on esimerkki tilanteesta, jossa sovelluksen sisäiset komponentit ovat tiiviisti kytkettyjä ulkoiseen komponenttiin. Tässä tilanteessa muutos kolmannen osapuolen komponenttiin tai sen vaihtaminen kokonaan toiseen aiheuttaisi muutoksia kaikkiin kuvion neljään muuhun komponenttiin. Oikealla puolella olevassa esimerkissä neljän sisäisen komponentin ja ulkoisen komponentin välillä esiintyy väljä riippuvuus, joka on toteutettu hyödyntäen viidettä sisäistä yleiskäyttöistä komponenttia. Tässä tilanteessa ulkoisen komponentin vaihtaminen tai muuttaminen aiheuttaa muutostarpeen ainoastaan viidenteen komponenttiin, muut sisäiset komponentit voivat toimia kuten ennenkin. (Gorton 2006, 4.)



KUVIO 13. Kaksi esimerkkiä komponentti riippuvuuksista (mukaillen Gorton 2006, 4)

5.4 Arkkitehtuurin arviointi

Kun arkkitehtuurin viitekehys tai viitekehykset on valittu ja arkkitehtuurin muodostavat komponentit on määritetty, ollaan vaiheessa, jossa keskeiset arkkitehtuuripäätökset on tehty. Tällöin suurimpiin tunnistettuihin ongelmakohtiin on

määritetty ratkaisu ja järjestelmän tai sovelluksen yleinen rakenne on suunniteltu. Arkkitehtuurin toimivuudesta käytännössä ei kuitenkaan voida olla varmoja. (Rozanski & Woods 2011, 217.)

Arkkitehtuurin arvioinnin tarkoituksena on vakuuttaa suunnittelijat ja sidosryhmät siitä, että suunniteltu arkkitehtuuri on tarkoitukseensa sopiva ja toimiva. Arvioinnissa tarkastellaan myös, onko suunnittelun aikana tehty oikeita päätöksiä ja valintoja kilpailevien tai ristiriidassa olevien vaatimusten osalta. Arkkitehtuuri ei ole kuitenkaan konkreettisin menetelmin arvioitavissa tai testattavissa, eikä sitä myöskään voi suorittaa tai ajaa, jotta nähtäisiin kuinka se toimii ja suoriutuu. (Rozanski & Woods 2011, 217; Gorton 2006, 109.)

Tärkeimpiä arvioinnissa huomioitavia kohtia ovat:

- tehtyjen yleistysten ja oletusten oikeellisuus
- teknisten päätösten oikeellisuus ja yhdenmukaisuus
- arkkitehtuurin selittäminen ja hyväksyttäminen sidosryhmillä

(Rozanski & Woods 2011, 218 – 219)

Arkkitehtuurin arviointi tarjoaa myös:

- luonnollisen tilanteen päätöksenteolle liittyen arkkitehtuuriin tai järjestelmään ja sovellukseen kokonaisuutena
- perustan muodollisen sopimuksen tekemiselle

(Rozanski & Woods 2011, 218 – 219)

Arkkitehtuurin järkevään arviointiin on olemassa joukko hyödyllisiä lähestymistapoja. Oikea lähestymistapa vaihtelee käyttötarkoituksen ja tilanteen mukaan. Yleisiä lähestymistapoja ovat:

- esittely (Presentation)
- muodollinen katselmointi (Formal Review) tai jäsennelty läpikäynti (Structured Walkthrough)
- testitapauksiin perustuva manuaalinen testaus
- prototyyppi tai toteutettavuusdemo (proof-of-concept)
- sovellusrunko (skeleton system)

(Rozanski & Woods 2011, 219 – 225)

Esittely on yksinkertainen ja epämuodollinen tapa esitellä arkkitehtuuria sidosryhmille. Yksinkertaisuudestaan huolimatta esittely on hankala tapa konkreettiseen arviointiin. Jotta sen avulla voitaisiin arvioida arkkitehtuuria tai sen keskeisiä piirteitä ja päätöksiä, pitäisi kaikkien esittelyyn osallistuvien henkilöiden olla asiaan perehtyneitä sekä kyetä arkkitehtuurin kriittiseen arviointiin. Näin ollen esittely itsessään ei varsinaisesti edesauta arviointia, vaan se on parhaimmillaan viestinnän välineenä sekä lähtökohtana keskeisten asioiden käsittelylle yhdessä sidosryhmien kanssa. (Rozanski & Woods 2011, 219 – 220.)

Arkkitehtuurin muodollinen katselmointi tai jäsennelty läpikäynti on tehokas tapa arvioida arkkitehtuurin toimivuutta yhdessä sidosryhmien kanssa. Muodollinen katselmointi mahdollistaa sidosryhmille keskeisten huolenaiheiden läpikäynnin sekä varmistumisen siitä, että ne on ymmärretty ja ratkaistu oikein. Arkkitehtuurin arviointi tällä lähestymistavalla vaatii merkittävän määrän valmistelua kaikilta arviointiin osallistuvilta osapuolilta. Katselmoinnissa arkkitehtuuria läpikäydään ennalta sovitun suunnitelman mukaisesti keskustellen ja tehden päätöksiä mahdollisesti arkkitehtuuriin tarvittavista muutoksista. Sidosryhmiltä saatava palaute edesauttaa arkkitehtuurin ja sen dokumentoinnin parantamisessa. (Rozanski & Woods 2011, 220 – 222.)

Manuaalinen arkkitehtuurin testaus perustuu vaatimusten syvälliseen ymmärtämiseen sekä vaatimuksista muodostettuihin testitapauksiin, joita järjestelmä tai sovellus tulee käyttönsä aikana todennäköisesti kohtaamaan. Testitapaus sisältää järjestelmän tai sovelluksen ei-toiminnalliseen vaatimukseen kohdistuvan ärsykkeen, johon reagoimista arvioidaan suunniteltua arkkitehtuuria hyödyntäen. Testitapaus läpäistään, jos arkkitehtuuri reagoi ärsykkeeseen toivotulla tavalla. Jos testitapausta ei läpäistä tai sen arviointi on hankalaa, on arkkitehtuurissa toden-näköisesti korjausta vaativa ongelma tai ainakin toteutuksessa huomioitava riski. (Rozanski & Woods 2011, 222 – 223; Gorton 2006, 109 – 110.)

Manuaalinen arkkitehtuurin testaus antaa vastauksia arkkitehtuurin toimivuudesta käytännön tilanteissa ennen kuin sovellusta on edes olemassa. Manuaa-

lisesti läpikäytävien testitapausten, eli käytännön tilanteiden, tuottaminen on aikaa vievää, mutta niiden läpikäynti arkkitehtuuria hyväksikäyttäen tuottaa kohtuullisen hyvän tuloksen arkkitehtuurin toimivuudesta. Manuaalisen testauksella ei kuitenkaan voida vastata kaikkiin, varsinkaan monimutkaisempiin, ongelmiin tyydyttävästi.

Prototyyppi eli toteutettavuusdemo (Proof-of-Concept) on pieni ja rajoitettu toteutus jostain järjestelmän tai sovelluksen osa-alueesta. Sitä käytetään arvioimaan arkkitehtuuria tapauksissa, joihin liittyy suuri riski tai jotka ovat huonosti ymmärrettyjä. Yleensä prototyyppien toteutuksella pyritään pienentämään uudesta teknologiasta johtuvia teknisiä riskejä. Prototyypin rakentamisen ja testaamisen perusteella voidaan arkkitehtuuria arvioida luotettavasti. Rakennettuja proto-tyyppejä ei ole tarkoitus käyttää osana varsinaista järjestelmää tai sovellusta, niiden tarkoitus on ainoastaan tuottaa riittävästi tietoa päätöksenteon pohjaksi. (Rozanski & Woods 2011, 224 – 225; Gorton 2006, 112 – 113.)

Prototyypin tai toteutettavuusdemon toteuttaminen arkkitehtuurillisesti haastavasta osa-alueesta antaa konkreettista informaatiota arkkitehtuurisuunnitelman onnistumisesta. Prototyypin tai toteutettavuusdemon ongelmana on sen pitäminen riittävän rajattuna ja yksinkertaisena, jotta se voidaan hylätä tarkoituksen täyttyessä. Hylkäämistä ei kuitenkaan aina tapahdu onnistuneiden prototyyppien osalta, vaan ne jäävät varsin usein osaksi varsinaista toteutettavaa sovellusta.

Sovellusrunko (skeleton system) on huomattavasti prototyyppiä pitemmälle viety toteutus järjestelmästä tai sovelluksesta, jota arkkitehtuurisuunnitelma koskee. Sovellusrunko on periaatteessa lopullisen järjestelmän tai sovelluksen ensimmäinen versio, joka toteuttaa keskeisimmät rakenteet, mutta vain minimaalisen joukon varsinaisia toimintoja. Vaikka sovellusrungon toteuttaminen luo parhaan mahdollisen tilanteen arkkitehtuurin arviointiin, on se samalla kustannuksiltaan myös selkeästi kallein tapa arvioida arkkitehtuurin soveltuvuutta käytäntöön. (Rozanski & Woods 2011, 225.)

Sovellusrunko on usein onnistuneen prototyypin tai toteutettavuusdemon seuraava vaihe. Sovellusrunko on hyvin pitkälle viety arkkitehtuurisuunnitelmaa vastaava toteutus järjestelmästä tai sovelluksesta. Yleensä näin pitkälle vietyä toteutusta järjestelmästä tai sovelluksesta ei enää pidetä arkkitehtuurin arviointina, vaan osana varsinaista järjestelmän tai sovelluksen tuottamista. Yleensä sovellusrunkoa on hankala hylätä ilman taloudellisia menetyksiä – menetyksen merkittävyys riippuu täysin sen toteuttamiseen käytetystä työajasta.

Tavasta riippumatta arvioinnin tavoitteena on arkkitehtuurisuunnitelmassa olevien virheiden ja heikkouksien löytäminen, jotta ne voidaan korjata tai niiden vaikutukset minimoida ennen toteutuksen aloittamista. Arkkitehtuurin arviointi ei ole kertaluonteinen tapahtuma, vaan siihen pitäisi suhtautua jatkuvana prosessina. Arkkitehtuurin arviointi loppuu kun asiakkaalle on toimitettu toimiva järjestelmä tai sovellus. (Rozanski & Woods 2011, 217; Gorton 2006, 109.)

5.5 Arkkitehtuurin kuvaaminen

Arkkitehtuuri edustaa suunnitelmaa järjestelmän tai sovelluksen rakenteesta. Kohteesta riippuen arkkitehtuuri saattaa olla mitä tahansa yksinkertaisen ja hyvin monimuotoisen sekä monitasoisen arkkitehtuurin väliltä. Arkkitehtuuri sisältää kaikki järjestelmän tai sovelluksen merkittävät piirteet ja ominaisuudet kuvattuna käyttötarkoitukseen sopivalla tarkkuudella.

Arkkitehtuuria voi kuvata yleisellä tasolla tuottamalla yhden sivun mittaisen kaavion, jossa esitellään järjestelmän tai sovelluksen pääkomponentit, niiden välinen vuorovaikutus sekä arkkitehtuurin kantavat suunnitteluperiaatteet. Tätä yleistystä kutsutaan englanninkielessä termillä ”Marketecture”, jolla viitataan sen markkinoinnilliseen luonteeseen arkkitehtuurin kuvaajana. Tämä yleistason kuvaus on kaikessa yksinkertaisuudessaan helposti ymmärrettävä ja toimii lähtökohtana syvällisempään määrittelytyöhön. (Gorton 2006, 6 – 7.)

Koko arkkitehtuurin kuvaaminen yksittäisellä kaaviolla ei yleensä ole käytännöllistä. Tällainen tapa saattaa tuottaa epäselvän ja tulkinnanvaraisen esityksen

arkkitehtuurista, jossa kaavion elementit ja niiden väliset yhteydet jäävät tarkoitukseltaan epäselviksi. (Kruchten 1995, 42; Rozanski & Woods 2001, 33). Gortonin (2006, 6 – 7) tarkoittama ”marketecture” on epämuodollinen yleistys arkkitehtuurista sidosryhmien välisen keskustelun apuvälineeksi suunnitteluun, toteutukseen ja katselmointiin – Sen ei ole tarkoitus antaa kattavaa kuvausta arkkitehtuurista.

Esitystavan sekä elementtien ja niiden välisten yhteyksien kuvaamisen ongelmaa voidaan lähestyä toisella tavalla. Kuten kaikkia monimuotoisia ja monitasoisia asioita, myös arkkitehtuuria voidaan katsoa ja määritellä useista näkökulmista ja useilla tarkkuuksilla. Sen sijaan, että arkkitehtuuria yritetään kuvata yhdellä monimutkaisella kaaviolla, voidaan kuvaamisessa hyödyntää näkökulmia ja tarkkuustasoja, joita hyödyntäen tuotetaan useita käyttötarkoitukseltaan selkeitä kaavioita. Jokainen näkökulma ja tarkkuustaso esittävät arkkitehtuurin jonkin järjestelmään liittyvän sidosryhmän edun mukaisesti. (Gorton 2006, 7; Kruchten 1995, 42 – 43.)

5.5.1 Kuvaamisen näkökulmat

Näkökulmia järjestelmään tai sovellukseen kutsutaan näkymiksi (view). Arkkitehtuurin mallintamisen näkymien avulla esitteli vuonna 1995 Philippe Kruchten julkaistessaan arkkitehtuurin kuvaamista ohjaavan 4+1 mallin. Kruchtenin malli esitteli tavan kuvata ja ymmärtää arkkitehtuuria seuraavien näkymien mukaisesti:

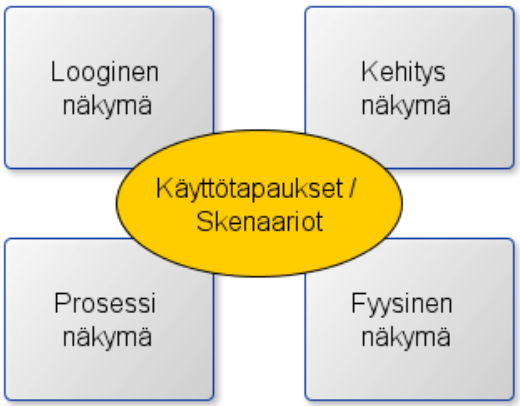
- looginen näkymä (Logical View)
- prosessi näkymä (Process View)
- fyysinen näkymä (Physical View)
- kehitysnäkymä (Development View)

(Kruchten 1995, 42 – 43)

Looginen näkymä (Logical View) kuvaa arkkitehtuurisesti merkittävät elementit vuorovaikutussuhteineen. Se sisältää siis loppukäyttäjien kannalta merkitykselliset elementit ja palvelut. Prosessinäkymän (Process View) tarkoituksena on

kuvata järjestelmän toimintaa loogisten prosessien ja niiden keskinäisen kommunikoinnin kautta. Prosessinäkymässä kuvataan myös sovelluksen säikeistykseen, replikointiin sekä kontrollin siirtymiseen liittyviä piirteitä. Fyysinen näkymä (Physical View) kuvaa arkkitehtuurin pääelementtien ja -prosessien sijoittumisen fyysisille laitteille ja kuvastaa samalla sovelluksen jaettua rakennetta eri laitteiden välillä. Kehitysnäkymä (Development View) kuvaa sovelluksen jakamista kehityksen kannalta merkityksellisiin elementteihin. Kehitysnäkymän perusteella tiedetään, mitä sisäisiä rakenteita ja kirjastoja voidaan käyttää. Lisäksi sen perusteella voidaan kehittäminen jakaa useiden kehittäjien kesken. (Gorton 2006, 8 – 9; Kruchten 1995, 43 – 47.)

Kruchtenin mallin nimessä oleva +1 muodostuu viidennestä näkymästä, joka kuvaa arkkitehtuurisesti merkittävät käyttötapaukset tai skenaariot. Ne perustuvat arkkitehtuurille asetettuihin vaatimuksiin, koskevat yleensä useita arkkitehtuurin kuvaamiseen käytettyjä näkymiä ja näin ollen sitovat näkymien esittämät tiedot toisiinsa. Käyttötapauksia ja skenaarioita prosessoimalla voidaan kuvattua arkkitehtuuria testata. Testaaminen tapahtuu kuvailemalla, kuinka arkkitehtuuri reagoi kussakin käyttötapauksessa tai skenaariossa asetettuihin vaatimuksiin. Kruchtenin 4+1 mallin kaikki viisi näkymään voidaan esittää myös kuvion 14 mukaisesti.



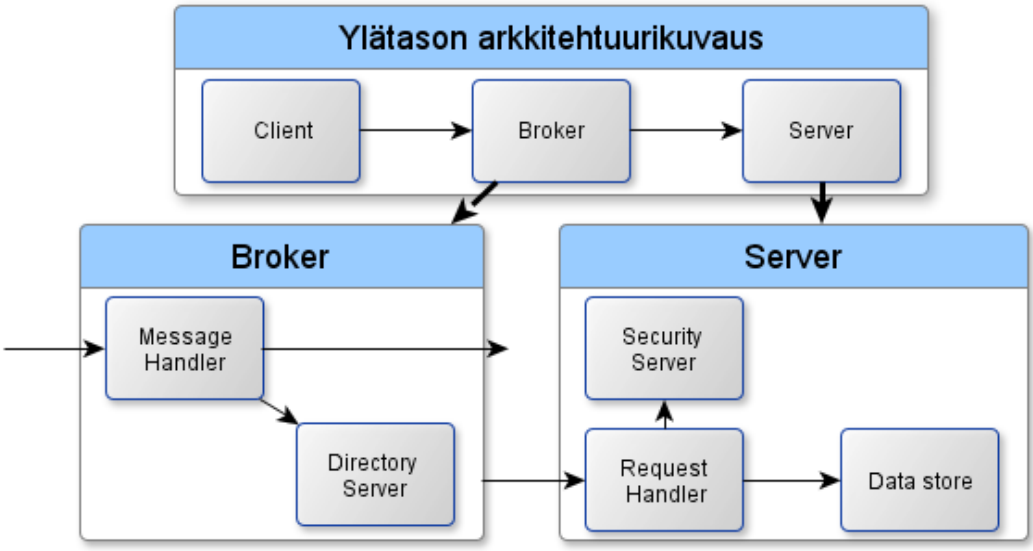
KUVIO 14. 4+1 arkkitehtuurimalli (mukaillen Kruchten 1995, 43)

5.5.2 Kuvaamisen tarkkuus

Näkymiin jakamisen lisäksi arkkitehtuuria voidaan kuvata useilla tasoilla tai tarkkuuksilla. Sovelluksen tai järjestelmän sidosryhmillä on erilaisia tarpeita nähdä tietoja arkkitehtuurista. Näin ollen myös kuvaamisen taso ja tarkkuus riippuu kohdejoukosta, jolle kyseinen arkkitehtuurikuvaus tai näkymä on tarkoitettu. (Gorton 2006, 6 – 7.)

Kuvaamisen tason tarkentamiseen voidaan arkkitehtuurikuvauksessa käyttää hierarkkista erittelyä (hierarchical decomposition). Menetelmässä yhdellä arkkitehtuurin tasolla esiintyvä komponentti eritellään pienempiin osiin siirryttäessä kuvaushierarkiassa tarkemmalle kuvauksen tasolle. Menetelmää käytetään yleensä tarkentamaan arkkitehtuurin ja lopullisen sovelluksen toiminnan kannalta merkittäviä komponentteja. (Gorton 2006, 6 – 7.)

Kuviossa 15 esitetään yksinkertainen malli siitä, kuinka komponentin kuvaaminen muuttuu kuvauksen tason tarkentuessa. Kaaviossa eritellään ylätason arkkitehtuurissa esiintyvät Broker ja Server -komponentit tarkempiin osiin siirryttäessä kuvaamaan asioita tarkemmalla tasolla. Kaikkia komponentteja ei kuitenkaan ole tarpeen kuvata tarkemmalla tasolla. Näin tapahtuu yleensä silloin, jos komponentin sisäinen rakenne tai sisäiset toiminnot eivät vaikuta eitoiminnallisten vaatimusten saavuttamiseen. (Gorton 2006, 7 – 8.)



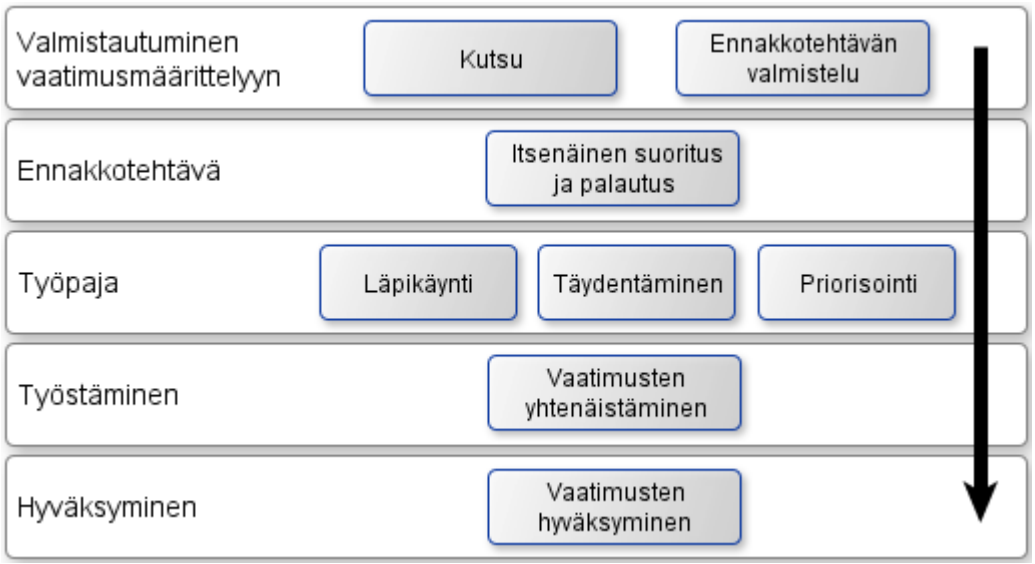
KUVIO 15. Komponenttien hierarkkinen erittely (mukaillen Gorton 2006, 8)

6 VAATIMUKSET ENTERPRISE WEB SERVICES -ARKKITEHTUURILLE

Enterprise Web Services -rajapinnalle ja sen arkkitehtuurille asetettiin sekä toiminnallisia että ei-toiminnallisia vaatimuksia. Vaatimukset määrittelevät millainen rajapinnasta halutaan ja mitä ominaisuuksia sen pitää sisältää. Yhtenä osana vaatimusmäärittelyä oli myös rajoituksien asettaminen rajapinnalle ja sen sisältämälle toiminnallisuudelle.

Vaatimusmäärittelyyn osallistuivat Enterprise ERP:n tuotekehityksestä vastaavat henkilöt, kehittämiseen osallistuvat järjestelmäarkkitehdit sekä muutama erikseen valittu asiantuntija. Kaikki vaatimusmäärittelyihin osallistuneet henkilöt tunsivat Enterprise ERP -tuotetta ja siihen liittyvää liiketoimintaa tai vaihtoehtoisesti web service –teknologioita ja sovellusaluetta sekä kehitys- ja toimitusprojekteja, joiden käyttöön rajapinta valmistuttuaan tulisi. Kokoonpanolla pyrittiin varmistamaan, että näkemyksensä pääsevät esittämään sekä tuotekehitykseen että projektitoimintaan osallistuvat henkilöt ja näin takaamaan tulevan toteutuksen sopivuus sekä tuotteen että asiakasprojektien kannalta.

Vaatimusmäärittely haluttiin pitää hyvin kevyenä ja osallistujia mahdollisimman vähän kuormittavana prosessina. Näin ollen valmisteluihin kiinnitettiin erityisesti huomiota, jotta osallistujat voisivat keskittyä olennaiseen, eli vaatimusten tuottamiseen. Kuviossa 16 esitetään vaatimusten keräämisessä käytetty prosessi.



KUVIO 16. Arkkitehtuurivaatimusten keräämisprosessi

Vaatimusmäärittely aloitettiin lähettämällä kaikille määrittelyihin osallistuville kutsu ja ennakkotehtävä. Kutsussa määritettiin varsinaisen vaatimusmäärittelytyöpajan ajankohta sekä annettiin ohjeet vaatimusmäärittelystä. Ennakkotehtävä käsitteli vaatimusten listausta ennakkoon tehdyn vaatimusrungon perusteella. Vaatimusrunko oli yksinkertainen sisältäen taulukon Gortonin esittämistä (2006, 24 – 37) ei-toiminnallisten vaatimusten osa-alueista täydennettynä yhdellä toiminnallisia vaatimuksia kuvaavalla osa-alueella. Jokaisen osallistujan piti tehdä ennakkotehtävä itsenäisesti ja palauttaa se viikkoa ennen varsinaista työpajaa.

Ennakkotehtävien palautusten perusteella koottiin runko työpajalle. Ennakkotehtävän palautuksina saadut yksittäiset vaatimukset ja rajaukset käsiteltiin ja koottiin yhteen taulukkoon. Käsittelyn yhteydessä eri henkilöiltä tulleita samoja vaatimuksia yhdistettiin päällekkäisyyksien poistamiseksi. Näin luotu taulukko toimi työpajan työjärjestyksenä ja runkona.

Työpajassa jokainen ennakkoon saatu vaatimus ja rajaus esiteltiin ehdottajansa toimesta. Esittelyn jälkeen vaatimusta tai rajausta käsiteltiin siten, että jokainen sai esittää mielipiteensä sen tarpeellisuudesta sekä prioriteetista. Mahdollisista ristiriitaisista vaatimuksista ja rajauksista keskusteltiin ja niiden kohdalle pyrittiin löytämään selkeä ratkaisu tai prioriteetti, jonka perusteella ristiriita on mahdollista ratkaista. Käsittelyn yhteydessä vaatimus- ja rajaustaulukkoa tarkennettiin aiempien tietojen osalta sekä täydennettiin uusien tietojen osalta.

Työpajan lopputuloksena syntynyt täydennetty vaatimus- ja rajaustaulukko käytiin lävitse ja siihen kirjattujen tietojen esitystapa yhtenäistettiin. Yhtenäistäminen tapahtui JHS-173 -suositusten (2009, 20) mukaisesti noudattaen hyvän vaatimusilmaisun kriteerejä. Suurin haaste vaatimusten määrittelyissä oli se, että vaatimus ei itsessään saisi ilmaista ratkaisua. Vaatimusmäärittelyn lopputulos esitettiin työryhmälle ja hyväksyttiin määrittelytyön perustaksi.

6.1 Toiminnalliset vaatimukset

Toiminnalliset vaatimukset (functional requirements) määrittelevät järjestelmän käyttäytymistä tai toiminnallisuutta. Toiminnallisilla vaatimuksilla kuvataan järjestelmän tarjoamat palvelut, järjestelmän reagointi syötteisiin sekä käyttäytymistä annetuissa ja kuvatuissa tilanteissa (JHS-173 2009, 6). Toiminnalliset vaatimukset määrittelevät siis sitä, mitä järjestelmän täytyy tehdä. Yksittäinen vaatimus määrittää yhden rajatun toiminnon, joka järjestelmän täytyy toteuttaa.

Web service -rajapinnan toiminnalliset vaatimukset ovat melko yksinkertaiset. Tämä johtuu siitä, että rajapinta on mahdollistaja, jonka avulla varsinaisia palveluita voidaan toteuttaa, eikä niinkään tiettyyn ongelman tai käyttötarkoituksen ratkaiseva sovellus. Esimerkiksi vaatimus tuetuista sanomatyypeistä esitetään vain yleisesti formaattien tasolla (xml, json), eikä ole lähdetty vaatimaan minkään tietyn standardin, vaikka SOAP:n, tukea. Enterprise Web ces -rajapinnalle asetut toiminnalliset vaatimukset kohdistuivat lähinnä tietojenvälittämiseen, sanomatyyppeihin, kehittäjien välineisiin sekä testauk-seen.

6.2 Ei-toiminnalliset vaatimukset

Ei-toiminnalliset vaatimukset (non-functional requirements) kuvaavat kuinka järjestelmälle asetetut toiminnalliset vaatimukset on tarkoitus saavuttaa (Gorton 2006, 23). Käytännössä ei-toiminnalliset vaatimukset tarkoittavat niitä rajoituksia ja reunaehtoja, jotka järjestelmän on täytettävä, jotta toiminnalliset vaatimukset voidaan yleensä toteuttaa (JHS-173 2009, 4).

Gortonin (2006, 6) mukaan arkkitehtuuri perustuu ei-toiminnallisille vaatimuksille. Ei-toiminnalliset vaatimukset käsitelevät juuri niitä järjestelmiin ja sovellukseen liittyviä osa-alueita, joiden perusteella arkkitehtuuri on mahdollista suunnitella (Gorton 2006, 6).

Ei-toiminnalliset vaatimukset täytyy määritellä konkreettisella tasolla, jotta niihin voidaan arkkitehtuurisuunnittelussa ottaa kantaa ja esittää ratkaisu. Liian yleiselle tasolle jääviä ja epätarkkoja vaatimuksia on hankala käsitellä ja ratkaista.

Tämä johtuu siitä, että vaatimukseen saattaa liittyä useita eri tavoin ratkaistavia osakokonaisuuksia. Esimerkiksi vaatimus ”järjestelmän täytyy olla skaalautuva” on liian epätarkka, jotta se voitaisiin ratkaista yksiselitteisesti. Tässä tapauksessa epätarkkuus johtuu siitä, että skaalautuvuus voidaan jakaa usealla eri tavalla osakokonaisuuksiin, jotka kohdistuvat sovelluksen eri osa-alueisiin ja jotka voidaan ratkaista eri tavoilla. (Gorton 2006, 23.)

Ei-toiminnallisia vaatimuksia käsiteltiin vaatimusmäärittelyn yhteydessä Gortonin (2006, 24 - 37) esittämän jaottelun mukaisesti. Jaottelussa ei-toiminnallisista vaatimuksista on tunnistettu seuraavat pääryhmät:

- suorituskyky (performance)
- skaalautuvuus (scalability)
- muunneltavuus (modifiability)
- tietoturva (security)
- saavutettavuus (availability)
- muut ei-toiminnalliset vaatimukset

6.2.1 Suorituskyky

Suorituskykyyn liittyvä vaatimus tai rajaus määrittelee järjestelmälle mittarin, jonka mukaan sen on suoriuduttava tietystä työkuormasta annetun aikaikkunan sisällä tai tiettyyn aikarajaan mennessä. Suorituskykyä mitataan usein suoritustehon (thoughput), vasteajan (response time) sekä määräaikojen (deadlines) perusteella. (Gorton 2006, 24 – 26.)

Suoritusteholla tarkoitetaan suoritusten määrää tietyn aikaikkunan sisällä, esimerkiksi transaktioita sekunnissa tai käsiteltyjä viestejä sekunnissa. Suoritustehon osalta on tärkeää ymmärtää keskimääräisen suoritustehon ja huippusuoritustehon välinen ero. (Gorton 2006, 25.)

Vasteajalla tarkoitetaan sitä aikaa, joka kuluu kutsutun palvelun suorittamiseen. Käytännössä vasteaika on aika, joka palvelulta kestää suorittaa ohjelma ja palauttaa suorituksen lopputulos kutsujalle. Vasteajankin osalta on tärkeää ym-

märtää ero keskimääräisen vasteajan ja taatun vasteajan välillä. (Gorton 2006, 25 – 26.)

Enterprise Web Services -rajapinnan osalta on kutsuttavalla palvelulla ja sen toteutuksella huomattava vaikutus sekä suoritustehoon että vasteaikaan. Näin ollen ne eivät ole kovin hyviä vaatimuksia varsinaiselle rajapinnalle. Rajapinnan arkkitehtuuri täytyy kuitenkin suunnitella siten, että rajapinta itsessään aiheuttaa kuormitusta tai viivettä (latenssia) vasteaikaan mahdollisimman vähän.

Määräajalla tarkoitetaan vaatimusta, jonka mukaan sovelluksen täytyy suorittaa jokin toimenpide tiettyyn määräaikaan mennessä (Gorton 2006, 26). Määräaikoihin liittyviä suorituskyvyllisiä vaatimuksia ei Enterprise Web ces -rajapinnalle vaatimusmäärittelyjen yhteydessä tunnistettu.

6.2.2 Skaalautuvuus

Skaalautuvuus tarkoittaa järjestelmän tai sovelluksen arkkitehtuurin selviytymistä tilanteessa, jossa jokin sen osa-alue kasvaa koossa suuremmaksi. Käytännössä tämä tarkoittaa sitä, kuinka jonkin ongelman ratkaisu toimii tilanteessa, jossa itse ongelma kasvaa aiempaa suuremmaksi. Skaalautuvuutta mitataan ja arvioidaan usein kuormituksen määrän (Request load), yhtäaikaisten yhteyksien määrän (Simultaneous connections) sekä datan koon (Data size) perusteella. (Gorton 2006, 27 – 28.)

Arkkitehtuuri suunnitellaan yleensä tukemaan tiettyä kuormitusta. Kuormitusta voidaan ajatella suoritustehon ja vasteajan yhdistelmänä, siten että järjestelmä tukee esimerkiksi sataa transaktiota sekunnissa keskimäärin sekunnin vasteajalla. Kuormituksen merkittävä kasvaminen vaikuttaa järjestelmän toimintaan yleensä negatiivisesti siten, että suoritusteho pienenee ja vasteaika kasvaa. (Gorton 2006, 28.)

Kuormituksen huomioimisen lisäksi järjestelmän arkkitehtuuri suunnitellaan yleensä tukemaan myös tiettyä yhtäaikaisten käyttäjien määrää. Jos yksittäisen käyttäjän yhteys kuluttaa resursseja, niin todennäköisesti yhtäaikaisten yhteyk-

sien määrälle on olemassa sekä laskennallinen maksimi että tätä pienempi käytännön maksimi. (Gorton 2006, 29 – 30.)

Käsiteltävän datan koon kasvamisen vaikutus järjestelmään on myös merkityksellistä. Datan koon kasvaminen, joko käsiteltävien sanomien koon kasvamisena tai taustalla olevan tietovaraston, esimerkiksi tietokannan, tiedon määrän kasvamisena, vaikuttaa yleensä järjestelmän suoritustehoon ja vasteaikaan heikentävästi. (Gorton 2006, 30 – 31.)

Järjestelmän skaalautuvuuden ymmärtämystä auttaa huomattavasti, jos pystyy vastaamaan seuraaviin kysymyksiin: Kuinka järjestelmä reagoi, jos sen kuormitus kasvaa huomattavasti? Kuinka järjestelmä reagoi, jos yhtäaikaisten yhteyksien määrä kasvaa huomattavasti? Kuinka sovellus reagoi, jos sen käsittelemän datan koko kasvaa huomattavasti? (Gorton 2006, 27 – 31.)

Enterprise Web Services –rajapinnan teknisillä puitteilla ja niiden mahdollisuuksilla on merkittävä rooli rajapinnan skaalautumisessa erilaisiin tilanteisiin. Rajapinnalle on asetettu vaatimus siitä, että sen on pystyttävä skaalautumaan sekä hyödyntämällä yhden laitteen kaikki käytettävissä olevat resurssit (myös lisätyt resurssit, kuten muisti, levytila, nopeammat prosessorit) sekä hyödyntämällä useita palvelimia.

6.2.3 Muunneltavuus

Sovellukset muuttuvat elinkaarensa aikana yleensä useitakin kertoja. Tämän tosiasian huomioiminen arkkitehtuurisuunnittelussa on yleensä välttämättömyys, jolla pienennetään jatkossa muutoksista tulevia riskejä ja niistä aiheutuvia kustannuksia. (Gorton 2006, 31 – 32.)

Muunneltavuuden huomioimisessa on tärkeää se, että voidaan määritellä ja arvioida mahdolliset muutoksia vaativat kohteet etukäteen. Tämä on yleensä hyvin haastavaa koska muutoksien määrittämiseen tarvittavia tietoja ei yleensä ole etukäteen saatavilla. Arviointi rajoittuukin usein muutoksia tarvitsevien kom-

ponenttien listaamiseen, muutosten tarvitseman työmäärän arviointiin ja näiden perusteella tehtävään kustannusarvioon. (Gorton 2006, 32.)

Enterprise Web Services –rajapinnan osalta vaatimusten asettelu on lähtenyt siitä, että suunnittelussa ja toteutuksessa täytyy varautua uusien sisältötyyppien (content-type) tukemiseen sekä toiminnanohjausjärjestelmän palveluiden uudistamiseen.

6.2.4 Tietoturva

Tietoturva on monimutkainen tekninen aihealue, joka arkkitehtuurisuunnittelussa pyritään huomioimaan sovelluksen tarpeina ja vaatimuksina tietoturvalle. Yleisesti tietoturvaa käsitellään arkkitehtuurisuunnittelussa autentikoinnin (authentication), valtuutuksen (authorization), salauksen (encryption), eheyden (integrity) sekä kiistämättömyyden (non-repudiation) kautta. (Gorton 2006, 33.)

Autentikoinnilla tarkoitetaan järjestelmää käyttävien käyttäjien ja järjestelmien tunnistamista. Valtuutuksella taas tarkoitetaan tunnistetun käyttäjän oikeuksia järjestelmän tietosisältöön ja toimintoihin. Salauksella tarkoitetaan käsiteltävien pyyntöjen ja viestien salaamista. Eheydellä tarkoitetaan sitä, että viestien sisältöä ei ole muutettu siirron aikana ja kiistämättömyydellä sitä että viestinnän molemmat osapuolet voivat olla varmoja toistensa identiteetistä. (Gorton 2006, 33 – 34.)

Tietoturva oli erittäin tärkeässä roolissa vaatimusmäärittelyssä. Enterprise Web Services –rajapinnalle on asetettu vaatimuksena että rajapinnan on tuettava oletuksena autentikointia, mutta mahdollistettava myös kaikille avoimet palvelut. Rajapinnan on tuettava myös Enterprise toiminnanohjausjärjestelmän valtuutustoimintoja (käyttäjän oikeudet tietosisältöön ja toimintoihin). Lisäksi viestiliikenne on pystyttävä salaamaan. Eheyden ja kiistämättömyyden osalta ei Enterprise Web Services –rajapinnalle asetettu vaatimuksia.

6.2.5 Saavutettavuus

Järjestelmän saavutettavuuden vaatimus on kohtuullisen helposti määriteltävissä ja mitattavissa. Saavutettavuudella tarkoitetaan sitä yhteenlaskettua aikaa jona järjestelmää voidaan käyttää tietyn seuranta-ajanjakson kuluessa. Toisinpäin käännettynä voidaan määritellä aika, jonka järjestelmä saa olla pois käytöstä tiettynä ajanjaksona ilman että siitä aiheutuu varsinaista haittaa. (Gorton 2006, 34 – 35.)

Saavutettavuus määritellään yleensä prosentteina aikayksiköstä. Tällöin esimerkiksi 90% saavutettavuus tarkoittaa, että järjestelmä saa olla pois käytöstä 36,5 päivää vuodessa, 72 tuntia kuukaudessa tai 16,8 tuntia viikossa. Kaikkia sovelluksia ei tarvitse käyttää vuorokauden ympäri, joten saavutettavuuden vaatimuksetkin vaihtelevat asiakastarpeen mukaan huomattavasti.

Enterprise Web Services –rajapinnan osalta varsinaiset palvelut sekä asiakastarve ovat tärkeitä saavutettavuuteen liittyen. Rajapinnan on kuitenkin pystyttävä tarjoamaan palveluita asiakastarpeesta riippuen myös korkean saavutettavuuden ympäristöissä. Tämä vaatimusmääritys jäi kovin yleiselle tasolle, eikä sille määritetty tarkkoja rajoja.

6.2.6 Muut ei-toiminnalliset vaatimukset

Muita ei-toiminnallisia vaatimuksia ovat sovelluksen integroitavuuteen (integration), siirrettävyyteen (portability), testattavuuteen (testability) sekä tuettavuuteen (supportability) liittyvät vaatimukset. (Gorton 2006, 35 – 37.)

Integroitavuudella tarkoitetaan sitä, kuinka helposti järjestelmä on liitettävissä osaksi suurempaa sovellus- tai järjestelmäkokonaisuutta. Sovelluksen tai komponentin arvo on yleensä sitä suurempi, mitä useammilla tavoilla sitä voidaan hyödyntää (Gorton 2006, 35). Enterprise Web Services –rajapinta tulee toteuttamaan juuri tämän vaatimuksen Enterprise toiminnanohjausjärjestelmän kannalta. Rajapinta mahdollistaa järjestelmän liittämisen osaksi suurempaa koko-

naisuutta tai muiden sovellusten ja päätelaitteiden integroimisen toiminnanohjausjärjestelmään suuremman kokonaisuuden luomiseksi.

Järjestelmän siirrettävyyteen liittyvät vaatimukset asettavat ehtoja siitä, millaisessa sovellus- ja laiteympäristöissä sovellusta pitää pystyä ajamaan. Siirrettävyys itsessään tarkoittaa sitä, kuinka helposti järjestelmä voidaan siirtää ajettavaksi johonkin muuhun sovellus- tai laiteympäristöön (Gorton 2006, 37). Enterprise Web Services –rajapinnan osalta asetettiin vaatimus, että sitä pitää pystyä ajamaan sovellusalustan (Progress Openedge) tukemissa laiteympäristöissä.

Testattavuudella tarkoitetaan sitä, kuinka helppoa tai vaikeaa sovelluksen testaaminen on. Sovelluksen monimutkaisuus vaikuttaa suoraan myös testauksen vaikeuteen (Gorton 2006, 37). Enterprise Web Services -rajapinnan osalta asetettiin vaatimukseksi, että itse rajapinta sekä jokainen yksittäinen palvelu pitää pystyä testaamaan sekä toiminnan että kuormituksen keston osalta.

Järjestelmän tuettavuus tarkoittaa sitä, kuinka helppoa sovelluksen tukeminen on käyttöönoton jälkeen. Tukemisella tarkoitetaan yleensä järjestelmän toiminnan diagnosointia sekä ongelmantilanteiden selvitystä. Enterprise Web Services -rajapinnan osalta asetettiin vaatimukseksi, että sen on pystyttävä kirjaamaan loki-tietoja virheistä ja tapahtumista, eli palvelukutsuista.

6.3 Rajoitteet

Rajoitteet ja reunaehdot ovat vaatimuksia, jotka selkeästi ja tarkoituksella rajoittavat järjestelmän suunnittelua, toteutusta, käyttöä, elinkaarta tai päätöksentekoa (JHS-173 2009, 5). Rajoitteet voidaan jakaa teknisiin ja liiketoiminnallisiin rajoitteisiin ja niitä voidaan pitää osana ei-toiminnallisia vaatimuksia (Gorton 2006, 6).

Tekniset rajoitteet ovat melko yleisiä ja ne asettavat reunaehtoja niille teknologioille joita sovelluksen toteuttamisessa voidaan käyttää. Tekniset rajoitteet ovat yleensä kiinteitä, eikä niitä voida kovinkaan helposti kiertää (Gorton 2006, 6).

Enterprise Web Services –rajapinnan osalta teknisiä rajoitteita asettavat Enterprise toiminnanohjausjärjestelmä, toiminnanohjausjärjestelmän tekemiseen käytetty sovelluskehitin (Progress Openedge) sekä tuotekehitys- ja toimitusprojekteissa toimivien henkilöiden osaaminen. Tämä tarkoittaa sitä, että rajapinta on toteutettava Progress Openedge tuoteperheen välineillä.

Liiketoiminnalliset rajoitteet koskevat yleensä resurssien käyttämistä, riippuvuuksia ja aikataulua liittyen sovelluksen suunnittelun ja toteuttamiseen. Kehitysprojektille asetettiin selkeät resurssien käyttöä koskevat rajoitteet, jotka koskivat lähinnä käytettävää työmäärää ja tavoiteaikataulua. Taloudellisiin rajoitteisiin kuuluu myös se, että lopputuloksena syntyvää rajapintaa pitää pystyä hyödyntämään nykyisillä toiminnanohjausjärjestelmäprojektissa käytettävissä olevilla resursseilla. Näitä resursseja ovat niin henkilöt, kuin tekniset laitteet ja järjestelmätkin.

Enterprise Web Services -rajapinnalle on asetettu rajoitteeksi myös se, että nyt tehtävään toteutukseen ei lähdetä tekemään erillistä REST-tukea. Progress on ilmoittanut että OpenEdge Webspeed ja Appserver sovellusalustoihin tulee natiivi REST-tuki version 11 aikana. Tarkkaa aikataulua REST-tuelle ei ole julkistettu, mutta ensimmäisessä 1.1.2012 tapahtuneessa julkaisussa REST-tukea ei ole.

Tämän lisäksi on päätetty, että rajapintaan ei rakenneta SOAP-pohjaisten XML-sanomien tukea. SOAP-pohjaiset integraatiot tullaan jatkossakin toteuttamaan perinteisillä integraatiovälineillä.

7 ENTERPRISE WEB SERVICES -ARKKITEHTUURIN SUUNNITTELU

Arkkitehtuurisuunnittelun lähtökohtana olivat päätetyt toiminnalliset ja ei-toiminnalliset vaatimukset. Lisäksi suunnittelussa pyrittiin ottamaan huomioon mahdollisimman hyvin sekä tekniset että taloudelliset rajoitteet, kuten vaatimus sille että rajapinta täytyy olla hyödynnettävissä nykyisin käytössä olevin resurssein.

Arkkitehtuurisuunnittelussa olisi myös huomioitava tarve tuottaa sellainen toiminta malli ja tekninen ratkaisu, joka ei vaadi, että kaikkien tekijöiden tulisi ymmärtää web service -ratkaisuja teknologiselta kannalta. Yksinkertaistettuna ohjelmoitavan palvelun tuottamiseksi pitäisi riittää se, että tekijä ymmärtää Progress ABL kielen tietorakenteita.

Suunnittelussa pitäisi huomioida myös jo nyt nähtävissä olevia jatkokehitystarpeita. Esimerkiksi arkkitehtuurin pitää tukea ja sen pohjalta tehtävää fyysistä konstruktiota pitää pystyä myöhemmässä vaiheessa kehittämään siten, että rajapinnan yhteyteen voidaan tuottaa palveluita myös konfiguroimalla. Konfiguroitavuus merkitsee tässä tapauksessa sitä, että uusi palvelu voitaisiin luoda pelkästään toiminnanohjausjärjestelmän käyttöliittymällä valitsemalla tiedot, joita näytetään ulospäin ja tiedot, joiden perusteella näytettäviä tietoja rajataan. Palveluiden konfiguroitavuus toisi huomattavaa etua uusien palveluiden tuottamisessa.

7.1 Prototyyppi

Progress OpenEdgen tarjoamien mahdollisuuksien selvittämiseksi ja mallin sopivuuden kokeilemiseksi toteutettiin ennen varsinaisen suunnittelun aloittamista yksinkertainen toteutettavuusdemo (proof of concept). Toteutettavuusdemo koostui Progress OpenEdgen suosituksen mukaisesta teknisestä ympäristöstä, johon toteutettiin ohjelmoimalla muutama yksinkertainen SOAP-RPC palvelu.

Toteutettavuusdemon tekninen ympäristö muodostettiin kuvion 3. mukaiseksi sillä poikkeuksella että ympäristö koostui yhdestä fyysisestä työ-asemasta. Työasemalla sijaitsi asiakasta (client) lukuun ottamatta kaikki ympäristöön liittyvät komponentit, kuten toiminnanohjausjärjestelmän tietokannat, Appserver-sovelluspalvelin sekä edustapalvelimena toimiva Apache Tomcat -www-palvelin. Asiakkaana toimi http-protokollalla SOAP-kutsuja lähettämään pystyvä SoapUI testiohjelmisto.

Ympäristöön toteutettiin ohjelmoimalla muutama palvelu. Palveluina toimivat erilaiset yksinkertaiset listaukset toiminnanohjausjärjestelmän tiedoista. Palveluita toteuttamalla saatiin testattua myös palveluiden julkaisemiseksi tarvittavat toimet. Yksittäisen palvelun toteuttamiseksi ja julkaisemiseksi pitää suorittaa seuraavat vaiheet:

- palvelua vastaavan ohjelman tai ohjelmien toteuttaminen ohjelmoimalla
- ohjelmien kääntäminen
- web service –palvelun generointi käännetyistä ohjelmista OpenEdge Open Client Proxy Generator (ProxyGen) –apuohjelmalla
- käännettyjen ohjelmien siirto sovelluspalvelimelle
- generoinnissa syntyneen web service –palvelun konfigurointi ja julkaiseminen edustapalvelimella sijaitsevalle Web Service Adapterille.

Lähes kaikki suositellun Progress OpenEdge mallin mukaiset palvelut täytyy ensin ohjelmoida ja sen jälkeen suorittaa melko monimutkainen ja monivaiheinen julkaisuprosessi, ennen kuin palvelu on valmis käytettäväksi. Sinällään mallin mukaan ohjelmointi on jokseenkin suoraviivaista, mutta se edellyttää tekijältään teknistä osaamista xml-rpc tyyppisestä web service -teknologiasta, soap-formaatista sekä työkaluista, jotka eivät normaalisti kuulu toiminnanohjausjärjestelmän kehittämisen yhteyteen. Malli ei myöskään mahdollista kovin helposti konfiguroitavien palveluiden tuottamista.

Toteutettavuusdemon avulla pystyttiin toteamaan, että Progressin suosittelema malli web service –rajapinnan ja palveluiden toteuttamiseksi ei mahdollista kaikkia niitä vaatimuksia, joita rajapinnalta halutaan. Tärkeimpinä puutteina ja ongelmina havaittiin:

- tuettujen formaattien rajoittuminen ainoastaan xml/soap-sanomiin
- palvelun toteutuksen ja julkaisun monimutkaisuus
- konfiguroitavien palveluiden järkevän tuottamismahdollisuuden puuttuminen

Näin ollen toteutettavuusdemon perusteella päädyttiin siihen lopputulokseen, että Progress OpenEdge sovelluskehitysalustan mukainen tapa suunnitella ja toteuttaa web service –rajapinta ja palveluita on liian jäykkä vastaamaan Enterprise toiminnanohjaussovelluksen yhteydessä esiintyviin tarpeisiin. Tällaisesta lopputuloksesta oli olemassa aavistus etukäteen, mutta se haluttiin varmistaa prototyypin avulla. Näin ollen päädyttiin siihen lopputulokseen, että prototyypillä koestettu malli ei sovellu Enterprise-toiminnanohjausjärjestelmän yhteydessä käytettäväksi.

7.2 Arkkitehtuurisuunnittelu ja viitekehykset

Enterprise web service –rajapinnan suunnitteluun ja toteuttamiseen vaikuttaa kaksi teknistä viitekehystä. Ensimmäinen viitekehys tulee valitusta Progress OpenEdge WebSpeed toteutustekniikasta, toinen tulee Digia se -toiminnanohjausjärjestelmästä. Molemmat viitekehykset asettavat vaatimuksensa ja rajoitteensa rajapinnan arkkitehtuurille ja sen seurauksena myös toteutukselle.

7.2.1 Asiakas-palvelin –arkkitehtuuri

Progress OpenEdge WebSpeed –sovellusalusta noudattaa asiakas-palvelin (client-server) arkkitehtuuria viitekehyksenään. Asiakas-palvelin –arkkitehtuurissa useat asiakkaat (client) ottavat yhteyden keskitettyyn palvelimeen. Palvelin on passiivinen osapuoli, joka jakaa resursseja ja palveluja asiakkaille yleensä tietoverkon ylitse. Asiakkaat ovat aktiivisia toimijoita, jotka ovat aina yhteyden aloittava osapuoli. Asiakkaat eivät jaa omia resurssejaan, vaan hyödyntävät

palvelimen resursseja ja palveluja. Enterprise Web Services –rajapinta toteuttaa teknisesti palvelimen roolin asiakas-palvelin arkkitehtuurissa.

7.2.2 Kolmitasoarkkitehtuuri

Enterprise toiminnanohjausjärjestelmä noudattaa viitekehyksenään kolmitasoarkkitehtuuria. Tämä tarkoittaa sitä, että sovellus on jaettu loogisiin kokonaisuuksiin, eli tasoihin, erottelemalla eri intressejä edustavat kerrokset toisistaan selkeillä rajapinnoilla. Tasoiksi eroteltavia sovelluksen intressejä ovat:

- esitystapakerros, eli sovelluksen käyttöliittymä
- logiikkakerros, eli sovelluksen liiketoimintalogiikka
- datakerros, eli sovelluksen tietovarastot

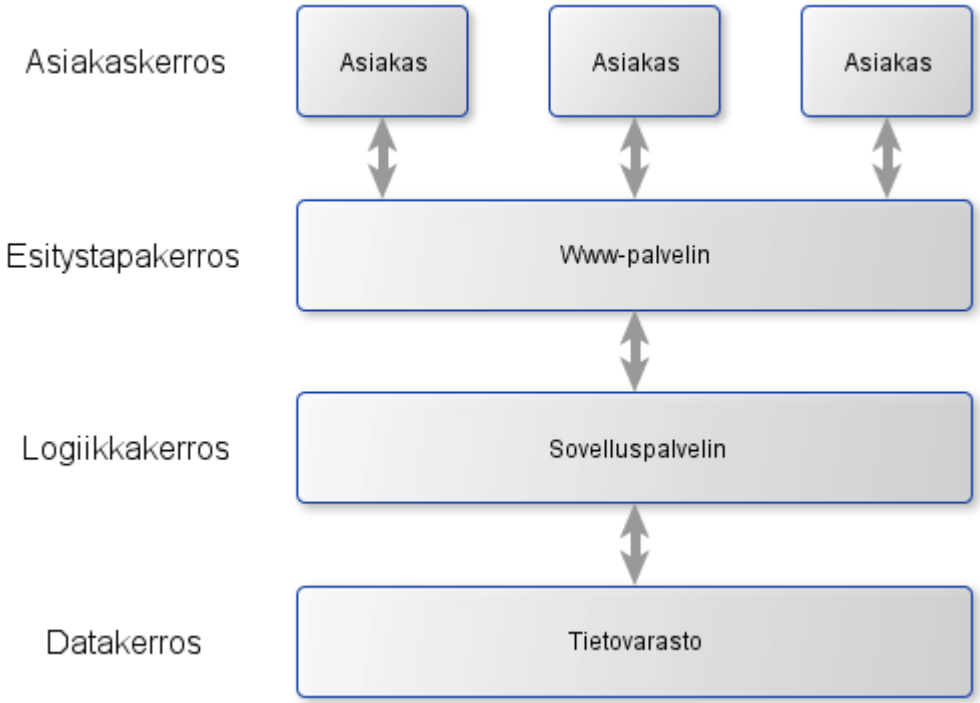
Enterprise Web Services –rajapinta korvaa Enterprise toiminnanohjausjärjestelmän koko esitystapakerroksen. Rajapinta muodostaa ulkopuolisille järjestelmille ja päätelaitteille konekielisen käyttöliittymän toiminnanohjausjärjestelmään.

7.2.3 Monitasoarkkitehtuuri

Teknisen alustan ja toiminnanohjausjärjestelmän viitekehykset huomioiden on luontevaa, että Enterprise Web Services –rajapinta käyttää viitekehyksenä monitasoarkkitehtuuria (n-tier architecture). Monitasoarkkitehtuuri sisältää asiakas-palvelin -arkkitehtuurin ja kolmitasoarkkitehtuurin piirteet yhdessä hyödynnettävässä viitekehyksessä. Gortonin (2006, 97 – 98) mukaan monitasoarkkitehtuurin keskeiset ominaisuudet ovat:

- intressien erottaminen (Separation of concerns)
- synkroninen viestintä (Synchronous communication)
- joustava käyttöönotto (Flexible deployment)

Kuviossa 17 esitetään monitasoarkkitehtuuri web-ympäristössä.



KUVIO 17. Monitasoarkkitehtuuri web-ympäristössä (mukaillen Gorton 2006, 97)

Intressien erottaminen toisistaan vastaa täysin kolmitasoarkkitehtuuria, jossa esitystapa-, logiikka- ja datakerros on erotettu toisistaan selkeillä rajapinnoilla. Monitasoarkkitehtuurissa esitystapa-kerrosta vastaa www-palvelin ja www-palvelimen tarjoama ulkoinen rajapinta asiakkaille, logiikkakerrosta vastaa sovelluspalvelin ja datakerrosta tietolähteet, kuten tietokannat.

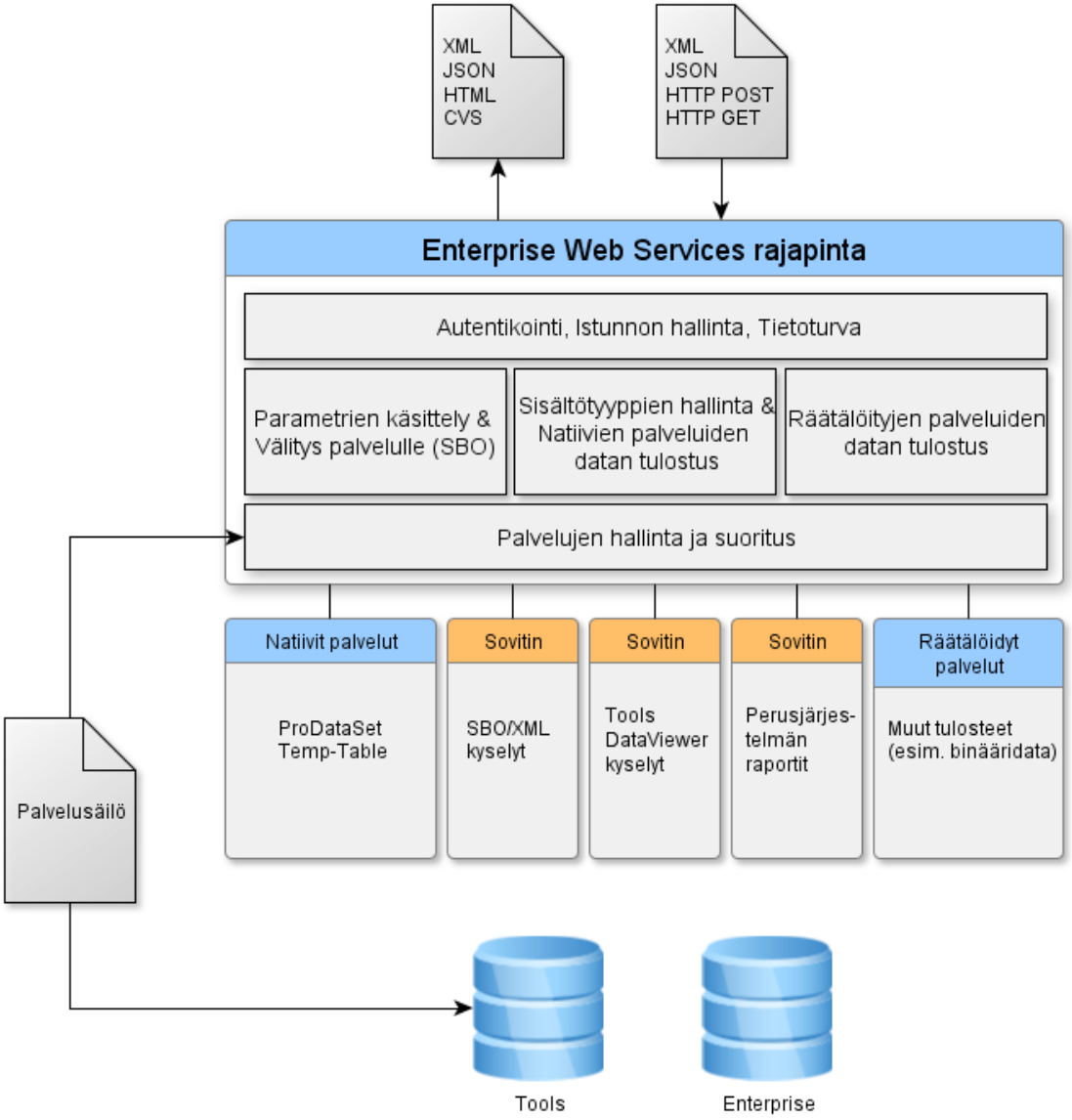
Jokaisella kerroksella on selkeä rajapinta seuraavan kerroksen kanssa. Kerrokset viestivät toistensa kanssa synkronisesti, eli jokaiseen pyyntöön lähetetään vastaus. Pyynnöt etenevät kerroksittain asiakaskerroksesta datakerrokseen asti. Jokainen kerros odottaa vastausta alemmalta kerrokselta ennen vastaamistaan eteenpäin. (Gorton 2006, 98.)

Monitasoarkkitehtuurin käyttöönottoon ja sen eri kerroksien sijoittelulle teknisessä ympäristössä ei periaatteessa ole rajoituksia. Gortonin (2006, 98) mukaan joustava käyttöönotto mahdollistaa sen, että kaikki kerrokset voidaan suorittaa yhdessä laitteessa tai jokainen kerros voidaan sijoittaa omalle laitteelleen.

Asiakaskerros ja sen fyysiset ilmentymät, kuten selaimet tai web ce -asiakkaat, on aina omilla erillisillä laitteillaan.

7.3 Yleisarkkitehtuuri

Tarkemman suunnittelun tueksi luotiin yleisarkkitehtuuri-kaavio. Kaavio kuvaa kaikki web service –rajapintaan liittyvät ylätason käsitteet ottamatta kantaa niiden varsinaiseen sisältöön. Yleisarkkitehtuuria käytettiin keskustelun ja määrittelyn tukena. Yleisarkkitehtuuri on kuvattu kuviossa 18.



KUVIO 18. Enterprise Web Services -yleisarkkitehtuuri

Yhtenä lähtökohtana arkkitehtuurin suunnittelulle oli se, että Enterprisen Web Service -rajapinnan pitää pystyä ottamaan vastaan ja tuottamaan tietoa useissa formaateissa. Tuetut formaatit päätettiin rajapinnan käyttötarpeiden perusteella. Tuettaviksi käyttötarpeiksi tunnistettiin perinteiset integraatiot ulkopuolisiin järjestelmiin ja päätelaitteisiin sekä web-teknologioilla tuotettavat järjestelmät ja sovellukset. Rajapinnan tulee pystyä ottamaan vastaan tietoa seuraavissa formaateissa: XML, JSON, HTTP POST ja HTTP GET. Rajapinnan täytyy myös pystyä tuottamaan tietoa seuraavissa formaateissa: XML, JSON, HTML ja CVS. Tuki useille formaateille mahdollistaa rajapinnan hyödyntämisen useilla erilaisilla tavoilla ja näin ollen maksimoidaan myös potentiaalisten rajapintaa hyödyntävien järjestelmien ja sovellusten määrä.

Tietoturva otettiin huomioon myös arkkitehtuurisuunnittelussa. Jokainen ulos avattu yhteys on tietoturvariski ja mahdollinen kohde hyökkäykselle. Tietoturvalla on näin ollen erityinen merkitys tapauksissa, joissa järjestelmä, sen tietoja ja toimintoja avataan ulkopuolisten käyttöön. Järjestelmää voidaan käytännön tasolla turvata sekä fyysiseen laiteympäristöön ja verkkoon liittyvillä toimilla sekä itse sovellukseen liittyvillä ominaisuuksilla. Yleisessä arkkitehtuurikuvauksessa tarkoitetaan juuri sovelluksen sisäisiä tietoturvaan liittyviä ominaisuuksia.

Enterprise Web Services –rajapinnan on myös pystyttävä käsittelemään ja prosessoimaan sille välitettävää parametritietoa. Palvelukutsun mukana voidaan välittää parametritietoa joko nimi-arvo pareina, kutsun rungossa (body) xml- tai json-merkkijonona tai html-lomakkeena. Välitettävä parametritieto voidaan tapauksesta riippuen tarkoittaa joko web service -rajapinnalle, ajettavalle palvelulle tai molemmille. Esimerkiksi nimi-arvo-parina välitetty session-tunnus on tarkoitettu rajapinnalle, kun taas kutsun mukana tuleva xml- tai json-muotoinen sanoma on yleensä tarkoitettu varsinaiselle palvelulle. Parametritiedon välittämisessä palvelulle käytetään Enterprise-toiminnanohjausjärjestelmän SBO-rajapintaa.

SBO-rajapinta, eli Business Object -rajapinta on Enterprise-toiminnanohjausjärjestelmässä käytettävä rajapinta esitystapakerroksen (käyttöliittymä) ja varsinaisen liiketoimintalogiikkakerroksen välillä. Rajapinta mahdollistaa erilai-

set siirtotavat parametreille ja esitystapakerroksen tietosisällölle käyttöliittymän ja liiketoimintalogiikan välillä. SBO-rajapintaa käytetään erottelemaan sovelluksen logiikassa näytön käsittely varsinaisesta liiketoimintalogiikasta ja tietolähteiden käsittelystä. (Enterprise ohjelmointiohjeet 2012)

Yleinen tapa käsitellä tietotyyppejä on vastata kutsuun samalla tietotyypillä, jolla itse kutsukin on tehty. Tämä tarkoittaa sitä, että jos kutsussa välitetään tietoa xml-formaatissa, myös vastaus on muodoltaan xml-sanoma. Enterprise Web Services –rajapinnan arkkitehtuuri mahdollistaa kuitenkin sekä kutsun että vastauksen tietotyyppien määrittelyn itsenäisesti asiakkaan (client) toimesta, kunhan asetetut tietotyypit ovat rajapinnan tukemia. Kutsu voidaan siis lähettää puhtaana http-kutsuna tai xml-sanomana ja saada vastaus json-muodossa. Itse rajapinta huolehtii sekä natiivien että sovittimen kautta ajettavien palveluiden tuotoksen tulostamisesta vastauksena asiakkaan kutsuun. Räätälöidyt palvelut huolehtivat täysin tulostukseen liittyvistä rutiineista, kuten tietotyypin asettamisesta ja tiedon tulostamisesta tietotyyppiä vastaavassa muodossa.

Rajapinnan natiivit palvelut tuottavat ainoastaan Progress OpenEdgen ProDataSet tai Temp-Table -tietomallin mukaisen data entiteetin, jonka rajapinta käsittelee ja tulostaa halutussa formaatissa asiakkaalle. Rajapinnan ensimmäisessä vaiheessa natiivit palvelut ovat ohjelmoimalla tuotettavia palveluita. Palveluiden avulla voidaan tulostaa toiminnanohjausjärjestelmän tietoa ulkopuolisten käyttöön, tuottaa uutta tietoa toiminnanohjausjärjestelmään tai päivittää toiminnanohjausjärjestelmän nykyistä tietosisältöä. Myöhemmässä vaiheessa tullaan tuottamaan konfiguroitavia palveluita, jotka mahdollistavat tietojen tulostaminen ulkoisten osapuolten käyttöön ilman ohjelmointityötä.

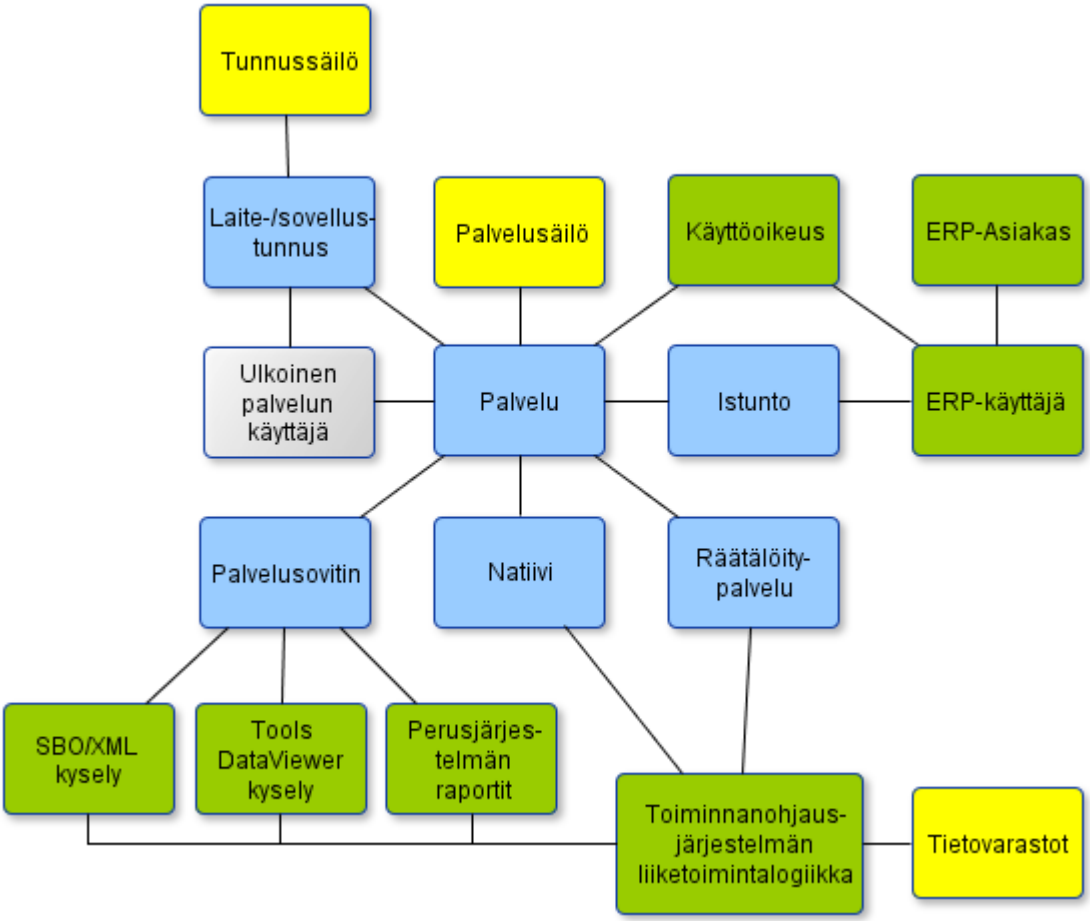
Sovittimen kautta ajettavat palvelut ovat toiminnanohjausjärjestelmän ominaisuuksia, joilla on mahdollista tulostaa dataa tietyssä määrämuodossa. Rajapinta tulostaa sovittimen tuottaman lopputuloksen kyseisen sovittimen tukemassa muodossa. Sovittimien käyttämisen yhteydessä tuetaan ainoastaan vastaavan toiminnanohjausjärjestelmän ominaisuuden tukemia formaatteja. Sovittimet ovat pääasiassa tapa hyödyntää jo olemassa olevia ominaisuuksia ja toteutuksia palveluiden tuottamiseen.

Räätälöidyt palvelut ovat ominaisuuksia, joita voidaan tuottaa johonkin erityiseen, perusrajapinnan toiminnasta poikkeavaan tarpeeseen. Yksinkertainen esimerkki tällaisesta tarpeesta on binäärimuodossa olevan tiedon, kuten kuvan, tulostaminen vastauksena pyyntöön.

Rajapinnalla pitää olla myös toiminnot sen sisältämien palveluiden hallintaan ja suorittamiseen. Hallinnalla tarkoitetaan niitä rajapinnan keinoja, joiden perusteella voidaan päätellä, voiko palvelun yleensäkään suorittaa ja onko suorittaminen mahdollista kyseiselle asiakkaalle (client). Palvelusäilö on tietokantatason ominaisuus, joka vastaa juuri tähän tarpeeseen. Palvelusäilö sisältää tietueen jokaisesta rajapinnan sisältämästä palvelusta sekä niihin liittyvistä suojauksista ja rajauksista, kuten ajo-oikeuksista. Palvelun suorittaminen on rajapinnan kannalta erillisen ohjelman suoritus.

7.4 Looginen näkymä (Logical view)

Enterprise Web Services koostuu itse rajapintaan, Enterprise-toiminnanohjausjärjestelmään, toiminnanohjausjärjestelmän tietokantaan ja ulkoiseen asiakkaaseen liittyvistä kokonaisuuksista. Enterprise Web Services –rajapintaan liittyvät loogiset kokonaisuudet on kuvattu kuviossa 19. Kuviossa sininen väri kuvastaa web service -rajapinnan kiinteitä osia, vihreä väri kuvastaa Enterprise toiminnanohjausjärjestelmän ominaisuuksia, keltainen väri kuvastaa tietokantatason ominaisuuksia ja harmaa väri järjestelmän ulkoisia loogisia osia. Loogisten osien väliset viivat kuvastavat niiden yhteyttä ja suhdetta toisiinsa.

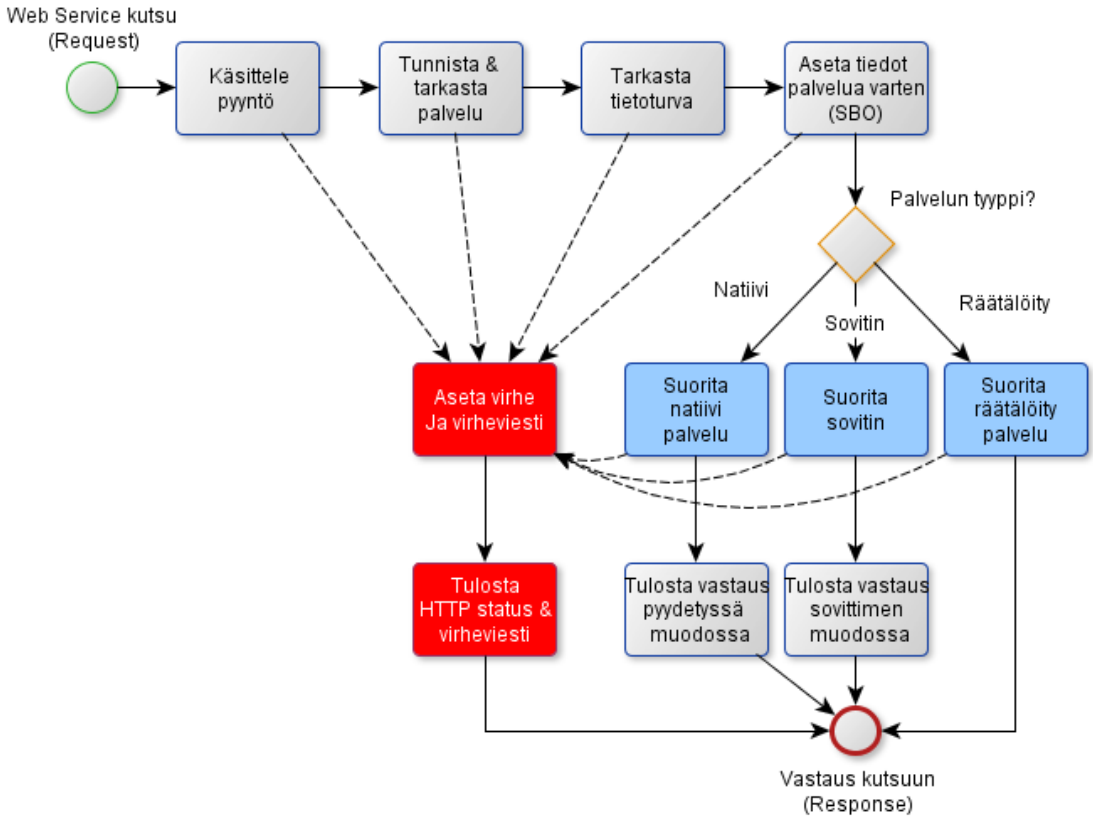


KUVIO 19. Enterprise Web Services -rajapinnan loogiset kokonaisuudet

Rajapinta konkretisoituu palvelun käsitteeseen ja siihen liittyviin kokonaisuuksiin. Rajapinta itsessään tarjoaa ulkoisen liitännän palveluiden käyttäjille, eli asiakkaille (client). Ulkoisen liitännän kautta asiakkaiden on mahdollista käyttää järjestelmän sisältämää liiketoimintalogiikkaa ja tietoa tarjottujen palveluiden välityksellä. Ulkoisen liitännän lisäksi rajapinta pitää sisällään useita sisäisiä liitäntöjä toiminnanohjausjärjestelmän kokonaisuuksiin, kuten teknisesti erityyppisiin palveluihin, istuntoon (kontekstiin) sekä Enterprise toiminnanohjausjärjestelmän käyttöoikeustoimintoihin. Varsinainen tietovarastojen tietoa hyödyntävä liiketoimintalogiikka on käytettävissä ainoastaan yksittäisten palvelujen kautta. Rajapinta on siis toimiva ”kone”, jonka muodostaa ja jonka osina toimivat toiminnanohjausjärjestelmän ulkoiset ja sisäiset kokonaisuudet.

7.5 Prosessi näkymä (Process View)

Enterprise Web Services –rajapinnan prosessi on suoraviivainen: se sisältää vähän vaiheita ja itse palvelun suorittamista lukuun ottamatta, ne kaikki ovat suuntautuneet vastaanotetun kutsun todentamiseen ja tarkastamiseen. Kuviossa 20 esitetään rajapintaan liittyvä prosessi yleisellä tasolla kutsun vastaanottamisesta aina vastauksen lähettämiseen asti. Kuviossa harmaat laatikot esittävät prosessin vaiheita, siniset laatikot ovat varsinaisen palvelun suorittamiseen liittyvät prosessin vaiheet, punaiset laatikot kuvaavat virhekäsittelyn vaiheita, nuolet siirtymiä vaiheiden välillä sekä katkonuolet siirtymistä normaalista prosessista virheenkäsittelyyn.



KUVIO 20. Enterprise Web Services -rajapinnan prosessi

Palvelu kutsun, eli pyynnön, käsittely tarkoittaa niitä yleisiä toimenpiteitä, joilla varmistetaan, että kutsu on muodollisesti oikein. Muodollisuus vaatimukset täytetään, jos kutsu tulee tuetussa formaatissa, tuetulla http-protokollan metodilla ja että se sisältää riittävät tiedot kutsuttavan palvelun tunnistamiseen.

Palvelu tunnistetaan kutsun URI:n, eli osoitteen, perusteella. URI määrittää rajapinnan sijainnin lisäksi myös suoritettavaksi tarkoitetun palvelun yksilöivän tunnisteen. URI muodostuu seuraavasti:

protokolla://sijainti/rajapinta/palvelu

Palvelun URI voisi olla esimerkiksi https://server.domain/webservice/item. Palvelun yksilöimisen jälkeen rajapinta tarkastaa, että palvelusäilöstä löytyy voimassaoleva palvelu kyseisellä tunnuksella.

Jos kutsuttu palvelu tunnistetaan ja todetaan voimassaolevaksi, aloitetaan sovelluksen sisäiset tietoturvaan liittyvät toimenpiteet. Järjestelmän avulla voidaan teknisesti tuottaa sekä kaikille avoimia että oikeuksilla suojattuja palveluita. Suojattujen palveluiden osalta varmennetaan (autentikoidaan) palvelua kutsuvan käyttäjän (client) oikeudet palveluun ja sen toimintoihin. Autentikoinnin yhteydessä varmistetaan myös istunnon eheys (integrity) sekä asetetaan kutsua vastaavat puitetiedot (context). Puitetietojen perusteella varsinainen palvelu tietää esimerkiksi kutsujan ”henkilöllisyyden” toiminnanohjausjärjestelmässä, kuten käyttäjän tai asiakkaan tiedot.

Jos kutsu läpäisee tietoturvatarkastukset, käsitellään sen sisältämät tiedot, kuten URI:ssa välitetyt parametrit, kutsussa olevat xml- tai json-muotoiset tiedot tai välitetyn html-lomakkeen tiedot. Käsittelyn yhteydessä tiedot tarkastetaan muodon osalta siten, että ne ovat kelvollisia välitettäväksi varsinaiselle palvelulle. Käsitellyt tiedot välitetään Enterprise-toiminnanohjausjärjestelmän sisältämän rajapinnan (SBO) kautta varsinaiselle palvelulle lähtötiedoiksi. Jokainen palvelu tarkastaa itsenäisesti palvelukohtaiset pakolliset tiedot ja tietotyypit.

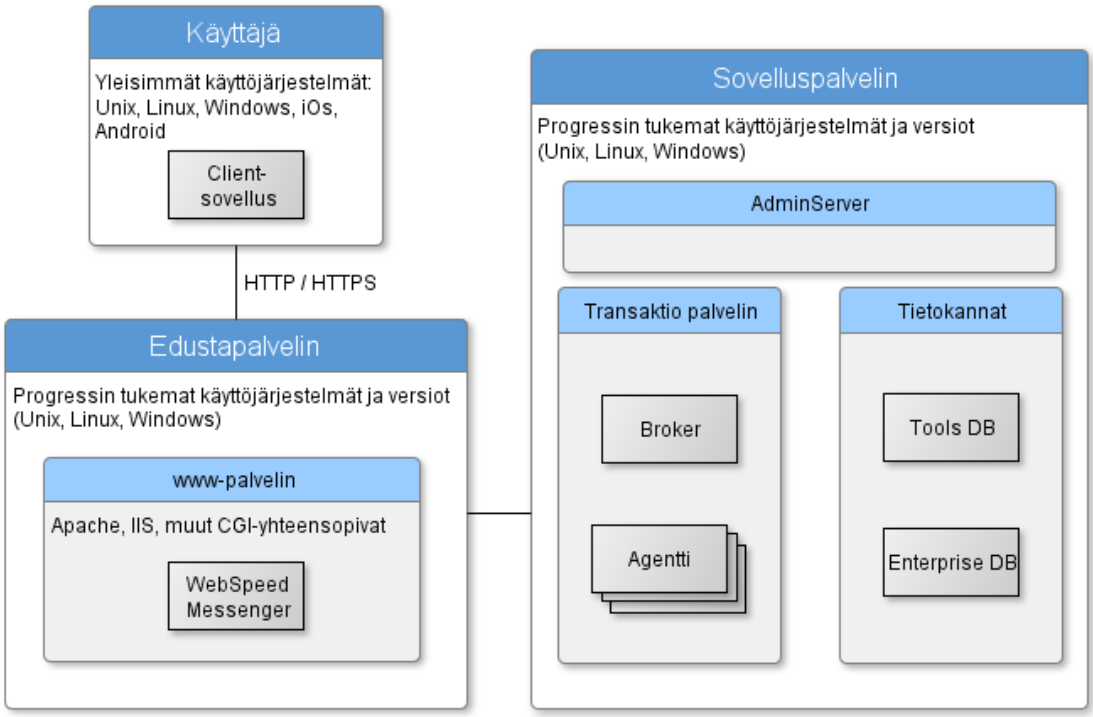
Kun kutsu on läpäissyt kaikki varsinaiset tarkastukset, suoritetaan varsinainen palvelu. Palvelu voi olla natiivi Progressin tietomalleihin (ProDataSet tai TempTable) perustuva palvelu, sovitinta hyödyntäen ajettava Enterprise-toiminnanohjausjärjestelmän ominaisuus tai täysin räätälöity palvelu. Web service -rajapinta hoitaa sekä natiivien että sovitinta hyödyntävien palveluiden tuotoksen tulostamisen vastauksena saatuun kutsuun. Räätälöityjen palveluiden

osalta palvelu itse hoitaa kaikki tulostamiseen liittyvät toimet. Tämä johtuu siitä, että rajapinta on luotu tulostamaan määrämuotoista merkkipohjaista tietoa (xml, json, html ja csv), mutta räätälöityä palvelua voidaan käyttää tulostamaan myös binäärimuodossa olevaa dataa.

Enterprise Web Services -rajapinta tukee keskitettyä virheiden käsittelyä. Jokainen rajapinnan vaihe voi asettaa kutsun käsittelyn virhetilanteeseen. Virhetilanteen asettaminen katkaisee rajapinnan normaalin prosessin ja siirtyy käsittelemään aiheutunutta virhetilannetta. Virhetilanteen käsittely tarkoittaa asianmukaisen http-status-koodin sekä virhetilanteesta kertovan virheviestin palauttamista vastauksena kutsuun.

7.6 Fyysinen näkymä (Physical View)

Enterprise Web Services –rajapinnan fyysinen näkymä koostuu kolmesta kokonaisuudesta: käyttäjä, edustapalvelin ja sovelluspalvelin. Käyttäjä on teknisesti täysin rajapinnasta riippumaton ja käyttäjä voi käytännössä olla mikä tahansa järjestelmä tai sovellus, joka kykenee määrämuotoiseen viestintään http-protokollaa käyttäen. Edustapalvelin ja sovelluspalvelin muodostavat teknisen toisistaan riippuvan kokonaisuuden, joka tuottaa teknisen alustan palveluiden tuottamiselle ja tarjoamiselle. Kuviossa 21 kuvataan yleisellä tasolla rajapintaan liittyvät fyysiset kokonaisuudet.



KUVIO 21. Enterprise Web Services -ympäristön fyysinen näkymä

Enterprise Web Services –rajapinnan käyttäjä (client) voi olla mikä tahansa tietotekninen järjestelmä tai sovellus, joka http-protokollaa hyödyntäen keskustelee rajapinnan välityksellä toiminnanohjausjärjestelmän kanssa. Käyttäjä on rajapinnasta täysin teknisesti riippumaton, joten sen taustalla olevat ratkaisut voivat olla hyvin monimuotoisia. Ainoat vaatimukset asiakkaalle ovat http-protokollan käyttäminen sekä jonkin rajapinnan käyttämän formaatin tukeminen.

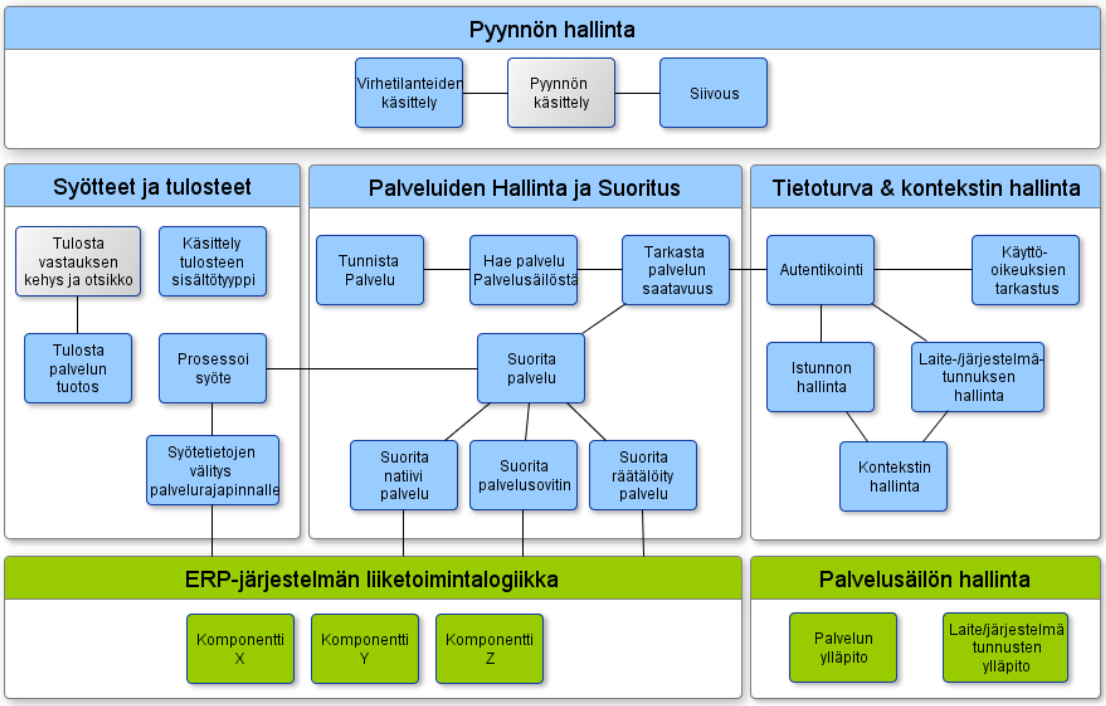
Edustapalvelin toimii rajapinnan julkisena yhteyspisteenä kaikille asiakkaille. Edustapalvelin vastaanottaa asiakkaiden palvelukutsut ja välittää ne suoritettavaksi sovelluspalvelimelle. Se myös välittää sovelluspalvelimen muodostaman vastauksen asiakkaalle. Edustapalvelin voi olla mikä tahansa palvelin ja käyttöjärjestelmä yhdistelmä, johon voidaan asentaa CGI-yhteensopiva www-palvelin. Progress OpenEdge -sovelluskehitysalusta ja ajoympäristö ja niiden komponentit asettavat rajoitteita edustapalvelimen laiteympäristölle, käyttöjärjestelmälle ja www-palvelimelle. Yleisimmät käyttöjärjestelmät, kuten Windows eri versioineen sekä yleisimmät Linux ja UNIX distribuutiot ovat tuettuja. Www-palvelimena voidaan käyttää mitä tahansa CGI-yhteensopivaa palvelinta, mutta käytännössä

vaihtoehdot ovat Microsoft IIS Windows-ympäristössä ja Apache HTTP Server Linux ja UNIX ympäristöissä.

Sovelluspalvelin on koko Enterprise Web Services –rajapinnan ydin. Sovelluspalvelin esitetään tässä loogisena kokonaisuutena, joka itse asiassa sisältää varsinaisen sovelluspalvelimen ja tietokantapalvelimen. Nämä kokonaisuudet voivat tarpeen mukaan sijaita samalla fyysisellä resurssilla tai ne voidaan eriyttää omille laitteilleen. Sovelluspalvelimella tapahtuu kaikki varsinainen palveluihin liittyvä prosessointi, sekä rajapintaan liittyvät toimet että varsinaisen palvelun logiikan suoritus. Tietokantapalvelin ylläpitää Enterprise-toiminnanohjausjärjestelmän tietokantoja. Sovelluspalvelimella sijaitsevat agentti-prosessit suorittavat palvelun ohjelmakoodin ja hakevat tietokantapalvelimelta sen suorittamiseen vaadittavat tiedot. Sekä edustapalvelimen että sovelluspalvelimen laitealusta ja käyttöjärjestelmä ovat riippuvia Progress OpenEdge sovelluskehitysalustan ja ajoympäristön tukemista versioista. Tuettuja ovat yleisimmät käyttöjärjestelmät, kuten Windows, Linux sekä UNIX yleisimpine versioineen ja distribuutioineen.

7.7 Kehitysnäkymä (Development View)

Enterprise Web Services -rajapinnan täytyy sisältää toiminnot pyynnön hallintaan, syötteiden ja tulosteiden hallintaan, palveluiden hallintaan ja suoritukseen sekä tieturvan ja kontekstin hallintaan. Lisäksi rajapinnan täytyy pystyä hyödyntämään Enterprise-toiminnanohjausjärjestelmän liiketoimintalogiikkaa palveluiden välityksellä. Erillisenä kokonaisuutena rajapinnan täytyy sisältää myös ylläpitotoiminnot palvelusäilön hallintaan. Kuviossa 22 on kuvattu rajapinnan kehitykselle merkitykselliset kokonaisuudet.



KUVIO 22. Enterprise Web Services -rajapinnan pääkomponentit

Pyynnön hallinta tarkoittaa niitä ominaisuuksia, joilla hallitaan yleisesti koko rajapintaan liittyvää prosessia, sen keskitettyjä poikkeus- ja virhekäsittelyitä sekä suorituksen jälkeistä ajotietojen siivousta. Pyynnön käsittely on looginen prosessi, joka edetessään suorittaa kaikki rajapintaan liittyvät yksittäiset toiminnot. Pyynnön käsittely on siis ohjelmarunko, joka suorittaa prosessinäkymän mukaiset toimet. Virhetilanteiden käsittely on keskitetty poikkeus- ja virhetilanteiden käsittely. Se on yleisen tason catch-rutiini, johon propagoidaan kaikki alemman tason rutiineissa tapahtuvat virheet. Siivous tarkoittaa ajonaikaisten tietojen tuhoamista rajapintaa suorittavan prosessin muistista siten, että ne eivät haittaa seuraavia palveluiden suorituksia.

Syötteiden ja tulosteiden käsittely tarkoittaa niitä rutiineja, jotka vastaavat rajapintaan kohdistuvien pyyntöjen syötteen, eli sisällön ja välitettyjen parametrien, käsittelystä, sekä kutsuihin vastauksena lähetettävien tietojen muotoilusta ja tulostamisesta. Syötteen käsittely tarkoittaa sitä, että tarkastetaan välitettyjen tietojen muoto sekä prosessoidaan ja välitetään tiedot toiminnanohjausjärjestelmän sisäistä SBO-rajapintaa hyödyntäen palvelulle. Tietojen tulostaminen sisältää asiakkaan (client) pyytämän sisältötyypin (content-type) evaluoinnin,

sisältötyypin asettamisen sekä palvelun vastauksen tulostamisen asetetussa sisältötyypissä.

Palveluiden hallinta tarkoittaa pyydetyn palvelun tunnistamista sekä tunnistetun palvelun ja sen saatavuuden tarkastamista palvelusäilöstä. Suoritus itsessään tarkoittaa varsinaisen palveluun liittyvän ohjelman suorittamista palveluun liittyvän tyypin perusteella. Erilaisia palvelutyyppejä ovat natiivipalvelu, sovittimen kautta ajettava toiminnanohjausjärjestelmän piirre sekä räätälöity palvelu.

Tietoturva ja kontekstin hallinta tarkoittavat niitä rutiineja, joilla tunnistetaan palvelua kutsuva taho ja joilla tarkastetaan palvelua kutsuvan tahon oikeudet ajaa palvelua sekä ylläpidetään kontekstia saman asiakkaan eri kutsujen välillä. Tunnistaminen ja käyttöoikeuksien tarkastaminen tapahtuu ainoastaan sellaisten palveluiden osalta, jotka on suojattu käyttöoikeuksin. Kontekstin ylläpitäminen tarkoittaa istunnon muodostamista tausta-järjestelmään sekä istuntoa vastaavan tunnuksen välittämistä jokaisen kutsun mukana.

Palvelusäilön hallinta tarkoittaa niitä toiminnanohjaussovelluksen näyttöjä, joita käytetään yksittäisten palveluiden sekä laite- ja sovellustunnusten ylläpitämiseen. Jokainen yksittäinen palvelu täytyy tallentaa palvelusäilöön sekä asettaa siihen liittyvät käyttöoikeustiedot, jotta se on julkaistu ja käytettävissä. Laite- ja sovellustunnus mahdollistaa yksittäisen laitteen tai sovelluksen tunnistamisen toiminnanohjausjärjestelmän puitteissa. Laitteiden ja sovellusten tunnistaminen ilman käyttäjätunnukseen perustuvaa autentikointia vaatii tunnuksen tallentamista järjestelmään sekä sen välittämistä salatussa muodossa jokaisen kutsun mukana.

Jokaisella yksittäisellä palvelulla on mahdollisuus hyödyntää toiminnanohjausjärjestelmän liiketoimintalogiikkaa. Yksinkertainen esimerkki hyödynnettävästä logiikasta on esimerkiksi nimikkeen varastosaldon tai hinnan laskeminen. Palveluiden hyödyllisyyden sekä toteuttamisen kannattavuuden kannalta on toiminnanohjausjärjestelmän sisältämän logiikan uudelleenkäytettävyys erittäin tärkeässä roolissa.

8 ARKKITEHTUURIN ARVIOINTI

Enterprise Web Services –rajapinnan arkkitehtuurin arviointiin käytettiin useita tapoja epävirallisesta läpikäynnistä ja esittelyistä aina viralliseen katselmointiin ja konkreettisen sovellusrungon luontiin asti.

Itse arkkitehtuurin arvioinnin kannalta hedelmällisimmiksi tavoiksi osoittautuivat sovellusrungon toteuttaminen sekä muodollinen katselmointi. Näillä tavoilla oli mahdollista pureutua riittävän konkreettisella tasolla arkkitehtuuriin ja siinä mahdollisesti piileviin epäkohtiin. Sovellusrunko ja muodollinen katselmointi toivat esille konkreettisimmat suunnitelmaan vaikuttaneet muutokset.

8.1 Esittely

Tehdyn arkkitehtuurisuunnitelman esittelyä käytettiin lähinnä kommunikointiin sisäisten ja ulkoisten sidosryhmien kanssa. Esittelyssä käytettiin yleisarkkitehtuuriin perustuvaa esitystä, jonka avulla sidosryhmille esiteltiin lähinnä rajapinnan mahdollisuuksia ja toimintoja. Varsinaisen arkkitehtuurin arviointiin tai teknisen toteutuksen mahdollisuuksien määrittelyyn esittelyä ei käytetty.

Esittely on vahvimmillaan informaation levittämisessä kyseessä olevasta hankkeesta. Sen avulla voidaan pitää lyhyitä epävirallisia tai virallisia tietoiskuja riippuen kuulijakunnasta. Esittely on omiaan keskustelun avaajana ja apuvälineenä keskeisten suunnitelmaan liittyvien kokonaisuuksien käsittelyssä. Esittelyllä saatiin tarkennettua Enterprise Web Services –rajapinnan arkkitehtuurille asetettuja vaatimuksia ja rajauksia.

Varsinaiseen arkkitehtuurin arviointiin esittely ei sovellu. Näin siksi, koska kaikkien osallistujien pitäisi ymmärtää arvioitavaa kokonaisuutta syvällisesti ja lisäksi ymmärtää laajempia riippuvuussuhteita, jotta hyvin abstraktilla tasolla olevan suunnitelman arviointi olisi ylipäänsä mahdollista.

8.2 Sovellusrunko

Ensimmäinen varsinainen arkkitehtuurisuunnitelman arviointi tapahtui toteuttamalla sovellusrunko. Sovellusrunko toteutettiin iteroimalla arkkitehtuurisuunnitelman pohjalta eri kokonaisuuksia yksi osio kerrallaan. Sovellusrungon toteuttamisessa käytetyt iteraatiot olivat kehitysnäkymän mukaisesti jaoteltuna seuraavat:

1. pyynnön hallinta
2. syötteet ja tulosteet
3. palveluiden hallinta ja suoritus
4. tietoturva ja kontekstin hallinta

Varsinaisen rajapinnan lisäksi sovellusrunkoa tehtäessä toteutettiin myös useita mallipalveluita. Mallipalveluita käytettiin sovellusrungon ja sen sisäisen logiikan testaamiseen. Myös mallipalvelut kehittyivät iteraatioiden edetessä yksittäisen kyselyn perusteella luotavasta palvelusta aina periaatteessa tuotantokelpoisen toiminnanohjausjärjestelmän liiketoimintalogiikkaa hyödyntävien palveluiden luontiin asti.

Sovellusrunkoa toteutettaessa tehtiin seuraavat huomiot ja päätökset:

- rajapinnan sisäiset tapahtumat pitää pystyä kirjaamaan lokitiedostoon
- palvelusovittimia ei toteuteta osana alkuperäistä kehitysprojektia.

Sovellusrunkoa toteutettaessa ja testattaessa huomattiin, että rajapinnan sisäistenkin tapahtumien seuraamiselle jälkikäteen on erityinen tarve. Arkkitehtuurin vaatimuksissa oli mainittu ainoastaan vaade virhetilanteiden ja tapahtumien, eli palvelukutsujen, kirjaamiselle loki-tiedostoon. Tämä ei kuitenkaan riitä, vaan myös rajapinnan sisäisistä tapahtumista pitää pystyä kirjaamaan loki-tapahtumat. Tällainen tarve seuraa tilanteista, joissa asiakas (client) ei ole pystynyt käsittelemään rajapinnan palauttamaa virhettä oikein ja tapahtuman lopputulos jää epäselväksi. Ilman sisäisten tapahtumien kirjaamista tällaisesta tilanteesta voitaisiin loki-tietojen perusteella selvittää ainoastaan että ajettiinko palvelu ja tapahtuko ajossa jokin virhe. Jos virheen syy oli rajapinnassa, siitä ei olisi jäänyt merkintää loki-tiedostoon. Tarpeen seurauksena päätettiin, että raja-

pintaan tullaan erillisenä kehitystyönä toteuttamaan sisäisten tapahtumien kirjaus lokitiedostoon.

Palvelusovittimien osalta päätettiin siirtää toteutus odottamaan ensimmäistä varsinaista asiakastarvetta. Sovellusrungon toteutuksessa huomioitiin palvelusovittimien liittäminen osaksi rajapintaa, mutta itse palvelusovittimia ei toteutettu. Päätös johtui siitä, että päätöksenteko hetkellä tiedossa olleissa rajapinnan käyttötarpeissa ei sovittimille ollut tarvetta. Ominaisuutta ei kokonaan haluttu hylätä, koska sovittimille nähtiin kuitenkin tarve tietynlaisissa projekteissa.

Sovellusrungon toteuttaminen vahvisti sen, että suunniteltu arkkitehtuuri on käytännössä toimiva. Sovellusrungon perusteella oli myös mahdollista havaita ja oikaista asioita, joita arkkitehtuurisuunnittelussa ei ollut osattu huomioida tai joista oli tehty vääriä päätelmiä. Tehtyjen havaintojen perusteella arkkitehtuurisuunnitelmaa oikaistiin ja korjaukset tehtiin mahdollisuuksien mukaan myös sovellusrunkoon.

8.3 Muodollinen katselmointi

Arkkitehtuurin muodollinen katselmointi tapahtui vasta sovellusrungon toteuttamisen jälkeen. Koska melko laaja sovellusrunko oli jo olemassa, voitiin muodollisen katselmoinnin yhteydessä tarkastella tarkemmin myös itse toteutusta sekä suorittaa rajapinnalle erilaisia testejä.

Katselmoinnin tekniselle ratkaisulle ja tekniselle dokumentaatiolle suoritti Enterprise-toiminnanohjausjärjestelmäkehityksessä toimiva vanhempi järjestelmäarkkitehti. Katselmointia suorittamaan valittiin henkilö, jolla oli riittävät tekniset tiedot ja taidot sekä Enterprise toiminnanohjausjärjestelmästä että kyseessä olevista teknologioista ja tekniikoista yleensä. Lisäksi henkilö pystyi myös tietojensa ja kokemuksensa perusteella arvioimaan toteutetun kokonaisuuden hyödyllisyyttä projektitoiminnassa.

Arkkitehtuuri ja toteutettu sovellusrunko läpäisivät muodollisen katselmoinnin dokumentointeineen. Katselmoinnin lopputuloksena nostettiin esille vähäisiä tarpeita muutoksiin ja korjauksiin koskien:

- yksittäisten palvelusovellusten muistinkäyttöä sekä muistin varaamista ja vapauttamista
- rajapinnan ja yksittäisen palvelun välisen tietojen siirron optimointia
- toiminnanohjausjärjestelmän moniyrityspiirteiden tukemisen taso rajapinnassa
- http-protokollan statuskoodien käyttöä vastauksen yhteydessä

Joidenkin toteutettujen yksittäisten palvelusovellusten muistinkäytön kanssa oli sellainen ongelma, että tietyissä tapauksissa ei vapautettu kaikkea varattua muistia, vaan muisti jäi suorittavan prosessin varaamaksi. Tämän ongelman seurauksena rajapinnan toiminta olisi ajan myötä hidastunut ja jossain vaiheessa koko sovellus ja mahdollisesti palvelinkin olisi kaatunut vapaan muistin puutteeseen. Ongelma korjattiin muuttamalla rajapinnan siivousrutiineja kattamaan havaitut poikkeustapaukset.

Rajapinnan ja yksittäisen palvelun välisen tietojen siirron optimointi liittyy myös sovelluksen muistinkäyttöön. Alkuperäisessä mallissa yksittäinen palvelu palauttaa takaisin tuottamansa tietorakenteen tavallisena parametrina rajapinnalle. Tämä toimintatapa aiheutti tietorakenteen kopioitumisen, eli samasta tiedosta löytyi kaksi eri ilmentymää sovelluksen muistista. Ongelma korjattiin siten, että palvelu palauttaa rajapinnalle kahvan alkuperäiseen tietoon, jolloin kopioitumista ei tapahdu. Näin rajapinta käyttää palvelun tuottamaa tietorakennetta, eikä muistinkäyttö lisäänny.

Enterprise-toiminnanohjausjärjestelmä tukee moniyrityskäyttöä. Moniyrityskäyttö tarkoittaa sitä, että samassa järjestelmässä voi olla useiden yritysten tietoja ja käyttäjillä on oikeus käyttää joko yhtä tai useampaa yritystä. Käyttöliittymässä käyttäjällä on mahdollisuus valita, minkä hänelle sallitun yrityksen tietoja hän haluaa järjestelmässä käyttää. Tämän osalta tehtiin päätös, että Enterprise Web Services –rajapinnassa tuetaan toistaiseksi ainoastaan käyttäjän oletusyritystä.

Rajapinta palautti alkuperäisen suunnitelman mukaisesti ainoastaan Http-protokollan statuskoodin ”200 OK” kaikissa tilanteissa, myös silloin jos sovelluksen sisällä tapahtui virhe. Muita statuskoodeja palautettiin ainoastaan teknisessä ympäristössä tapahtuneiden virheiden seurauksena. Muutoksen jälkeen rajapinta palauttaa asianmukaisen statuskoodin myös sovelluksen sisäisistä virhetilanteista.

Katselmoinnin yhteydessä rajapintaa testattiin lähinnä kuormituksen keston sekä itse rajapinnan vasteeseen aiheuttaman viiveen osalta. Kuormituksen keston osalta todettiin, että rajapinta skaalautuu teknistä sovellusalustaa hyödyntäen riittävästi tukeakseen tapauksia, joissa käyttäjiä ja käyttöä on runsaasti. Kuormitus-testauksen osalta todettiin kuitenkin, että kuormituksella on vaikutusta yksittäisen palvelukutsun suoritusaikaan. Rajapinnan aiheuttaman viiveen todettiin olevan hyväksyttävissä rajoissa, mutta sen osalta tulisi tutkia mahdollisuuksia pienentää viivettä.

9 TULOKSET

Arkkitehtuurisuunnitelman perusteella toteutettiin arkkitehtuuriarvioinnin yhteydessä tuotetusta sovellusrungosta täysin toimiva web service –rajapinta. Toteutettu rajapinta on hyväksytty osaksi tuotetta ja se tullaan julkaisemaan virallisesti Enterprise toiminnanohjausjärjestelmän version 4.7.0 yhteydessä toukokuussa 2012.

Enterprise Web Services –rajapinta esiteltiin toimeksiantajan marraskuussa 2011 pitämässä asiakaspäivässä. Samalla kerralla esiteltiin myös lyhyesti toiminnanohjausjärjestelmän yhteyteen rajapintaa hyödyntäen rakennettuja palveluita ja sovelluksia. Pisimmälle viedyt esimerkit olivat Joomla! sisällönhallintajärjestelmän hyödyntäminen web-sovelluksien alustana sekä Apple iPad tabletissa toimiva mobiilia työskentelyä tukeva SmartTab-sovellus. Sovellukset esitellään lyhyesti kappaleessa 9.2. Esitellyt kokonaisuudet saivat asiakkailta erittäin hyvän vastaanoton. Asiakaspalaute sekä ominaisuuksista että tapahtumasta yleensä oli erittäin positiivista.

9.1 Toteutuivatko vaatimukset ja pysyttiinkö rajoitteissa?

Opinnäytetyötä kirjoitettaessa on web service –rajapinta toteutettu ja ensimmäiset siihen perustuvat asiakasprojektit ja kehityshankkeet ovat lähteneet liikkeelle. Yleisesti voidaan siis arvioida, että kehittämistehtävä on onnistunut. Tarkemmalla tasolla kehittämistehtävän onnistumista toimeksiantajan näkökulmasta voidaan arvioida vertaamalla tuloksia asetettuihin vaatimuksiin ja rajoituksiin. Tätä samaa vertailua on osittain tehty myös arkkitehtuurin arvioinnin yhteydessä.

9.1.1 Yleiset tavoitteet

Yleisen tason tavoitteina web service –rajapinnan kehittämisellä asetettiin toiminnanohjausjärjestelmään liittyvien integraatioiden helpottaminen sekä sovellusteknisesti että laite- ja henkilöresursointien osalta. Lisäksi tavoitteena oli myös työskentelyn tehostaminen teknisestä ratkaisusta seuraavalla toimintatapojen muutoksella sekä potentiaalisten projektiresurssien määrän kasvattaminen yleisellä tasolla.

Integraatioiden helpottuminen sovellusteknisesti ja integraatiomahdollisuuksien laajeneminen toteutuivat erinomaisesti. Tästä on hyvänä esimerkkinä alaluvussa 9.2 esiteltävät kehityshankkeet. Käytännön tasolla rajapintaa hyödyntäen on toteutettu eri teknologioihin perustuvia ratkaisuja, jotka hyödyntävät toiminnanohjausjärjestelmän tietosisältöä sekä liiketoimintalogiikkaa. Ja mikä parasta, samoja rajapinnan tuottamia palveluita on voitu hyödyntää ristiin molemmissa hankkeissa.

Myös työskentelyn tehostaminen ja potentiaalisten projektiresurssien määrän kasvu ovat toteutuneet jo tässä vaiheessa kohtuullisella tasolla. Molemmissa alaluvussa 9.2 esiteltävissä hankkeissa on osa mukana olleista kehitysresursseista sellaisia henkilöitä, joilla ei ole aiempaa kokemusta toiminnanohjauksesta. Toki tukena on ollut toiminnanohjausta tuntevia henkilöitä. Pitkälle meneviä päätelmiä tästä ei kuitenkaan voida vielä tehdä, koska rajapinta palveluineen on käytössä vasta rajatusti muutamassa projektissa ja se vaikuttaa toistaiseksi vain pienen henkilöjoukon tekemiseen.

Projektista seurasi kuitenkin myös suunniteltua laajempia hyödyntämismahdollisuuksia liittyen toiminnanohjauksellisen tiedon hyödyntämiseen ulkoisissa välineissä. Uusia käyttömahdollisuuksia tunnistettiin esimerkiksi ulkoisten raportointivälineiden, kuten Cognos Powerplay tai QlikView, yhteydessä. Kaikkea toiminnanohjauksen tietoa ei saada selville suorilla tietokantakyselyillä, joten näissä tapauksissa voidaan hyödyntää laskennallisesti haastavia toiminnanohjauksen ominaisuuksia web service –rajapinnan palveluina.

Varsinainen haaste työskentelyn tehostamisen ja rajapinnan laajemman käytön osalta on vastassa uuden rajapinnan sisältävän tuote-version julkaisemisen yhteydessä. Julkaisemisen yhteydessä rajapinta jalkautetaan koulutusten avulla asiakasprojektien käyttöön. Vasta jalkauttamisen jälkeen voidaan ensimmäisen kerran arvioida toteutetun teknisen ratkaisun mahdollisia vaikutuksia toimintatapoihin ja hyötyjä asiakasprojekteille. Lisäksi konkreettisten hyötyjen arvioimiseen tarvitaan useampia alkuperäisistä toteuttajista riippumattomia asiakastoteutuksia. Siihen asti kaikki on enemmän tai vähemmän teoreettisella tasolla tapahtuvaa pohdintaa.

9.1.2 Toiminnalliset vaatimukset

Enterprise Web Services –rajapinnalle asetetut toiminnalliset vaatimukset olivat melko yksinkertaiset. Vaatimusten yksinkertaisuus johtuu siitä, että itse rajapinta on mahdollistaja eikä yksittäinen sovellus, joka ratkaisee tietyn ongelman. Rajapinnan hyödyntäminen mahdollistaa asiakasprojekteissa asetettujen vaatimusten saavuttamisen sen päälle rakennettujen ratkaisujen avulla.

Rajapinnalle voidaan kuitenkin asettaa toiminnallisia vaatimuksia sekä projektien että sovelluskehitystyön näkökulmasta. Tällaisia vaatimuksia ovat esimerkiksi tiettyjen formaattien tukeminen, tietyn tietoliikenneprotokollan tukeminen tai vaatimus välineille kehitystyön ja testauksen tukemiseen.

Rajapinnalle asetetut toiminnalliset vaatimukset täytettiin kaikkia osapuolia tyydyttävällä tavalla. Rajapinnan sisältämä tuki useille määritetyille formaateille mahdollistaa asiakasprojekteissa ja muissa kehityshankkeissa mahdollisimman laajan vaihtoehtojen kirjon tietosisältöön liittyvien tarpeiden ratkaisemiseksi. Lisäksi rajapintaa kehitettäessä kiinnitettiin erityisesti huomiota sekä kehittämisettä testausvälineisiin. Näin pyritään pienentämään uuden ratkaisun käyttöönoton yhteydessä mahdollisesti esiintyvää muutosvastarintaa.

9.1.3 Ei-toiminnalliset vaatimukset

Osaan ei-toiminnallisista vaatimuksista saatiin valmisratkaisu rajapinnan tekniseen toteutukseen käytetyistä Progress OpenEdge teknologiosta. Rajapinnan skaalautuminen sekä yhden fyysisen laitteen resurssien että usean laitteen resurssien hyödyntämisen osalta on mahdollista Progress OpenEdge WebSpeed -teknologian tarjoamien konfiguraatio- ja hajauttamisvaihtoehtojen avulla. Rajapinta ja sen sisältämien palveluiden suorittaminen on näin mahdollista jakaa useille fyysisille resursseille. Progress-teknologian hyödyntäminen mahdollistaa myös rajapinnan ja sen sisältämien palveluiden siirrettävyyden erilaisten Progressin tukemien palvelin ja käyttöjärjestelmäyhdistelmien välillä. Vaikka tämä ei kata kaikkia mahdollisuuksia, löytyy tuki kuitenkin kaikille yleisille laitealustoille ja käyttöjärjestelmille.

Enterprise Web Services –rajapinnan vasteaika ja itse rajapinnan aiheuttaman viiveen määrä kutsujen käsittelyyn koettiin tärkeiksi huomioitaviksi seikoiksi arkkitehtuurin suunnittelussa ja varsinaisen rajapinnan toteutuksessa. Itse rajapinnan vasteaikaan aiheuttaman viiveen minimointiin käytettiin kohtuullisesti aikaa ja se saatiinkin laskettua hyväksyttävälle tasolle optimoimalla rajapinnan toimintoja. Varsinaisten palvelujen toteutus on kuitenkin huomattavasti rajapinnan toteutusta merkittävämmässä roolissa mitattaessa rajapinnan vasteaikaa kokonaisuudessaan. Yksittäisen palvelun toimintaan ei rajapinnan keinoin pystytä juuri vaikuttamaan. Tästä syystä dokumentaatiolla, ohjeistuksella sekä piirteen jalkautukseen tähtäävällä koulutuksella on merkittävä rooli rajapinnan ja sen sisältämien palveluiden käytettävyydessä.

Tietoturva on erityisessä asemassa silloin, kun tarkoituksena on toteuttaa yrityksen toimintaan liittyvään arvokasta tietoa sisältävään järjestelmään rajapinta tiedon jakamista ja sisään lukemista vasten. Arkkitehtuuria suunniteltaessa ja rajapintaa toteutettaessa tietoturvaa käsiteltiin sekä sovelluksen ulkoisen että sovelluksen sisäisen tietoturvan kannalta. Sovelluksen ulkoinen tietoturva keskittyy lähinnä infrastruktuurin, eli laitteiden, verkkoympäristön sekä laitteiden välisten yhteyksien suojaamiseen ja turvaamiseen. Ulkoinen tietoturva hoidetaan täysin sovelluksesta riippumattomin keinoin ja tätä tukemaan on olemassa

jo aiemmin toiminnanohjausjärjestelmän web-moduulia varten tuotettu tietoturvadokumentaatio. Ulkoisen suojaamisen keinoin toteutetaan esimerkiksi ulkoisen viestiliikenteen salaaminen. Rajapinta-sovelluksen sisäisen tietoturva toteutettiin Enterprise-toiminnanohjausjärjestelmän ehdoilla ja toiminnoilla. Rajapintaan toteutettu sovelluksen sisäinen tietoturva vastaa käyttäjien tunnistamisesta, käyttäjien oikeuksien tunnistamisesta ja valvomisesta sekä tarvittaessa istunnon eheyden valvomisesta. Tietoturvavaatimukset saatiin ratkaistua rajapinnan osalta jo tuotantokäytössä hyväksi todetuilla keinoilla.

Rajapinta on tarkoitettu tukemaan asiakkaiden tarpeita muuttuvassa liiketoimintaympäristössä pitkällä aikajänteellä. Tämän toteutumiseksi rajapinnan pitää olla riittävän muunneltava, jotta tarvittavat muutokset voidaan toteuttaa liiketoiminnan kärsimättä tai keskeytymättä. Kokonaan uusien palveluiden ja vanhojen palveluiden uudistamisen osalta tämä vaatimus pystyttiin täyttämään. Uusi palvelu tai uusi versio vanhasta palvelusta voidaan tuoda rajapintaan ilman, että olemassa olevien palveluiden käyttö häiriintyy millään tavalla. Uusien sisältötyyppien tukemisen osalta tähän tavoitteeseen ei kuitenkaan täysin päästy. Uusia sisältötyyppejä voidaan tuottaa kustannustehokkaasti ainoastaan Progress OpenEdge -teknologian tukemissa rajoissa. Lisäksi suunniteltu ja toteutettu sovelluksen rakenne vaatii huoltokatkon uuden sisältötyypin käyttöönoton yhteydessä.

Web service –rajapintaa hyödyntävien ulkoisten sovellusten ja palveluiden osalta saavutettavuus on merkittävässä roolissa ja liittyy suoraan siihen kuinka hyödyllisiä tarjotut palvelut yleisesti ovat. Jos palvelu ei ole saavutettavissa, eli käytettävissä, sen hyödyllisyys ulkoisille käyttäjille on pyöreä nolla. Käytännössä saavutettavuus on sovelluksen toteutuksen laadukkuuden ja fyysisen laiteympäristön vakauden varassa. Koska saavutettavuudelle ei asetettu tarkkaan rajattua vaatimusta, todettiin sen osalta vain, että korkean saavutettavuuden vaatimukset projekteissa ratkaistaan tapauskohtaisesti.

Jotta rajapinta olisi hyödyllinen väline projekteissa, on kaikki sen kautta käytettävät palvelut myös pystyttävä testaamaan tehokkaasti. Koska rajapinta on periaatteessa pelkästään määrämuotoisen tiedon lukemista ja tulostamista on tes-

tattavuuden tukeminen melko helppoa. Rajapinta ja sen palvelut voidaan testata hyödyntäen ulkoisia välineitä sekä toiminnan että kuormituksen keston osalta. Yksinkertainen tapa toiminnan testaamiseen on esimerkiksi Firefox-selaimen Httprequester niminen lisäosa. HttpRequester lisäosa mahdollistaa vapaamuotoisten http-protokollaa käyttävien kutsujen tekemisen palveluihin. Kuormituksen testaamiseen voidaan käyttää vaikkapa Apache JMeter kuormitustestausvälinettä.

Rajapinnan käyttöä asiakkailla pitää pystyä myös tukemaan. Tuki tarkoittaa asiakkaan ohjeistamista, rajapinnan toiminnan seuraamista sekä virhetilanteiden selvittämistä. Asiakkaan ohjeistaminen tehdään dokumentaation avulla sekä projektikohtaisesti annettavalla koulutuksella. Rajapinnan toiminnan seuraaminen ja virhetilanteiden selvittäminen perustuu itse rajapinnan kirjoittamiin lokitietoihin. Näiden toimien ja toimintojen avulla on mahdollista tukea asiakasta rajapintaan liittyvissä asioissa.

9.1.4 Rajoitteet

Arkkitehtuurille ja varsinaiselle rajapinta-sovellukselle asetetuissa teknisissä rajoitteissa pysyttiin täysin. Asetetut tekniset rajoitteet tarkoittivat rajapinnan toteuttamista Progress OpenEdge teknologian välineillä. Tämä rajoite asetettiin, jotta asiakasprojekteissa voidaan jatkossakin toimia pääasiassa samoilla tutuilla resursseilla, eli välineillä, ympäristöillä ja henkilöillä. Rajapintaa suunniteltaessa ja toteutettaessa on pyritty vähentämään tarvetta uusien teknisten asioiden opettelemiseen. Väistämättä muutamia uusia asioita rajapintaan liittyen tulee kehittäjille eteen. Tällaisten asioiden osalta oppimis- ja käyttämiskynnystä pyritään madaltamaan dokumentoinnilla, ohjeistuksella ja koulutuksilla.

Liiketoiminnalliset rajoitteet koskivat lähinnä kehitystyöhön käytettävää kokonaistyömäärää ja työlle asetettua tavoiteaikataulua. Kokonaistyömäärä rajapinnan kehityksen osalta ylittyi, mutta ei merkittävästi. Osittain ylitys selittyy toteutuksen aikana tehdyillä päätöksillä tukea alkuperäiseen määrittelyyn tai työmääräarvioon kuulumatonta toiminnallisuutta. Tavoiteaikataulun osalta ylitys oli

huomattavasti merkittävämpi. Alkuperäisen aikataulun mukaan rajapinnan olisi pitänyt valmistua vuoden 2011 toisella neljänneksellä, mutta varsinaiset työt päästiin aloittamaan kuitenkin vasta vuoden 2011 kolmannella neljänneksellä. Aikataulun siirtyminen selittyy pääasiassa työnkuvani muuttumisella vuoden 2011 alussa. Työnkuvamuutoksen mukanaan tuomat uudet tehtävät veivät niin paljon aikaa, ettei rajapinnan arkkitehtuurisuunnittelu tai toteutus ollut mahdollista alkuperäisessä aikataulussa.

9.2 Enterprise Web Services –rajapintaan perustuvat kehityshankkeet

Toimeksiantajan Enterprise-toiminnanohjausjärjestelmään liittyvät Joomla! sisällönhallintajärjestelmä- ja SmartTab mobiilisovellus -hankkeet mahdollistivat Enterprise Web Services –rajapinnan todentamisen ja testaamisen oikeissa asiakastarpeisiin perustuvissa sisäisissä kehitysprojekteissa. Näin ollen itse rajapinta hankkeessa ei tarvinnut lähteä erikseen toteuttamaan demo-asiakassovelluksia.

Olen itse ollut mukana molemmissa kehityshankkeissa alusta alkaen. Joomla!-sisällönhallintajärjestelmän osalta olen vastannut sekä taustajärjestelmän web services –rajapinnasta että edustajajärjestelmänä toimivasta Joomla!-sisällönhallintajärjestelmästä kehityshankkeen teknisenä arkkitehtina ja projektipäällikkönä. Vedin myös ensimmäisen Joomla!-pohjaisen asiakasprojektin määrittelystä käyttöönottoon asti teknisenä projektipäällikkönä.

SmartTab-sovelluksen osalta vastasin alun perin taustajärjestelmän web services -rajapinnasta teknisesti, siihen liittyvien palvelukuvausten tuottamisesta sekä osittain myös palveluiden toteuttamisesta. Vastuisiini kuului myös varsinaisen tablet-sovelluksen määrittelytyöt sekä muiden toteuttajien ohjaaminen kehityshankkeen edetessä. Hankkeen edetessä roolini kehityksessä on kasvanut edelleen.

Opinnäytetyötä kirjoitettaessa huhtikuussa 2012 olen vastuussa molemmista kehityshankkeista ja niissä tuotettavista ratkaisuista tuotepäällikkönä.

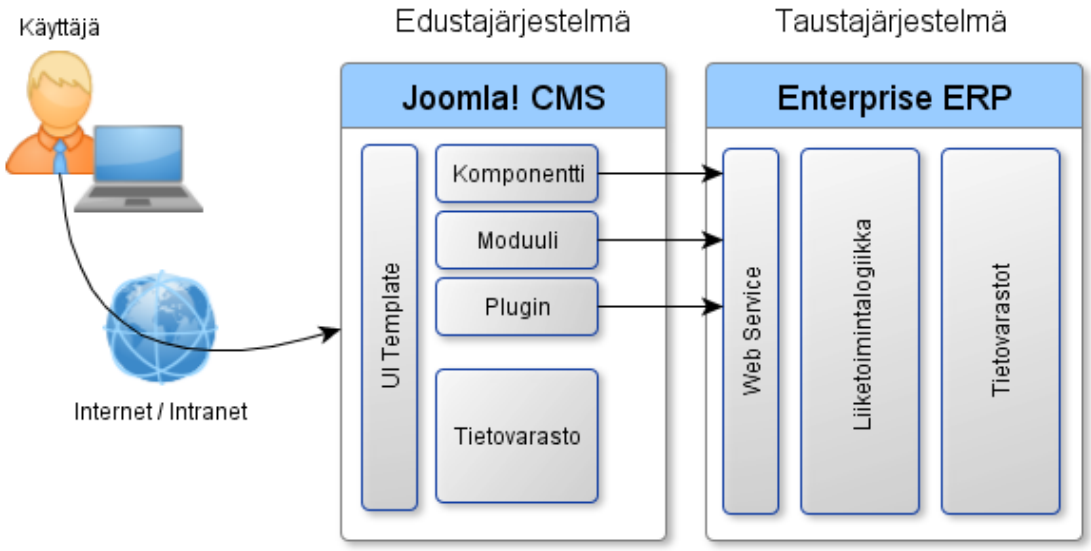
9.2.1 CASE: Joomla CMS

Enterprise-toiminnanohjausjärjestelmän päälle on perinteisesti rakennettu www-sovelluksia täysin Progress OpenEdge WebSpeed -sovellusalustalla ja Progress OpenEdge ABL –ohjelmointikielellä. Käytännössä tällä tavalla on saatu toteutettua toimivia ja skaalautuvia toiminnanohjausjärjestelmään integroituja järjestelmiä, joiden toteuttaminen ei ole vaatinut liian suurta panostusta asiakashyötyyn nähden. Progress OpenEdge teknologian heikkoutena on kuitenkin ollut se, että vaikka samoja työkaluja ja ympäristöjä käyttävät muutkin ohjelmistotalot ympäri maailmaa, on yleiskäyttöisten komponenttien saaminen oman talon ulkopuolelta käytännöllisesti katsoen mahdotonta.

Tästä on seurannut se, että jokainen asiakastarve on ratkaistu omien resurssien voimin ja tätä ratkaisua on pyritty mahdollisuuksien mukaan monistamaan omissa uusissa asiakasprojekteissa joko tuotteistamisella tai räätälöinnillä. Ratkaisut eivät aina ole olleet tyydyttäviä ja joissakin tapauksissa ne ovat olleet myös toteuttamiskustannuksiltaan liian korkeita.

Enterprise Web Services –rajapinta mahdollistaa web-järjestelmien rakentamisen Enterprise toiminnanohjausjärjestelmän yhteyteen myös muilla, kuin Progress OpenEdge teknologioilla. Tähän tarkoitukseen on valittu PHP-ohjelmointikieleen ja MySQL-tietokantaan perustuva vapaanlähdekoodin sisällönhallintajärjestelmä (Content Management System, CMS) Joomla!.

Kuviossa 23 esitetään taustajärjestelmänä toimivan Enterprise toiminnanohjausjärjestelmän ja edustajärjestelmänä toimivan Joomla! sisällönhallintajärjestelmän muodostama järjestelmäympäristö. Ympäristössä käyttäjä on tekemisissä selaimen välityksellä vain edustajärjestelmän kanssa. Käyttäjä ei periaatteessa edes tiedä, että hän on välillisessä vuorovaikutuksessa myös toiminnanohjausjärjestelmän kanssa.



KUVIO 23. Enterprise ERP ja Joomla! CMS järjestelmäympäristö

Ensimmäinen Joomla!–sisällönhallintajärjestelmää hyödyntävä asiakasprojekti aloitettiin vuoden 2012 alussa. Kyseisen projektin puitteissa rakennetaan asiakkaalle www-portaali, joka toimii sidosryhmä- ja asiakasviestinnän kanavana sekä tilausjärjestelmänä, eli verkkokauppana, sopimusasiakkaille.

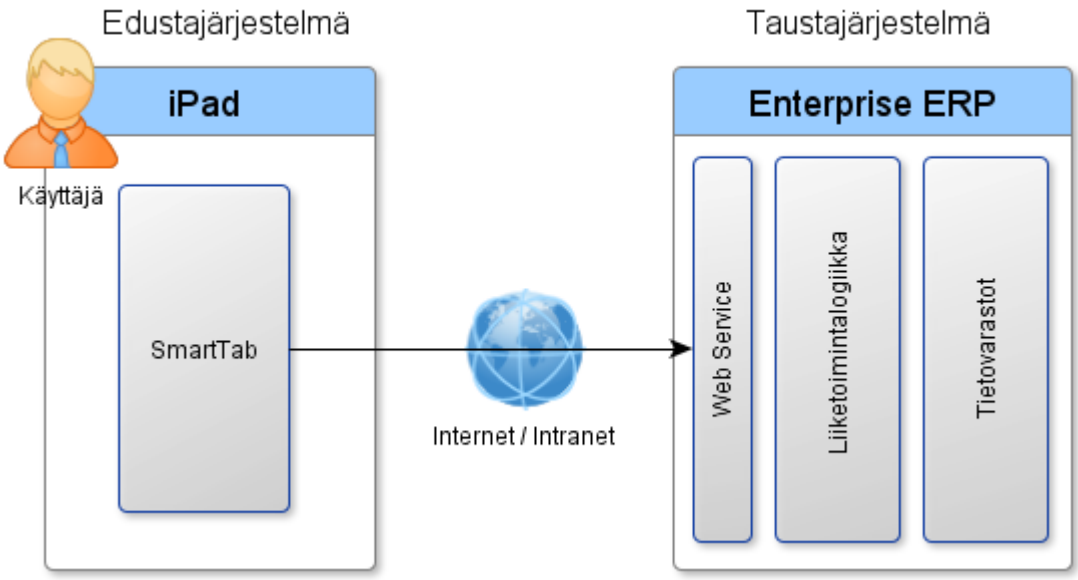
Erona aikaisempiin toteutettuihin järjestelmiin on se, että sisällönhallintajärjestelmän keinoin voidaan portaalissa ylläpitää sellaista tietoa ja suorittaa sellaisia toimia, jotka eivät liity yrityksen toiminnanohjausjärjestelmään. Aiemmin nämä ominaisuudet olisi pitänyt rakentaa osaksi toiminnanohjausjärjestelmää. Lisäksi sisällönhallintajärjestelmän ominaisuuksia voidaan laajentaa käyttämällä ilmaisia tai maksullisia kolmannen osapuolen komponentteja ilman että toiminnanohjausjärjestelmään tarvitsee välttämättä tehdä muutoksia.

9.2.2 CASE: iPad ja SmartTab

Enterprise Web Services –rajapinta mahdollistaa myös täysin uudenlaisten sovellusten ja päätelaitteiden hyödyntämisen osana toiminnanohjausjärjestelmään liittyvää infrastruktuuria. Natiivi ja standardeja noudatteleva rajapinta mahdollistaa ulkoisten välineiden ja sovellusten liittämisen toiminnanohjausjärjestelmään kustannustehokkaasti ja ilman ulkoisia integraatiovälineitä.

SmartTab on Digian mobiilia työskentelyä tukeva sovellusalusta, joka hyödyntää Applen iPad-tablet-tietokonetta. Sovellus toimii itsenäisesti tabletissa ja kommunikoi taustalla toimivan toiminnanohjausjärjestelmän kanssa web service -kutsuilla. SmartTab voi siis rajapinnan avulla hyödyntää taustajärjestelmän tietoja ja liiketoimintalogiikkaa.

Kuviossa 24 esitetään Tablet-tietokoneella toimivan SmartTab-sovelluksen ja taustajärjestelmänä toimivan Enterprise toiminnanohjausjärjestelmän muodostama sovellusympäristö. Ympäristössä käyttäjä käyttää Apple iPad-tablet-tietokoneessa toimivaa SmartTab-sovellusta. SmartTab sovellus kommunikoi taustajärjestelmän kanssa web service –rajapintaa hyödyntäen. SmartTab sovellukset ovat toiminnanohjaukseen liittyviä käyttötarkoitukseen kohdennettuja sovelluksia, joten käyttäjä tietää käyttävänsä välillisesti toiminnanohjausjärjestelmää.



KUVIO 24. Enterprise ERP ja SmartTab järjestelmäympäristö

SmartTab-sovellusalusta ja sovellukset eivät liity pelkästään Enterprise toiminnanohjausjärjestelmään, vaan ne voidaan kytkeä toimimaan muidenkin standardeja web service -rajapintoja tarjoavien toiminnanohjausjärjestelmien kanssa. Enterprise-toiminnanohjausjärjestelmän web service –rajapintaa hyödyntävien edustasovellusten sopiminen myös muiden järjestelmien yhteyteen on hy-

vä esimerkki siitä, että Enterprise-toiminnanohjausjärjestelmään toteutettu rajapinta toimii yleisten standardien mukaisesti.

9.3 Jatko

Tuotteistettu versio Enterprise Web Services –rajapinnasta tullaan julkaisemaan Enterprise-toiminnanohjausjärjestelmän versiossa 4.7.0. Julkaisuun asti rajapintaa kehitetään ja testataan erillisenä hankkeena, mutta julkaisun yhteydessä rajapinnan ylläpito ja kehittäminen siirtyvät osaksi normaalia tuotekehitystä.

Viralliseen julkaisuun liittyy myös tuotetasoisen dokumentaation tuottaminen sekä ulkoiseen että sisäiseen käyttöön. Rajapinnasta tuotetaan oma dokumentaationsa osaksi sekä julkista Enterprise-toiminnanohjausjärjestelmän tuotekuvausta että sisäisiä kehittäjädokumentteja. Näiden lisäksi tullaan tuottamaan ohje rajapinnan mahdollisuuksista ja hyödyntämisestä myyjien, projektipäälliköiden ja muiden asiakasrajapinnassa työskentelevien henkilöiden avuksi.

Julkaisemisen yhteydessä tullaan rajapintaan liittyvä dokumentaatio jalkauttamaan asiakasprojektien käyttöön koulutuksilla. Koulutuksia järjestetään sekä yleisellä tasolla myyjille ja projektipäälliköille sekä teknisellä tasolla kaikille asiakasprojekteihin osallistuville. Jalkauttamisen lisäksi tuotekehitys tulee tukemaan asiakasprojekteja tehostetusti web service –ratkaisuihin liittyvissä asioissa.

Kehityshankkeen aikana nousi esille tarve tuottaa myös kehittäjille yksinkertaisia työkaluja yksittäisten palveluiden tuottamiseksi ja testaamiseksi. Tuotantokelpoisen ympäristön pystyttäminen kehittäjille on liian aikaa vievää ja toistaiseksi vielä erikoisosaamista vaativaa työtä. Enterprise-toiminnanohjausjärjestelmässä on kuitenkin olemassa sisäänrakennettu Progress Openedge -teknologialla toteutettu www-palvelin, jota voitaisiin periaatteessa hyödyntää rajapinnan alustana. Www-palvelimen käyttöä rajapinnan alustana tutkittiin ja tehtyjen kokeilujen perusteella todettiin, että toiminnot yhdistämällä voidaan toteuttaa yksinkertainen kehittäjille tarkoitettu kehitys- ja testikäyttöön soveltuva väline. Kehittäjän välineiden toteutus päätettiin ottaa osaksi kehitys-

hanketta ja ne tullaan julkaisemaan samassa yhteydessä virallisen rajapinnan kanssa.

Lisäksi jatkossa tullaan tutkimaan edellä mainitun kehittäjille suunnatun välineen hyötykäyttöä myös asiakasprojekteissa, joissa integraatiotarve on vähäinen sekä käyttäjien määrän että integraatioiden määrän suhteen. Tästä saataisiin luonnollinen lisä asiakasprojekteihin, joissa ulkoisten integraatiovälineiden kustannukset ovat tähän asti estäneet pienimuotoisten liittymien tuottamisen. Erinomainen esimerkki tällaisesta tarpeesta on vaikkapa järjestelmän toimintaan liittyvien tapahtumien tuottaminen RSS-syötteenä näytettäväksi ulkoisissa välineissä.

Arkkitehtuurin muodollisen katselmoinnin yhteydessä käsiteltiin uudelleen tarvetta REST-tyyppiseen palveluntarjontaa. REST-tuelle nähtiin selkeitä tarpeita modernien web-sovellusten käyttämien kehysten (framework) yhteydessä. Keskustelujen lopputuloksena tehtiin päätös, että REST-tuen mahdollisuutta tutkitaan ja tuki toteutetaan erillisenä kehityspiirteenä, mikäli se on toteutettavissa kustannustehokkaasti.

Rajapintaa ja sen mahdollisuuksia esiteltäessä nousi esille myös SOAP-tuen ja palveluiden WSDL-kuvausten puuttuminen. Puutteet saattavat aiheuttaa joissain tapauksissa sen, että rajapintaa ei voida hyödyntää halutussa laajuudessa tai että joudutaan tekemään lisätyötä yhteensopivuuden takaamiseksi. SOAP-tuen ja WSDL-kuvausten tarve tullaan tutkimaan erikseen ja niiden osalta tehdään päätös mahdollisesta jatkosta, eli ovatko piirteet tarpeellisia vai eivät.

10 POHDINTA JA JOHTOPÄÄTÖKSET

Työn merkittävyys toimeksiantajalla liittyy pääasiassa Enterprise-toiminnanohjausjärjestelmän asiakasprojekteihin sekä niissä oleviin integraatio tarpeisiin. Integraatiotarpeisiin voidaan laskea sekä järjestelmien väliset perinteiset integraatiot että päätelaitteissa ajettavien sovelluksien liittäminen osaksi toiminnanohjausjärjestelmää.

Tähän asti integraatioita on toteutettu projekti- ja asiakaskohtaisesti ja tehdyt ratkaisut ovat olleet sekä teknisesti että laadullisesti vaihtelevia. Tuotteistetulla ja standardoidulla web service -ratkaisulla pyritään yhtenäistämään asiakasprojekteissa tehtäviä integraatioita sekä teknisesti että laadullisesti – yhtenäinen toimintapa ja tekniikka tuovat säästöjä sekä käytettävän työajan että kustannusten puitteissa.

Aiemmin projekteissa on jouduttu käyttämään ulkoisia integraatiovälineitä kaikkiin Enterprise-toiminnanohjausjärjestelmään liittyviin ulkoisiin hyödyntämistarpeisiin. Ulkoinen integraatioväline on voinut olla myös Digian oma tai jonkin 3. osapuolen toimittama. Ulkoisen välineen käyttäminen on tuonut projekteihin lisäkustannuksia lisenssimaksuina ja erillisistä osaajista aiheutuvina kuluina. Toteutettavalla ratkaisulla haetaan kustannussäästöjä minimoimalla ulkoisten integraatioratkaisujen tarve projekteissa – Näin ulkopuolelle maksettavat lisenssikustannukset pienenevät sekä erillisen osaamis- ja työtarpeen määrä pienenee.

Lisäksi toteutettavalla ratkaisulla mahdollistetaan Progress OpenEdge teknologian korvaaminen sellaisissa projektien osatoteutuksissa, jotka eivät varsinaisesti hyödy siitä ja saattavat jollain tapaa jopa kärsiä sen käyttämisestä. Esimerkiksi toiminnanohjausjärjestelmän päälle rakennettavat www-sovellukset voidaan jatkossa toteuttaa tarkoitukseen sopivalla yleisemmällä teknologialla, kuten Java, .Net, PHP tai RubyOnRails. Hyvä esimerkki tästä on Joomla! sisällönhallintajärjestelmän käyttäminen toiminnanohjausjärjestelmään integroidun verkkokaupan alustana. Käytettävissä olevien teknologioiden määrää lisäämällä

saadaan lisättyä myös asiakasprojekteissa ja toiminnanohjausjärjestelmän tuotekehityksessä mahdollisesti käytettävien henkilöiden joukkoa. Tosiasia on, että Progress osaajia on harvassa.

Avoin standardin mukainen rajapinta toiminnanohjausjärjestelmässä antaa mahdollisuudet lähteä tutkimaan ja suunnittelemaan toiminnanohjausjärjestelmän hyödyntämistä kokonaan uusilla tavoilla ja välineillä. Tästä hyvänä esimerkkinä on Digian SmartTab -sovellusalusta, joka mahdollistaa kokonaan uudenlaisten kohdennettujen sovellusten suunnittelun ja toteuttamisen osaksi toiminnanohjausta.

Kehityksen aikana tehtiin myös päätös siitä, että web service –rajapinta tullaan toteuttamaan siten, että se on mahdollisuuksien mukaan otettavissa käyttöön myös vanhemmissa Enterprise-toiminnanohjausjärjestelmän versioissa. Näin maksimoidaan niiden asiakkaiden määrä, joilla on mahdollisuus hyödyntää rajapintaa. Ratkaisun myötä asiakkaita ei myöskään pakoteta tekemään ”turhia” versionvaihtoja ominaisuuden käyttöönottamiseksi.

Varsinainen rajapinta saatiin toteutettua niin, että sen riippuvuus Enterprisen versiosta on mahdollisimman väljä. Toisin sanoen rajapinta voidaan ottaa periaatteessa käyttöön kaikissa nykyisin asiakkailla käytössä olevissa Enterprisen versioissa. Tämä ei kuitenkaan ole realistista, koska rajoituksia käytölle asettaa myös Progress OpenEdge -sovelluskehitysalustan tukemat ominaisuudet. Progress OpenEdge tukee tarvittavia piirteitä joulukuussa 2009 julkaistusta versiosta 10.2B lähtien.

Standardin ja avoimen rajapinnan käyttöönottamisessa toiminnanohjausjärjestelmässä saattaa piillä sellainen Digian kannalta negatiivinen seikka, että se mahdollistaa helpommin myös muiden toimittajien ratkaisujen hyödyntämisen Digian ratkaisujen sijaan. Positiivisempi näkökulma asiaan on se, että Digia voi toimia kuitenkin osajärjestelmän toimittajana, koska Enterprise toiminnanohjausjärjestelmään voidaan liittää tarvittaessa myös 3. osapuolen välineitä tilanteissa, joissa Digia ei pysty kokonaisuudessaan toimittamaan asiakasta tyydyt-

tävää ratkaisua. Näin asiakkaan ei tarvitse vaihtaa koko toiminnanohjausjärjestelmää, joka olisi Digian kannalta huonoin mahdollinen ratkaisu.

Rajapintaa suunniteltaessa, toteutettaessa ja esitellessä oli huomionarvoista se, että joidenkin henkilöiden oli hankalaa ymmärtää eroa web service -rajapinnan ja varsinaisten järjestelmien väliseen integraatioon tarkoitettujen välineiden välillä. Web service -palvelu on kuitenkin passiivinen rajapinta, joka tarjoaa palveluja ulkopuolisille käyttäjille – Web service -palvelu itsessään ei voi olla aktiivinen kommunikaation aloittaja, kuten varsinaiset integraatiovälineet voivat olla. Tästäkin syystä on ensiarvoisen tärkeää, että nyt toteutetun kokonaisuuden dokumentoimiseen ja jalkauttamiseen sekä myynnin että asiakasprojektien käyttöön tulee kiinnittää erityistä huomiota.

Rajapinnan hienoimpia asioita on sen skaalautuvuus hyvin monenlaiseen ja -tasoiseen käyttöön. Sen avulla voidaan toteuttaa järjestelmien välisiin integraatioihin liittyvät tarpeet, sillä voidaan liittää uusia laitteita ja sovelluksia osaksi toiminnanohjausympäristöä tai sillä voidaan toteuttaa jotain hyvin yksinkertaista, kuten esimerkiksi RSS-syöte toiminnanohjausjärjestelmään tulleista tilauksista ja liittää se henkilön Windows-työpöydällä olevaan syötteitä näyttävään gadgettiin.

Asiakkaat ovat ottaneet rajapinnan erittäin hyvin vastaan ja he ovat osoittaneet aitoa kiinnostusta rajapinnan tuomille mahdollisuuksille omissa liiketoimintaympäristöissään. Vaikka kyseessä on periaatteessa tekninen ominaisuus, niin sen asiakkaille tuomia hyötyjä on ollut helppo kuvata ja konkretisoida yksinkertaisilla esimerkeillä ja demoilla. Varsinaista rajapintaa ei ole tämän opinnäytetyön kirjoittamisen aikaan vielä virallisesti julkaistu, mutta sitä on kuitenkin jo alettu aktiivisesti hyödyntämään asiakasprojekteissa ja myynnissä.

LÄHTEET

Abeysinghe A., 2008, RESTful PHP Web Services, UK, Packt Publishing Ltd.

Achimugu P., Babajide A., Gambo I., Oluwagbemi O. & Oluwaranti A. 2010. Software Architecture and Methodology as a Tool for Efficient Software Engineering Process: A Critical Appraisal. Journal of Software Engineering and Applications. Scientific Research Vol.3 No.10, 933- 938

Barry D., 2003, Web Services and Service-Oriented Architectures: The Savvy Manager's Guide, USA, Morgan Kaufmann Publishers

Clements P., Bachmann F., Bass L., Garlan D., Ivers J., Little R., Merson P., Nord R., Stafford J., 2010, Documenting Software Architectures: Views and Beyond, Second Edition, USA, Software Engineering Institute

Digia Enterprise 4.6 Tuotekuvaus. 2011. Digia Oyj (Saatavuus rajattu toimeksiantajan sisäiseen käyttöön ja sovelluslisenssin omaaville)

Digia Enterprise ohjelmointiohjeet. 2012. Digia Oyj (Saatavuus rajattu toimeksiantajan sisäiseen käyttöön)

Digia vuosikertomus 2011. 2012. Digia Oyj. luettu 26.2.2012. http://vuosikertomus2011.digia.com/

IEEE 1471-2000. 2000. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. The Institute of Electrical and Electronics Engineers, Inc. http://ieeexplore.ieee.org/servlet/opac?punumber=7040 (luettu 19.11.2011) (saatavuus rajattu, vaatii tunnukset)

JHS 173 ICT-palvelujen kehittäminen: Vaatimusmäärittely. 2009. JUHTA - Julkisen hallinnon tietohallinnon neuvottelukunta.

Gorton I., 2006, Essential Software Architecture, Germany, Springer-Verlag Berlin Heidelberg

Kopack M., Potts S., 2003, Sams Teach Yourself Web Services in 24 Hours, USA, Sams Publishing

Kruchten P., 1995, The 4+1 View Model of architecture, USA, The Institute of Electrical and Electronics Engineers, Inc. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=469759 (luettu 9.1.2012) (saatavuus rajattu, vaatii tunnukset)

Lukka, K. 2001, Konstruktiivinen tutkimusote. Luettu 19.4.2011. http://www.metodix.com/fi/sisallys/01_menetelmat/02_metodiartikkelit/lukka_const_research_app/

Magal S., Word J., 2009, Essentials of Business Processes and Information Systems, USA, John Wiley & Sons, Inc

Monk E., Wagner B., 2009, Concepts in Enterprise Resource Planning, Third Edition, USA, Course Technology Cengage Learning

OpenEdge Getting Started: ABL Essentials, 2009, Progress Software Corporation

OpenEdge Getting Started: Application and Integration Services, 2009, Progress Software Corporation

OpenEdge Getting Started: Webspeed Essentials, 2009, Progress Software Corporation

Parthasarathy S., 2007, Enterprise Resource Planning (ERP) – A Managerial and Technical Perspective, Intia, New Age International Publishers

Progress Software, OpenEdge 11.0, http://www.progress.com/fi/openedge/application-development-platform (luettu 22.1.2012)

Progress Software, Who We Are, http://www.progress.com/fi/whoweare/at-a-glance.html (luettu 22.1.2012)

Richardson L., Ruby S., 2007, RESTful Web Services, USA: O'Reilly Media, Inc.

Rozanski N., Woods E., 2011, Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, Second Edition, USA; Pearson Education, Inc.

Wikipedia. 2012. Representational state transfer. http://en.wikipedia.org/wiki/Representational_state_transfer (luettu 29.1.2012)

Wikipedia. 2012. Web Service. http://en.wikipedia.org/wiki/Web_service (luettu 29.1.2012)

World Wide Web Consortium (W3C), 2004, Web Service Architecture. http://www.w3.org/TR/ws-arch/ (luettu 30.12.2011)