TAMPEREEN TEKNILLINEN YLIOPISTO

NIKO LAASANEN

ROOLIPOHJAISEN WEB-PORTAALIKÄYTTÖLIITTYMÄN TOTEUTUS TOIMINNANOHJAUSJÄRJESTELMIIN

Diplomityö

Tarkastaja: Dosentti Ossi Nykänen
Tarkastaja ja aihe hyväksytty tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 6.3.2013.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
LAASANEN, NIKO: Roolipohjaisen web-portaalikäyttöliittymän toteutus toiminnanohjausjärjestelmiin
Diplomityö, 64 sivua
Toukokuu 2013
Pääaine: Hypermedia
Tarkastaja: Dosentti Ossi Nykänen
Avainsanat: Toiminnanohjausjärjestelmä, roolipohjainen käyttöliittymä, portaalikäyttöliittymä, käytettävyys, Liferay

Yritysten toiminnanohjausjärjestelmillä on usein takanaan vuosien kehityshistoria, jolloin niistä muodostuu laajoja ja ominaisuuksiltaan kattavia järjestelmiä. Laajuus tuo kuitenkin mukanaan haasteita hyvän käyttökokemuksen tarjoavan ja yleisesti käytettävyydeltään hyvän sovelluksen luomiseen. Suomalaisella toiminnanohjausjärjestelmiä kehittävällä ohjelmistoyrityksellä Oscar Software Oy:llä vastaava tilanne on tullut eteen heidän omissa järjestelmissään, jossa ongelmaa lähestyttiin ideoimalla helppokäyttöistä verkkopohjaista ja rooleihin perustuvaa käyttöliittymää yhteen yrityksen toiminnanohjausjärjestelmistä. Tämän tutkimuksen tarkoituksena olikin selvittää millainen järjestelmän tulisi olla ja kuinka sellainen toteutettaisiin viisaasti myös tulevaisuuden kannalta. Ongelmaa lähestyttiin hyvin käytännönläheisesti kehittämällä käyttöliittymän prototyyppi Liferay-portaalijärjestelmän avulla ja pyrkimällä sitä kautta vastaamaan nousseisiin kysymyksiin. Prototyypin kehityksen lisäksi Oscar Softwarella haluttiin myös selvittää, soveltuuko Liferay hyvin käyttöliittymän alustaksi.

Tämä työ jakaantui kolmeen osaan: prototyypin määrittely, toteutus ja analysointi. Työn alkupuoli keskittyy määrittelyyn käymällä läpi sekä aiheeseen liittyvää teoriaa, että prototyypille asetettuja vaatimuksia. Prototyypin toteutus alkoi pian määrittelyvaiheen aloituksen jälkeen, sillä yrityksen puolesta haluttiin saada konkreettisia tuloksia jo heti työn alkuvaiheessa. Työn loppuosa keskittyy syntyneen prototyypin esittelyyn sekä yleisemmällä, että yksityiskohtaisella tasolla. Analysointi vaiheessa tutkittiin esimerkein kehitystyötä prototyypille Liferay'n avulla ja pohdittiin Liferay'n soveltumista ja etuja käyttöliittymän kehityksen kannalta tutkimuksen aikana ja tulevaisuudessa.

Prototyyppi osoitti roolipohjaisen web-portaalikäyttöliittymän soveltuvan hyvin toiminnanohjausjärjestelmän tueksi, mikäli nämä järjestelmät tarjoavat toiminnallisuutensa rajapintana käyttöliittymälle. Roolipohjaisuus ja käyttö verkkoselaimilla mahdollisti uusien käyttäjien tuomisen toiminnanohjausjärjestelmän piiriin. Lisäksi erottamalla tavallisimpia yksinkertaisia toimintoja toiminnanohjausjärjestelmästä omaan käyttöliittymään, saatiin käyttäjälle turhien näkymien ja toimintojen määrää karsittua merkittävästi. Lisäksi roolipohjaisuus toi mukanaan parantunutta tietoturvaa, kun kaikille käyttäjille ei tarvitse näyttää kaikkea järjestelmästä löytyvää tietoa. Prototyypistä koettiin olevan hyötyä myös toiminnanohjausjärjestelmän kehityksessä, kun järjestelmän kehitys ja parantelu voidaan sen avulla kohdistaa helposti yhteen kokonaisuuteen koko järjestelmän sijasta tuotteistaen samalla uudet käyttöliittymäratkaisut.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
LAASANEN, NIKO: Implementing a role-based web-portal user interface for enterprise management systems
Master of Science Thesis, 64 Pages
May 2013
Major: Hypermedia
Examiner: Adjunct professor Ossi Nykänen
Keywords: Enterprise Resource Planning, role-based user interface, web-portal, usability, Liferay

Enterprise resource planning systems tend to have years worth of development behind them, which has lead to massive products containing plethora of features. However, this magnitude has created challenge for providing great user experience and usability while keeping the variety of features intact. A Finnish software company Oscar Software Oy has already faced the described situation and they have answered this challenge by brainstorming an easy to use, role-based web-portal user interface for one of their enterprise resource planning systems. The purpose of this study was to find out what would this user interface be like and how to design it for an easy extensibility and maintenance in the future. The problem was approached in a very practical way by developing a prototype using Liferay portal platform and answering those question from the experience. In addition, Oscar Software wanted to know whether Liferay is a suitable platform for the user interface.

This study is divided into three distinctive parts: specifying and creating the prototype and analysing the user interface in this context. The first part of the study focuses on the specification through theory and it lays down a few necessary requirements for the prototype. Oscar Software wanted some concrete results as soon as possible, thus the development of the prototype was started after the first specifications were presented. The latter part of the study describes the finished user interface in a general manner as well as in detail. Examples were given for development on Liferay and both pros and cons of this portal system were discussed.

The prototype indicated role-based web-portal user interface being a great supporting software for the enterprise resource planning system, as long as sufficient interface was provided for the portal platform. Role-based user management and support for web browsers made it possible to expand enterprise resource planning system's user base to new sectors. By focusing on some core functionalities and thus limiting the number of irrelevant features and data shown to user, it was possible to provide better user experience and increased security. The prototype also helped to further direct development of Oscar Software's enterprise resource planning systems to easily managed sections.

ALKUSANAT

Tein tämän diplomityön Oscar Software Oy:lle, jonka puolesta työlle annettiin aihe ja tarvittavat resurssit työn suorittamiseen. Tutkimuksen aihe yhdisti sekä verkkoteknologioita että käytettävyyttä, jotka kummatkin ovat olleet minua kiinnostavia aiheita. Pidinkin kovasti työn käytännönläheisyydestä ja mahdollisuudesta osallistua uuden sovelluksen kehittämiseen aivan sen syntyhetkiltä lähtien. Oli myös mukavaa työskennellä selkeästi osana tiimiä käyttöliittymän prototyyppiä kehitettäessä ja päästä käytännössä kokeilemaan tähän mennessä lähinnä luennoilta tuttuja ketteriä kehitysmenetelmiä.

Oscar Softwaren puolelta haluaisin esittää kiitokseni toimitusjohtaja Simo Salmiselle, joka kiireistään huolimatta löysi aikaa lukea ja kommentoida työtäni tämän kuluneen kevään aikana. Lisäksi kiitokset vielä DI Markku Virtaselle työn ohjauksesta sen alkuvaiheessa ja DI Marko Tenkaselle kattavasta näkemyksestä ohjelmistoalasta ja rautaisesta teknisestä osaamisesta erityisesti PL/SQL-osuudessa. Yliopiston puolelta haluan kiittää erityisesti työn tarkastajaa, dosentti Ossi Nykästä, joka jaksoi istua kanssani useissa palavereissa ja jonka esittämät vaativat kysymykset ja aina aiheelliset huomiot työstä ohjasivat sitä vakaasti kohti oikeaa suuntaa.

Lopuksi suurimmat kiitokset vielä vanhemmilleni, Jaana ja Osmo Laasaselle, kannustuksesta ja tuesta koko opintojeni aikana ja joiden luo koulun ja arjen kiireiltä on aina voinut tarvittaessa paeta levähtämään.

Tampereella 10.5.2013.

Niko Laasanen

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

CMS	Content Management System. Sisällönhallintajärjestelmä.
CSS	Cascading Style Sheets. Tyylitiedostokieli, jolla voidaan kustomoida verkkosivun ulkoasua.
ERP-järjestelmä	Tietojärjestelmä, jonka toiminnallisuus kattaa yrityksen toiminnan kaikki osa-alueet (eng. Enterprise Resource Planning System).
JAVA	Yleiskäyttöinen ja laitteistoriippumaton oliopohjainen ohjelmointikieli.
JSON	JavaScript Object Notation. Yksinkertainen, ohjelmointikieliriippumaton tiedonsiirtomuoto.
JSP	Java Server Pages. Javaan pohjautuva teknologia HTML- ja XML-tiedostojen luontiin.
Kustomointi	Järjestelmän muokkaaminen kunkin asiakkaan tarpeita vastaavaksi (eng. customization).
PL/SQL	Procedual Language/Structured Query Language. Oraclen tietokantaohjelmointikieli.
Portaali	Verkkopalvelu, joka sisältää omien toimintojensa lisäksi pääsyn myös muihin palveluihin.[1]
Portaalisovelma	Portaalijärjestelmän osa, jota käytetään tietyn dynaamisen sisällön esittämiseen (eng. portlet).[2]
RBAC	Role-based Access Control. Roolipohjainen käyttöoikeuksien hallinta.
Roolipohjaisuus	Käyttäjien jakaminen eri rooleihin ja näiden roolien hyödyntäminen sisällön esittämisessä (eng. role-based).
SaaS	Software as a Service. Ohjelmiston hankkiminen palveluna.
Sivupohja	Tiedosto, joka sisältää osan verkkosivun näytettävästä sisällöstä (eng. template).
SOAP	Simple Object Access Protocol. Proseduurien kutsun etänä mahdollistava tietoliikenneprotokolla.
Verkkopalvelu	Verkkosivuston kautta tarjottava palvelu.[3]
Web-sisällönhallinta	Verkkopalvelun sisällön sekä sen määrän ja elinkaarena hallinta (eng. web content management).

1 Tietotekniikan termitalkoot, 2007-12-14 portaali
2 Tietotekniikan termitalkoot, 2005-11-25 portaalisovelma
3 Tietotekniikan termitalkoot, 2012-01-10 verkkopalvelu

1 JOHDANTO

1.1 Taustaa

Yritysten toiminnanohjausjärjestelmien (eng. Enterprise Resource Planning, ERP) historia ulottuu jo 1960-luvulle asti, jolloin ensimmäiset yrityksen toimintatapoja optimoimaan pyrkiviä järjestelmiä alkoi ilmestyä markkinoille. Näistä alun ROP-järjestelmistä (Reorder Point) on kehityksen myötä kasvanut yhä monimutkaisempia ja laajempia kokonaisuuksia hallitsevia ohjelmistojättiläisiä, joiden avulla yrityksen toimintaa voidaan tehostaa monella sektorilla. (Jacobs & Weston Jr. 2006, Virtasen 2007 mukaan) Kilpailu ERP-järjestelmiä toimittavien yritysten kesken onkin nykyisin kovaa ja vaihtoehtoisia järjestelmiä löytyy markkinoilta lukuisia[4].

ERP-järjestelmien käyttöönottoon liittyy moninaisia haasteita ja yksi merkittävimmistä on käyttöönoton vaikeus ja jokapäiväisen käytön monimutkaisuus. Toiminnoiltaan hyvin monimutkaiset ja laajat järjestelmät vaativat käyttäjiltään paljon, minkä lisäksi uusi toiminnanohjausjärjestelmä saattaa pakottaa työntekijät käyttämään uusia termejä ja muuttamaan totuttuja toimintatapojaan (Babaian et al. 2005). Käyttäjätyytyväisyys ja käyttökokemus ovatkin nousseet avaintekijöiden joukkoon arvioitaessa ERP-järjestelmien käyttöönoton onnistumista. Liian suuri määrä toimintoja ja epäselvä tai muuten huonosti toimiva käyttöliittymä aiheuttavat helposti negatiivisia reaktioita loppukäyttäjässä, mikä puolestaan synnyttää muutosvastaisuutta. Tämä on ERP-järjestelmien kohdalla nostanut esiin tarpeen helppokäyttöisille liittymille, jotka tiukassa kilpailutilanteessa saattavat olla se ratkaiseva tekijä uutta järjestelmää hankittaessa (Calisir & Calisir 2004.)

Toinen merkittävä ongelma ERP-järjestelmien käytössä liittyy niiden toteutukseen tyypillisesti perinteisinä työpöytäsovelluksina ja siitä aiheutuvana vaatimuksena työalustalle. Työpöytäsovellukset toimivat toki hyvin toimistoympäristössä työpäätteiden ääressä työskenneltäessä, mutta tänä päivänä töitä tehdään paljon myös toimiston ulkopuolella, kuten asiakkaiden tiloissa. Jankowska et al. (2006) arvioivat noin 37 prosenttia työtä tekevistä ihmisistä kuuluvan ryhmään, jonka työnkuva on liikkuvaa tai vaatii muuten esimerkiksi mobiililaitteita. Tuoreemmatkin tutkimukset vahvistavat saman, eikä tälle kasvulle ole odotettavissa loppua (Mobile Worker Population 2012).

4 Lista ensisijaisesti toiminnanohjausjärjestelmiksi luokitelluista järjestelmistä [WWW] [Luettu: 1.2.2013] http://www.toiminnanohjaus.fi/index.php?option=com_weblinks&catid=13&Itemid=23

Käytettävyys- ja mobiiliusvaatimuksista johtuen myös ERP-järjestelmiä on alettu toteuttaa portaaliratkaisuina, roolipohjaisina tai jopa täysin selaimen kautta toimivina. Esimerkiksi portaaliratkaisut ovat jo osoittaneet lupaavia merkkejä helpommasta käyttöönotosta sekä parantuneesta laajennettavuudesta ja skaalautuvuudesta monen muun ominaisuuden ohella (Zykov 2003). Monille yrityksille koko olemassa olevan järjestelmän sovittaminen uusiin tekniikoihin tai kokonaan uuden ERP-järjestelmän luominen on kuitenkin liian paljon aikaa ja resursseja vievä toimenpide. Tässä tutkimuksessa olikin tarkoituksena luoda jo olemassa olevan ERP-järjestelmän päälle selaimella toimiva käyttöliittymä, joka yhdistäisi portaali- ja roolipohjaisten ratkaisuiden hyviä puolia. Käytännössä käyttöliittymä pyrittiin toteuttamaan kohdennettuna verkkopalveluna, jota voi käyttää erilaisilla päätelaitteilla selaimen kautta.

1.2 Tutkimusasetelma

Tutkimuksen lähtökohtana on aiheen esittäneen yrityksen Oscar Software Oy:n halu kehittää omia ERP-järjestelmiään entistä käytettävämpään ja asiakkaita paremmin palvelemaan muotoon. Oscar Softwaren nykyiset järjestelmät ovat kuitenkin olleet kehitteillä jo yli kymmenen vuoden ajan, joten suurten muutosten tekeminen kohtuullisessa ajassa ominaisuuksiltaan jo varsin massiivisiksi paisuneisiin järjestelmiin on käytännössä mahdotonta. Ratkaisuna lähdettiinkin ideoimaan verkkoselaimen kautta käytettävää liittymää ERP-järjestelmään.

Oscar Softwaren sisällä huomattiin roolipohjaisuuden ja portaalitoteutusten olevan lupaavia tekniikoita helppokäyttöisen käyttöliittymän toteutukseen, joten nämä haluttiin sisällyttää tutkimukseen. Lisäksi asiakasprojektien kautta löydettiin mahdollisuus laajentaa asiakaskuntaa myös toimistojen ulkopuolelle, esimerkiksi asiakkaiden luona käyviin myyjiin tai vaikkapa korjaustöitä tekeviin huoltohenkilöihin. Tästä tarpeesta profiloitui tutkimuksen aikana toteutetun prototyypin luonne. Prototyypillä haluttiin Oscar Softwarella saada aikaan jokin konkreettinen todiste alkuperäisen idean toimivuudesta ja toteutuskelpoisuudesta. Jos jo tällä prototyyppikäyttöliittymällä voitaisiin luoda tarkkaan kohdennettuja toimintoja tietyille käyttäjille ERP-järjestelmään riisuen käyttöliittymä samalla kaikesta ylimääräisestä ja tarjoamalla kaikki tämä myös mobiiliutta tukevassa muodossa, voitaisiin jatkossa todennäköisesti laajentaa käyttäjäkunta toimistotyöntekijöiden ulkopuolelle.

Tutkimuskysymys rakentuukin kahdesta osasta. Ensimmäinen kysymys vastaa siihen, miten tällainen käyttöliittymä tulisi toteuttaa ja millaisia ominaisuuksia se voisi sisältää. Tähän olennaisesti liittyy tarvittavan rajapinnan ja prototyypin keskeisten toimintojen määrittely. Toinen tutkimuskysymys kertoo, soveltuuko työn aikana Liferay-portaaliympäristöön toteutettu prototyyppiliittymä käyttöliittymälle esitettyihin vaatimuksiin. Tavoitteena on siis saada selville, voidaanko tällaista järjestelmää ylipäänsä toteuttaa ja onko esimerkiksi valittu alusta siihen hyvin soveltuva. Lisäksi vastauksia pyritään

löytämään myös jatkokehityksen kannalta mielenkiintoisiin kysymyksiin, kuten kuinka helppoa asiakaskohtainen muokkaus tulee olemaan ja millaista ylläpitotyötä luotu järjestelmä vaatii. Tutkimuskysymyksiin haettiin vastauksia pitkälti prototyypin kehityksen kautta.

1.3 Työn suoritus

Tämän työn suoritus jakaantuu käytännössä kolmeen osaan: prototyypin määrittely, toteutus ja analysointi. Koska Oscar Softwaren puolelta prototyypin varsinainen toteutus haluttiin mahdollisimman pikaisesti käyntiin, etenee työ vahvasti toteutus edellä. Käytännön tekemisessä pyrittiin mahdollisuuksien mukaan soveltamaan ketteriä menetelmiä, jotta määrittelyjä voitiin tarkentaa iteratiivisesti prototyypin kehityksen aikana.

Määrittelyvaiheessa tärkeänä tietolähteenä olivat erilaiset alan artikkelit, tutkimusraportit ja myös käytettävyyden perusteokset. Näistä pyrittiin selvittämään, millaisia ominaisuuksia nykyaikaisilta ERP-järjestelmiltä vaaditaan ja mitä helppokäyttöisyys ohjelmistoissa merkitsee käytännössä. Lisäksi tärkeässä osassa olivat Oscar Softwarelta saatu tieto siitä, mitä heidän asiakkaansa toivovat järjestelmiltä, jotta ydinominaisuudet voidaan määritellä oikein. Prototyypin määrittely alkoi myös heti projektin alkuvaiheessa ja sen ominaisuuksia pyrittiin iteratiivisesti kehittämään sitä mukaa, kun tutkimuksessa rakentui parempi kuva siitä, millainen käyttöliittymän tulisi olla.

Toteutusvaihe alkoi siis heti määrittelyn kanssa rinnan ja tavoitteena oli saada melko nopeasti valmiiksi ominaisuuksiltaan karsittu versio. Prototyypin tuli kuitenkin sisältää ainakin tärkeimmät ydinominaisuudet, jotta sen avulla tutkimuskysymyksiin vastaaminen oli mahdollista. Prototyyppi tehtiin Oscar Softwarelle, joka tarjosi myös resursseja työn suoritukseen. Tarkoituksena ei kuitenkaan ole tässä tekstissä esitellä seikkaperäisesti koko syntynyttä prototyyppiä teknisellä tasolla, vaan ennemminkin keskittyä sen ydinideoihin ja toimivuuteen käytännössä.

Analysointivaiheessa arvioitiin, kuinka tutkimuksen aikana syntynyt prototyyppi vastasi alun määritelmiin ja tutkimuskysymyksiin. Lisäksi prototyypin luonnin aikana esiin nousseita huomionarvoisia seikkoja kerrattiin ja samalla pyrittiin nostamaan esille prototyypin onnistumisia ja epäonnistumisia. Analyysin – ja toisaalta koko työnkin – tarkoituksena on toimia päätöksenteon apuna yrityksille, jotka suunnittelevat prototyypin kaltaisten liittymien luontia omiin ERP-järjestelmiinsä. Lisäksi työ sisältää jonkin verran käytettävyyden kannalta hyväksi havaittuja konventioita, joista voi olla hyötyä muidenkin verkkopohjaisten portaali- ja roolipohjaisten ratkaisujen luonnissa.

1.4 Työn rakenne

Tämä työ on jaettu kolmeen suurempaan kokonaisuuteen: teoriaosuuteen, toteutetun prototyypin tarkasteluun ja tutkimuskysymyksiin vastaamiseen. Luvut yhdestä viiteen käsittelevät teoriaosuutta. Näistä ensimmäisessä kerrotaan hieman tutkimuksen taustois-

ta, suorituksesta ja Oscar Softwaresta. Toisessa luvussa kerrotaan hieman yleisesti käytettävyydestä ja käydään läpi myös suunnittelun kannalta hyvän käyttöliittymän vaatimuksia. Luvussa kolme käsitellään web-portaaleja määritelmän ja jaottelun tasolla. Samassa luvussa käydään roolipohjaisia portaaleja vielä tarkemmin läpi ja myös portaalijärjestelmissä paljon käytettyjen portaalisovelmien käsitettä avataan hieman. Neljännessä luvussa käsitellään lyhyesti toiminnanohjausjärjestelmiä erityisesti siltä osin, miten ne liittyvät toteutettavaan prototyyppiin. Viides ja viimeinen teorialuku keskittyy web-sisällönhallintajärjestelmiin. Siinä käydään läpi hieman näiden yleisiä piirteitä, sekä esitellään prototyypin alustana toimiva Liferay-portaalijärjestelmä ja Oscar Softwaren oma sisällönhallintajärjestelmä, Oscar CMS.

Prototyypin tarkastelu alkaa luvusta kuusi, jossa sitä käydään läpi lähtien yleiskatsauksesta ja vaatimuksista aina tärkeimpien roolien ja toimintojen kuvaukseen. Seitsemäs luku sisältää puolestaan prototyypin tarkastelun tarkemmin teknisemmältä kannalta käytännön esimerkkejä hyödyntäen. Lisäksi seitsemännessä luvussa on käyty tutkimuksen tuloksia läpi myös arvioinnin kannalta. Kahdeksannessa ja samalla viimeisessä luvussa kootaan yhteen tutkimuksen aikana havaitut oleelliset asiat ja pohditaan hieman prototyypin vaikutuksia myös tulevaisuuteen Oscar Softwaren kannalta.

1.5 Oscar Software Oy

Oscar Software Oy[5] on suomalainen, Tampereella päätoimistoaan pitävä, ohjelmistoyritys. Oscarilla ollaan keskitytty yritysten tietojärjestelmiin ja niistä ennen kaikkea toiminnanohjausjärjestelmiin, joita Oscar Softwaren tuotevalikoimasta löytyy useampiakin. Toiminnanohjausjärjestelmien lisäksi Oscar Software tekee erilaisia näihin liitettäviä järjestelmiä, kuten verkkokauppoja, ja tarjoaa myös palveluita esimerkiksi taloushallinnon muodossa.

Päätuotteita Oscarilla on kolme: Oscar PRO, Oscar Tisma ja Oscar Center. Kaikki kolme ovat toiminnanohjausjärjestelmiä, mutta jokainen on lähtökohdiltaan ja tavoitteiltaan hieman toisistaan poikkeava ja erilaisille yrityksille tarkoitettu. Oscar PRO on yrityksen koko liiketoiminnan hallintaan keskittyvä laaja järjestelmä, joka mahdollistaa monipuolisen tuotannonohjauksen ja projektien hallinnan yhdellä ohjelmistolla. Oscar Tisma on ennen kaikkea kaupan- ja palvelualan sektorille kohdennettu ERP-järjestelmä. Oscar Center puolestaan keskittyy erityisesti kehittyvien pienyritysten toiminnanohjaukseen ja se on suosittu muun muassa huollon toimialalla.

Tässä tutkimuksessa Oscar Software Oy toimii siis aiheen antajana ja tarjoaa resurssit työn suoritukseen. Työn avulla Oscarilla toivotaan löydettävän uusia mahdollisuuksia tuotekehitykseen ja asiakkaille entistä paremman käyttökokemuksen tarjoamiseen.

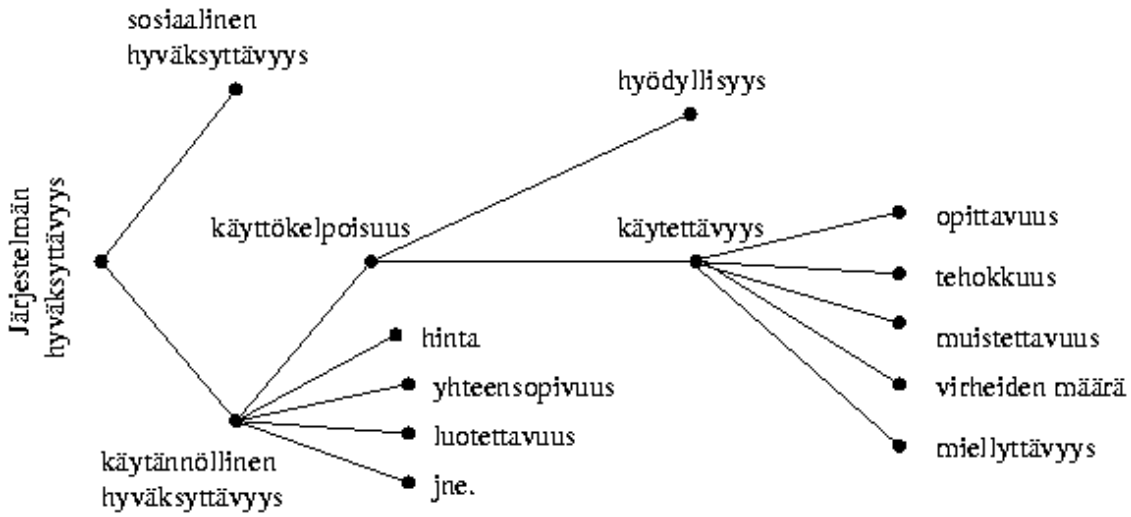
5 Oscar Software Oy:n verkkosivut [WWW]: http://www.oscar.fi

2 KÄYTETTÄVYYS

2.1 Määritelmä ja merkitys

Käytettävyyden (eng. usability) käsitteelle on olemassa useita eri määrittelyjä. Esimerkiksi standardissa ISO 9241-11 (1998), käytettävyyden on määritelty tarkoittavan tietyssä käyttöympäristössä tehokkuutta, tarkoituksenmukaisuutta ja tyytyväisyyttä, jolla käyttäjät saavuttavat tavoitteensa. Tähän listaan voitaisiin lisätä myös käytön helppous (Kuoppala et al. 2004, Järvelän 2012 mukaan), vaikka se mitä helppoudella tarkoitetaan, voi olla hyvin subjektiivista. Kuutti (2003) määrittelee puolestaan käytettävyyden vastaavan käytön sujuvuutta ja ihmisen ja koneen vuorovaikutusta (Human-Computer Interaction, HCI), vaikka toteaa ISO standardin vastaavan hyvin nykypäivän vaatimuksia.

Käytettävyyden määritelmä ei siis suinkaan ole yksiselitteinen. Toinen yleinen lähtökohta määrittelyyn on listata käytettävyyden osa-alueita. Varsin tunnettu onkin käytettävyyden esittäminen kuvan 1 mukaisesti osana puumaista rakennetta, jossa käytettävyys jaetaan vielä viiteen laatuattribuuttiin: opittavuus, tehokkuus, muistettavuus, virheiden määrä ja miellyttävyys. Tällaisten attribuuttien käyttö helpottaa osaltaan käytettävyyden suunnittelua ja arviointia, kun voidaan keskittyä vastaamaan tiettyihin kysymyksiin.



Kuva 1: Käytettävyys osana järjestelmän hyväksyttävyyttä [Nielsen 1993, Immosen 2001 mukaan]

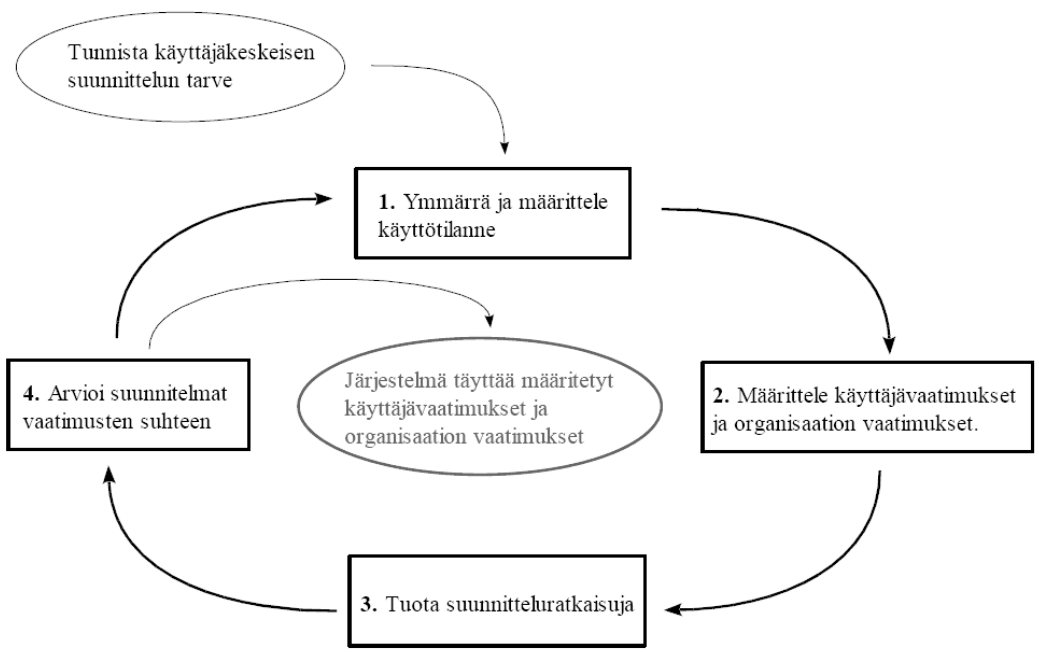
Ohjelmistotuotannon ala alkaa olla jo varsin kypsässä iässä, jolloin kilpailua esimerkiksi tuotannonohjausjärjestelmien kesken on paljon ja monet näistä sovelluksista ovat ominaisuuksiltaan monin paikoin vastaavia. Erityisesti tällaisessa tilanteessa käytettävyys voi olla se valtti, jolla kilpailuetu käännetään oman yrityksen puolelle. (Kuutti 2003) Ohjelmistoissa käytettävyys on kehityksen aikana usein sivuosassa ja käytettävyyttä aletaan kehittämään vasta ohjelman valmistuttua, vaikka on huomattu käytettävyyteen panostamisen tuovan säästöjä (Kalimo 1996).

Toiminnanohjausjärjestelmät eivät ole poikkeus sääntöön käytettävyyden merkitystä pohdittaessa – pikemminkin päinvastoin. Järjestelmien ollessa lähtökohtaisesti hyvin laajoja ja monipuolisia voidaan menettää suuri osa mahdollisesta hyödystä, mikäli tuotteen käytettävyys on huono. Tällöin moni ominaisuus voi jäädä käyttämättä, oppimiskustannukset lisääntyvät, asiakaspalvelu kuormittuu avustuspyynnöistä ja asiakkaan lopulta turhautuessa, saatetaan koko asiakkuus menettää (Kalimo 1996.)

2.2 Käyttäjäkeskeinen suunnittelu

Koska käytettävyys on nykyään niin oleellinen osa ohjelmistoja, ei ole suotavaa erottaa käyttöliittymää suunnittelun kannalta täysin erilliseksi osaksi. Käytettävyyden toteutumiseksi voidaan aihetta lähestyä useammalla eri tasolla, joista ensimmäinen on suunnittelun aikana käytettävyyden pitäminen mielessä. Toisena tasona voidaan pitää käytettävyyden huomioivien suunnitteluperiaatteiden käyttöä ja kolmantena yksittäisten sääntöjen tai muistilistojen läpikäyntiä. (Kalimo 1996) Ensimmäisellä tasolla siis ollaan lähinnä tietoisia, että käytettävyys voisi olla hyvä huomioida, toisella käytettävyys otetaan osaksi suunnittelua ja kolmannella joko hiotaan yksityiskohtia tai koitetaan paikata suunnittelun puutetta.

Käytettävyyttä suunniteltaessa voidaan ottaa lähtökohdaksi käyttäjäkeskeinen suunnittelu (User-Centered Design, UCD), joka koostuu neljästä periaatteesta: käyttäjien aktiivisesta osallistumisesta ja vaatimusten ymmärtämisestä, käyttäjän ja teknologian välisten toimintojen kohdentamisesta, suunnitteluratkaisujen iteroinnista sekä monialaisesta suunnittelusta (Järvelä 2012). Iteratiivisen prosessin rakenne voidaan nähdä kuvassa 2, jossa koko prosessi alkaa tunnistamalla ensin tarve käyttäjäkeskeiselle suunnittelulle. Tämän jälkeen siirrytään iteraation ensimmäiseen vaiheeseen, eli käyttötilanteen määrittelyyn, joista johdetaan sekä käyttäjien että yrityksen vaatimukset. Kun vaatimukset on selvitetty, voidaan tuottaa suunnitteluratkaisuja, joita arvioidaan vaatimuksia vasten. Mikäli suunnitellut ratkaisut täyttävät niille asetetut vaatimukset käyttäjien ja yrityksen taholta, voidaan suunnittelu todeta onnistuneeksi. Muussa tapauksessa siirrytään seuraavaan iteraatioon, joka lähtee jälleen käyttötilanteen tarkentamisesta.



Kuva 2: Iteratiivinen käyttäjäkeskeinen prosessimalli (ISO 13407, Vestolan 2007 mukaan)

Käyttäjäkeskeisessä suunnittelussa hyödyllistä on käyttäjään tutustuminen, jotta käyttötilanne voidaan määritellä. Suunnittelija voivat esimerkiksi käydä loppukäyttäjien työpaikoilla tarkkailemassa normaalia työnkulkua. Käyttäjiltä on tällöin helppo selvittää miksi ja miten asioita tehdään ja toisaalta oikeilta työpaikoilta voidaan saada suunnittelun tueksi tarpeellisia termejä tai vaikkapa käyttäjien artefakteja. Käyttäjien luona vierailu ei kuitenkaan suoraan takaa hyvää käytettävyyttä, sillä siltikin tarvitaan hyvin tarkka kuvaus käyttäjästä ja tämän tavoitteista (Järvelä 2012.)

Säännöt ja muistilistat tulevat käytettävyyttä parannettaessa usein tarpeeseen, jotta oikeisiin asioihin osataan kiinnittää huomiota. Tällaisia listauksia löytyy kirjallisuudesta lukuisia ja ne vaihtelevat pituudeltaan muutamista säännöistä aina satojen sääntöjen mittaisiin listauksiin (Kalimo 1996). Käyttäjäkeskeisessä suunnitteluprosessissa käytettävyyden sääntökokoelmia hyödynnetään erityisesti arviointivaiheessa, jolloin voidaan tehdä heuristista arviointia.

2.3 Arviointi

Käytettävyyden arviointia tarvitaan sekä selvittämään, onko käytettävyyden tuomisessa tuotteeseen onnistuttu, että käytettävyyspuutteiden löytämiseen. Suunnitteluprosessin aikana käytettävyyden arvioinnilla voidaan myös löytää useammista suunnitteluvaihtoehdoista parempi ja välttää tiettyyn ratkaisuun sitoutumista. Suunnitteluvaiheen arviointiin pätee lisäksi sama totuus kuin esimerkiksi testaukseen ohjelmistotuotannossa: mitä aikaisemmassa vaiheessa arviointia tehdään, sitä tehokkaampaa ja korjausten kannalta halvempaa se on.

Arvioinnin avulla voidaan parantaa useita ohjelmiston ominaisuuksia, kuten käytön tehokkuutta, virheiden määrä tai käyttökustannuksia. (Kalimo 1996) Varsinainen arviointi voidaan jakaa esimerkiksi asiantuntija-arviointiin ja käyttäjäpohjaisiin menetelmiin (Kalimo 1996).

2.3.1 Heuristinen arviointi

Asiantuntija-arviointi suoritetaan usein heuristisena arviointina, joka pohjautuu joihinkin käytettävyyden heuristiikkoihin. Käytetyimpiin heuristiikkalistoihin kuuluu Nielsenin lista, joka on esitetty taulukossa 1. Listojen etuna on niiden helppo käyttö, sillä käyttäjää ei tarvita mukaan testeihin ja asiantuntija-arviointi nimestä huolimatta voi kokematonkin henkilö suorittaa arvioinnin. (Kuutti 2003) Arvioijalta vaaditaankin lähinnä tietoja sovelluksen käsitteistöstä ja periaatteista, sekä ainakin pohjatietoja käytettävyydestä.

Taulukko 1: Nielsenin lista (Molich & Nielsen 1990, Kuutin 2003 mukaan).

1	Vuorovaikutuksen käyttäjän kanssa tulee olla yksinkertaista ja luonnollista.
2	Vuorovaikutuksessa tulee käyttää käyttäjän kieltä.
3	Käyttäjän muistin kuormitus tulee minimoida.
4	Käyttöliittymän tulee olla yhdenmukainen.
5	Järjestelmän tulee antaa käyttäjälle kunnollista palautetta reaaliajassa.
6	Ohjelmassa ja sen osissa tulee olla selkeät poistumistiet.
7	Oikopolkuja ja tehokasta työskentelyä tulisi tukea.
8	Virheilmoitusten tulee olla selkeitä ja ymmärrettäviä.
9	Virhetilanteisiin joutumista tulisi välttää.
10	Käyttöliittymässä tulee olla kunnolliset avustustoiminnot ja dokumentaatio.

Arviointi on kuitenkin vain yhden ihmisen mielipide, joten näkemys on väistämättä subjektiivinen. Heuristiikkojen käytöllä, sekä yhdistämällä useamman arvioijan tuloksia, voidaan lopputulosta pitää kuitenkin jopa lähes täysin objektiivisena. Heuristinen arviointi sopii hyvin jo aikaiseen vaiheeseen suunnitteluprosessia tai tilanteeseen, jossa lopullinen kohderyhmä ei vielä ole kunnolla selvillä tai käyttäjätestausta ei heillä voida tehdä.

2.3.2 Käyttäjätestaus

Käyttäjäpohjaisissa menetelmissä lähtökohtana on, että tuotteen loppukäyttäjä suorittaa testauksen. Useimmiten tällaisessa käyttäjätestauksessa käyttäjille on luotu valmiiksi jokin lista tehtävistä, jotka käyttäjän tulee suorittaa joko valmiilla tuotteella tai sen prototyypillä.

Heuristiseen arviointiin verrattuna käyttäjätestausta ei voida suorittaa aivan yhtä aikaisessa vaiheessa tai ainakaan saatavat tulokset eivät tällöin ole yhtä hyödyllisiä. Käyttäjätestaus voi olla myös hankalaa tai kallista järjestää, sillä sekä testauksen suunnittelu että testihenkilöiden kokoaminen voi olla haasteellista. Käyttäjätestaus tuottaa onnistuessaan kuitenkin arvokasta tietoa ja täydentää erinomaisesti heuristisia arviointimenetelmiä (Kuutti 2003.)

Erityisiä haasteita käyttäjätestaukseen asettaa luonnollisen testaustilanteen järjestäminen. Tarkkailtavana oleminen jo itsessään voi häiritä käyttäjää tai aiheuttaa tavallisesta käyttötilanteesta poikkeavia reaktioita. Paras tilanne olisi pyrkiä tekemään testausta mahdollisimman luonnollisessa käyttökontekstissa, mutta myös käytettävyyslaboratoriota voidaan hyödyntää testauksen suorituksessa.

2.4 Hyvän käyttöliittymän vaatimuksia

Hyvälle käyttöliittymälle ei ole olemassa yhtä selkeästi määriteltyä listaa vaatimuksista, jotka tulisi täyttää. Erilaisia sääntökokoelmia ja listoja on kuitenkin tarjolla runsaasti ja ohjelmistoteollisuudessa usein jopa kohdealustakohtaisesti. Suunnittelun apuna voidaan käyttää esimerkiksi Shneidermanin kahdeksaa kultaista sääntöä, joita voidaan pitää vaatimuksina hyvälle käyttöliittymälle. Vaikka nämä säännöt itsessään ovat jo vanhoja, pätevät ne yhä nykyaikaisiin käyttöliittymiin, vaikkakin esimerkiksi web ohjelmissa selaimen vastuulle jää joidenkin sääntöjen toteuttaminen.

Sääntöjen mukaan käyttöliittymän tulee olla yhtenäinen kautta ohjelman niin toimintojen kuin terminologiankin kannalta. Käyttäjän täytyy siis voida navigoida järjestelmässä yhtenäisen logiikan mukaan ja esimerkiksi samoin nimettyjen painikkeiden tulee johtaa järjestelmässä samoihin toiminnallisuuksiin. Käyttäjälle pitäisi myös antaa palautetta kaikista suorittamistaan toiminnoista, kuten vaikkapa tiedoston tallentamisesta tai prosessin valmistumisesta. Tämä ei kuitenkaan tarkoita, että jokaisesta pienimmästäkin tapahtumasta, kuten napin painalluksesta tarvitsisi antaa tekstuaalista palautetta käyttäjälle, vaan esimerkiksi klikkauksesta kertova pieni ääni tai napin animointi voi olla riittävä. Usein sanotaan, että hyvä käyttöliittymä ei mahdollista virheitä ja osa Shneidermanin säännöistä kuvastavat hyvin myös tätä. Käyttäjä ei nimittäin saisi tehdä ainakaan vakavia virheitä ja pienempien virheiden tapahtuessa tulisi tarjota helppo ratkaisu ja tarjota aina paluu aikaisempaan tilanteeseen, jossa virhettä ei vielä ollut syntynyt. Järjestelmän tulisi myös vahvistaa käyttäjän tunnetta siitä, että tämä hallitsee systeemiä, eikä toisinpäin. Lisäksi käyttäjän muistia ei saisi turhaan kuormittaa, jolloin kerran nähty tärkeä tieto tulisi aina pitää saatavilla. (Shneiderman 1987, Kuutin 2003 mukaan) Tätä vahvistaa muun muassa niin sanottu Millerin laki (Miller 1956), jonka mukaan ihminen kykenee pitämään työmuistissaan keskimäärin seitsemää asiaa kerrallaan.

Nykyään erityisesti verkkopalveluiden käyttöliittymille vaatimuksia asettaa alati kasvava mobiilikäyttö. Erilaisten mobiililaitteiden kirjo onkin valtava ja niiden osalta käyt-

töliittymän tulee huomioida erityisesti pienten ruutujen aiheuttama tilan vähyys, datan syöttöön käytetty tekniikka ja näkymien määrä (Jankowska et al. 2006). Esimerkiksi kosketusnäytöllisellä laitteella tavalliselle hiirikäyttöiselle käyttöliittymälle suunniteltu sovellus voi olla ongelmallinen. Mobiilidatayhteydet ovat myöskin usein merkittävästi hitaampia kuin kiinteät- tai WLAN-yhteydet (Wireless Local Area Network), joten näkymät eivät saisi sisältää turhaa dataa. Laitevalmistajakohtaisesti löytyy usein näiden omia suunnitteluohjeita, jotka auttavat hahmottamaan potentiaalisia alustakohtaisia ongelmia. Esimerkiksi Applen ohjeissa luetellut säännöt vastaavat pitkälti Shneidermanin sääntöjä. Applen sääntöjen mukaan sovelluksen ulkoasun tulisi kuvastaa ja tukea sen toiminnallisuutta, käyttöliittymän tulisi olla yhtenäinen sekä iOS:n että oman nykyisen ja vanhojen versioidensa kanssa. Lisäksi käyttäjän tulisi voida suoraan vaikuttaa toiminnoillaan sovellukseen, toimintojen tulisi johtaa näkyviin tuloksiin ja näin ollen antaa käyttäjälle palautetta. Sovelluksessa tulisi myös käyttää metaforia eli käyttöliittymän toimintoja tulisi liittää arkimaailman kappaleisiin ja lopuksi käyttäjän tulee koko ajan hallita sovellusta, eikä sovellus saa viedä hallintaa käyttäjältä. (iOS Human Interface Guidelines 2012) Muiltakin merkittäviltä mobiililaitteiden valmistajilta, kuten Androidilta (Android User Interface Guidelines) ja Microsoftilta (App Design Process for Windows Phone) löytyy vastaavia ohjeistuksia, joita tulisi noudattaa kehitettäessä sovelluksia erityisesti näille laitteille.

3 WEB-PORTAALI

3.1 Määritelmä

Tietotekniikassa web-portaali on käytettyjen portaaliteknologioiden erikoistapaus. Portaali itsessään tarkoittaa porttia, ovensuuta tai muuta vastaavaa sisäänkäyntiä[6] ja tämä onkin varsin osuva nimitys. Web-portaali voitaisiin siis määritellä esimerkiksi keskitetyksi verkossa toimivaksi palveluksi, joka yhdistää tietoa useammasta eri lähteestä ja jonka kautta kerättyä tietoa voidaan tehokkaasti jakaa isommallekin ryhmälle (Vainio 2012). Portaalia voidaan pitää myös niin sanottuna organisointiperiaatteena, jonka avulla yritykset tai organisaatiot voivat järjestää ja tarjota palvelujansa keskitetysti verkon kautta. Portaaleille on lisäksi tyypillistä asiakas- tai sidosryhmäkeskeinen lähestymistapa palvelun luontiin (Katz 2002.)

Web-portaaleilla pyritään ennen kaikkea helpottamaan loppukäyttäjän käyttökokemusta. Yrityksillä on nykyään lähes poikkeuksetta hyvin laajat tietovarastot käytössään, joko toiminnanohjausjärjestelminä tai jonkinlaisissa tietokannoissa. Portaalien avulla tämä suuri tietomäärä voidaan koota tehokkaasti ja käyttäjän kannalta mukavasti yhteen ja samaan paikkaan, jolloin tiedon käsittely ja jäsentely onnistuu paljon helpommin (Vainio 2012.)

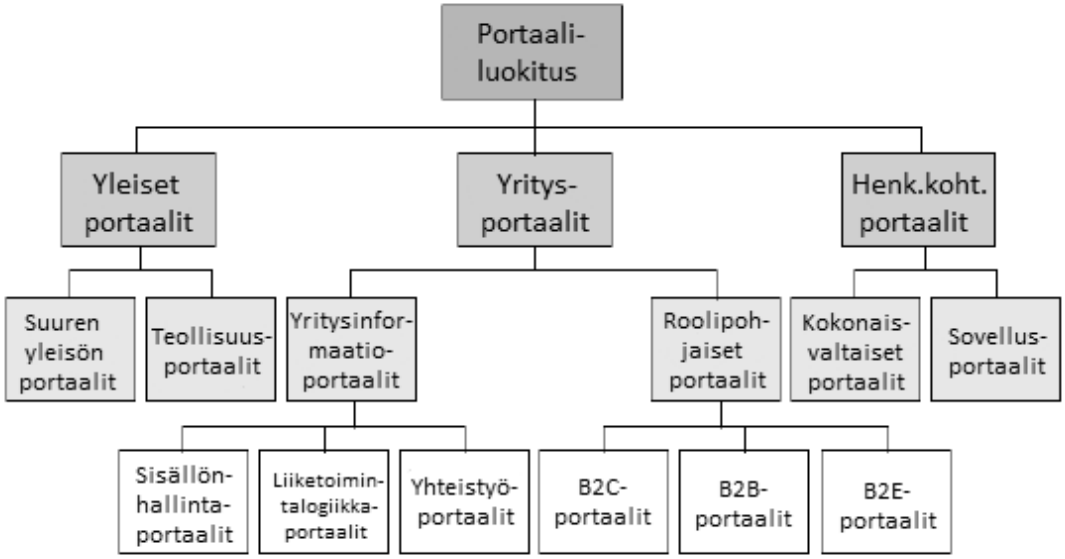
Ensimmäisenä web-portaalit, sellaisina kuin ne nykyään tunnetaan, yleistyivät 2000-luvun alussa yliopistojen tietojärjestelmiin integroituina ratkaisuina (Katz 2002), vaikka portaaleja voidaan katsoa kehitettäneen jo aiemmin 1990-luvun puolella (Vainio 2012). Tämä näkyykin selkeästi myös Suomessa katsottaessa yliopistojen oppimisalustoja tai opiskelijoiden ja henkilökunnan käyttöön tarkoitettuja sivustoja. Jo vuosituhannen alussa pidetyissä Educause-konferensseissa ammatti- ja yliopistojen edustajat pitivät portaalien kehittämistä ja käyttöä välttämättömänä, joskin ongelmiakin löydettiin, erityisesti uuden järjestelmän kehityksen ja opettelun raskaudesta ja löyhän infrastruktuurin aiheuttamista sopimusteknisistä ongelmista (Katz 2002.)

3.2 Portaalityypit

Mitään virallista standardia portaaleille ei ole määritelty, mutta muuta jaottelua portaaleille on kuitenkin tehty. Jaottelut voivat poiketa melko paljon toisistaan, johtuen portaalien mahdollistamasta käyttökohteiden paljoudesta. Kuvassa 3 on vapaasti suomennettu-

6 MOT Oxford Dictionary of English 1.0 s.v. Portal

na esitetty Davydovin käyttämä portaalien hierarkkinen luokittelu. Näistä yleiset portaalit (eng. Public portals) keskittyvät suurten kävijämäärien houkutteluun ja sitä kautta esimerkiksi mainostulojen keräämiseen. Yritysportaalit (eng. Corporate portals) voivat olla joko usein yrityksen sisäiseen toimintaan ja tiedonhallintaan liittyviä ratkaisuja tai vaihtoehtoisesti roolipohjaisia, jotka ovat ennemminkin tuotteisiin tai liiketoimintaan muuten liittyviä alustoja. Henkilökohtaiset portaalit (eng. Personal portals) ovat usein laajasti räätälöitäviä sovelluksia, joihin ei niinkään liity yhteisöllisiä piirteitä. (Vainio 2012) Tässä tutkimuksessa keskitytään erityisesti yritysportaaleihin ja tämän alityypeistä roolipohjaisiin portaaleihin, jotka mahdollistavat web-portaalien hyödyntämisen muun muassa toiminnanohjausjärjestelmissä.



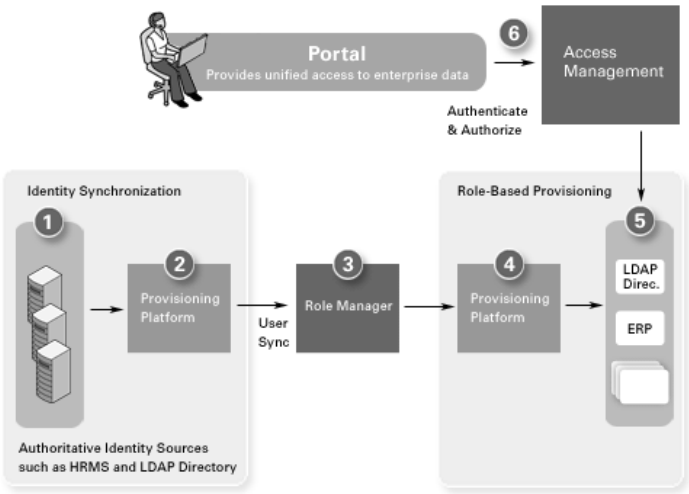
Kuva 3: Portaalityyppien hierarkkinen luokittelu 3x2x2 rakenteella (Davydov 2001, Vainion 2012 mukaan).

Työn aikana toteutettava prototyyppi voitaisiin osin lukea myös yritysinformaatioportaalien (eng. Enterprise information portals) ryhmään kuuluvaksi, sillä Vainion (2012) mukaan tällaisilla järjestelmillä tarkoitetaan yrityksen muihin tietojärjestelmiin linkittyvää ohjelmaa, jonka avulla tietoa saadaan kerättyä yhteen paikkaan esimerkiksi liiketoimintapäätöksiä tukemaan. Näiden kahden portaalityypin erottaminen täysin toisistaan on toisaalta erikoista, sillä yrityksenkin sisällä on monesti tarvetta rajoittaa pääsyä tietoon riippuen työntekijän asemasta. Johtotason henkilöstö haluaa usein päätöksentekoa tukevaa informaatiota, asiakkaiden luona paljon työskentelevät myyjät saattavat kaivata reaaliaikaista tuotetietoa ja esimerkiksi huoltohenkilöt voivat haluta selata työlistaansa (Jankowska et al. 2006). Tällöin roolipohjaisuus voisi hyvinkin tulla kyseeseen myös yrityksen sisäisissä järjestelmissä.

3.2.1 Roolipohjaiset portaalit

Roolipohjaisuus portaaleissa tarkoittaa lähinnä pääsynhallintaa portaalin sisältämiin resursseihin ja näkymien generointia käyttäjän rooliin pohjautuen. Se voidaankin käytännössä samaistaa roolipohjaiseen käyttöoikeuksien hallintaan (eng. Role-based access control, RBAC). Roolipohjainen käyttöoikeuksien hallinta jakaa käyttäjä rooleihin näiden toimenkuvaan liittyvien vastuiden tai pätevyyksien mukaan. Järjestelmään tallennetuille rooleille asetetaan käyttöoikeuksia, joiden perusteella käyttäjälle näytettävä tieto voidaan päätellä. (Sandhu et al. 1996) Ohjelmistoyrityksessä rooleja voisivat olla siis esimerkiksi projektin johtaja, ohjelmistosuunnittelija tai myyjä, jotka jokainen tarvitsevat järjestelmästä saatavaan tietoon omat näkymänsä ja toisaalta esimerkiksi toiminnanohjausjärjestelmien tapauksessa halunnevat tehdä hyvinkin erilaisia asioita järjestelmällä.

Roolit muistuttavat siis hyvin paljon perinteistä käyttäjäryhmiin pohjautuvaa mallia. Erona malleille kuitenkin on se, että roolit yhdistävät joukon käyttäjiä joukkoon oikeuksia, kun taas tyypillisessä tapauksessa käyttäjäryhmät ovat ainoastaan joukko käyttäjiä. Erottamalla nämä kaksi tilannetta keskenään, voidaan myös eriyttää roolien ja käyttöoikeuksien jako eri henkilöille ja näin jakaa ylläpitotoimia eri tasoille. RBAC:lla tuetaan myös helposti tietoturvallisuuden kannalta oleellisia käytäntöjä. Rooleille voidaan jakaa ainoastaan näiden tarvitsemat oikeudet, jolloin väärinkäyttöjä voidaan estää. Lisäksi tehtävien suoritukseen voidaan vaatia useamman roolin yhteistyötä estäen näin tarpeellisten vaiheiden ohituksen prosesseissa ja järjestelmän näyttämä data voidaan abstrahoida korkeammalle tasolle. Monissa järjestelmissä käytetäänkin yleisellä tasolla hyvin paljolti RBAC:n mukaista käyttäjienhallintaa, vaikka pääsynhallinnasta puhuttaisiinkin käyttäjäryhmien hallintana. Näissä tietoturvaan liittyviä puolia ei kuitenkaan usein toteuteta RBAC:n mukaisesta – tosin ei sitä voida RBAC:ia käytettäessäkään vaatia.



Kuva 4: Oraclen roolien- ja pääsynhallinta (Thexton et al. 2011).

Roolipohjaisuuden yhdistäminen portaaliin tarkoittaa siis käytännössä käyttäjienhallinnan toteuttamista RBAC-periaatteiden mukaisesti. Esimerkiksi Oraclen järjestelmissä uuden käyttäjän lisäys portaalijärjestelmään tapahtuu viiden vaiheen kautta, käyttäjäntunnistus kuudentena, kuten kuvassa 4 on esitetty. Aluksi uusi käyttäjä tunnistetaan ja tiedot välitetään roolienhallintaan (eng. Role manager). Roolienhallinnassa käyttäjälle määritellään sopiva rooli esimerkiksi toimenkuvan mukaan, jonka jälkeen oikeudet järjestelmiin asetetaan. Kun käyttäjä sitten kirjautuu portaaliin, voidaan käyttäjä tunnistaa pääsynhallinnassa (eng. Access management) ja jakaa tälle aiemmin määritellyn roolin mukaiset oikeudet (Thexton et al. 2011.)

3.3 Portaalisovelmat

Portaalisovelmat (eng. Portlets) ovat pieniä verkkosovelluksia, joita voidaan sijoittaa portaalisivustoon. Ne käsittelevät sovellukselle tulevia pyyntöjä ja muodostavat näiden perusteella pieniä sirpaleita (eng. fragment), jotka liitetään näytettävään HTML-merkkaukseen. Portaalisovelmien toiminta perustuu Java-teknologiaan ja niiden hallinnoinnista vastaa käytössä oleva portaalisovellus (JSR-000168 2003.)

Kuva 5: iGoogle-palvelun portaalisovellus.

Portaaleissa lähes kaikki käyttäjälle näytettävä HTML-merkkaus voi olla portaalisovelmien muodostamaa. Tällaiset yksittäiset portaalisovelmat muodostetaan Java-palvelinsovelmien (eng. servlet) tapaan ja portaalisovelluksen kautta niitä voidaan kytkeä päälle ja sijoittaa haluttuihin paikkoihin sivustolla (Vainio 2012). Portaalisovelmille on myös tyypillistä kuvan 5 mukainen ikkunamainen rakenne. Yksi sovelma näytetään siis omassa ikkunassaan ja sillä on otsikko- ja kontrollipalkki sekä sirpaleen näyttävä sisältöosa.

4 TOIMINNANOHJAUSJÄRJESTELMÄT

4.1 Taustaa

Toiminnanohjausjärjestelmät ovat olennainen osa lähes jokaisen yrityksen arkea. Toiminnanohjauksella itsellään tarkoitetaan sellaisia tekniikoita ja konsepteja, joilla yrityksen toimintaa voidaan hallita tehokkaasti. Nykyiset ERP-järjestelmät kattavat ideaalitilanteessa yrityksen kaikki liiketoiminnan hallinnointiin liittyvät järjestelmät, keskittäen toiminnan yhteen suureen kokonaisuuteen (Leon 2007.)

ERP-järjestelmiä on kuitenkin edeltänyt useat muut, hieman vastaavia tavoitteita jakavat järjestelmät. 1960-luvun alulla puhuttiin ROP-järjestelmistä (Reorder Point), jotka keskittyivät lähinnä varastonhallintaan, seuraten esimerkiksi materiaalien tarvetta ja käyttöä pyrkien optimoimaan näitä (Jacobs & Weston Jr. 2006, Virtasen 2007 mukaan). ROP-järjestelmät osoittautuivat kuitenkin pian riittämättömiksi vastaamaan toiminnanohjauksen ja tuotannonhallinnan kasvaneisiin vaatimuksiin, joten 1960-luvun lopulla näiden korvaajaksi nousivat MRP-järjestelmät (Material Requirements Planning). Niillä pyrittiin löytämään entistä tehokkaampia keinoja materiaalien ja komponenttien tilaamiselle, selvittämällä muun muassa tuoterakennetta ja valmistuskapasiteettia. Tähän liittyi myös oleellisesti kehittyneempi varastonhallinta, jonka avulla piti pystyä seuraamaan varaston materiaalitilannetta ja tarvittaessa tilaamaan täydennystä. Myöhemmin MRP-järjestelmien suosiosta johtuen niitä laajennettiin vielä niin sanotuiksi suljetun silmukan MRP-järjestelmiksi (eng. Closed-loop MRP), jotka tarjosivat työkaluja ja tekniikoita tuotantoprosessien automatisointiin (Leon 2007.)

Suljetun silmukan MRP-järjestelmiä seurasi 1980-luvulle tultaessa edelleen laajennettu versio toiminnanohjausjärjestelmistä: MRPII (Manufacturing Resource Planning). Erona aikaisempaan myynti- ja operatiivisen tason suunnittelua tuettiin ja päätöksentekoa pyrittiin helpottamaan erilaisilla simulaatiotoiminnoilla (Leon 2007). MRPII-järjestelmien aikana toiminnanohjausjärjestelmistä alkoi hiljalleen myös muovautua nykyisen kaltaisia, koko organisaation kattavia tietojärjestelmien keskuksia, joihin muita järjestelmiä voitiin liittää rajapintojensa kautta (Jacobs & Weston Jr. 2006, Virtasen 2007 mukaan).

1990-luvulla järjestelmien kehityttyä voitiin siirtyä yhä reaaliaikaisempaan tiedon lukemiseen ja käsittelyyn, jolloin alettiin puhua ERP-järjestelmistä. Periaate näissä uusissa järjestelmissä oli pitkälti sama, kuin MRPII-järjestelmissä, mutta käytännössä kaikki toiminnallisuudet vain olivat laajempia, toimintoja oli enemmän ja yrityksen muita jär-

jestelmiä alettiin aidosti integroimaan. ERP-järjestelmien tehtävänä on ennustaa ja tasapainottaa kysynnän ja tarjonnan tilannetta, sekä muodostaa täydellisiä, koko yrityksen kattavia toimitusketjuja. (Leon 2007) Kaikella tällä pyritään varmistamaan, että tarvittavat resurssit ovat aina saatavilla, silloin – ja vain silloin – kun niitä todella tarvitaan. Kuten voidaan havaita, ERP-järjestelmillä on nykyään avainrooli koko yrityksen tehokkaassa toiminnassa ja menestyksessä.

4.2 ERP-järjestelmien kategoriat

Nykyajan ERP-järjestelmät voitaisiin jakaa useammankin kriteerin perusteella, kuten ansaintalogiikan, kohdetoimialan tai laajuuden. Tässä yhteydessä käytetään melko yksinkertaista Ferraiolin esittämää kolmijakoa: toimialakohtaiset (eng. industry specific ERP), pienten- ja keskisuurten yritysten (eng. small business ERP) ja verkkopohjaiset (eng. web-based ERP) ERP-järjestelmät. Näistä toimialakohtaisilla ERP-järjestelmillä tarkoitetaan vahvasti tietyn toimialan tarpeisiin räätälöityjä järjestelmiä. Nämä pohjautuvat usein esikonfiguroituihin pohjiin, jotka johdetaan varsinaisesta niin sanotusti kaiken kattavasta ERP-järjestelmästä ja jotka räätälöidään toimialakohtaisesti (Gable et al. 2000). Pienten- ja keskisuurten yritysten ERP-järjestelmät tarjoavat tarkemmin kohdennettuja ja kustannustehokkaampia – mutta toisaalta ominaisuuksiltaan karsitumpia – järjestelmiä. Nämä järjestelmät pohjautuvat usein geneerisempään ERP-pohjaan, josta asiakkaalle sopiva toteutus räätälöidään tämä kulloisenkin tarpeen mukaan (Gable et al. 2000). Suhteellisen uutena ERP-järjestelmien kategoriana voidaan pitää verkkopohjaisia järjestelmiä, jotka verkkotekniikoiden kehitys ja yhteysnopeuksien kasvu ovat mahdollistaneet. Verkkopohjaiset ERP-järjestelmät kilpailevat ennen kaikkea käyttöönoton helppouden kautta, sillä nämä järjestelmät sijaitsevat fyysisesti esiasennettuna palveluntarjoajalla, jolloin asiakas pääsee verkkoselaimen kautta käsiksi ERP-järjestelmäänsä (Ferraioli). Siinä missä toimialakohtaiset ja pienten- ja keskisuurten yritysten ERP-järjestelmät ovat usein lisenssiperustaisia, myydään verkkopohjaiset ERP-järjestelmissä tavanomaisesti palveluina (SaaS, Software as a Service).

Tämän tutkimuksen aikana toteutettava prototyyppi tullaan yhdistämään Oscar Softwaren Oscar PRO ERP-järjestelmään. Oscar PRO:ta voitaisiin pitää tätä jaottelua vasten tarkasteltuna pienten- ja keskisuurten yritysten ERP-järjestelmänä, sillä vaikka se onkin kustomoitavissa laajasti monen eri alan yritysten tarpeisiin, on se kohdennettu kuitenkin erityisesti teollisuudelle, projektitoimituksiin, teollisuuden tukkukaupalle ja huoltoliiketoimintaan. Oscar PRO toimii myös perinteisesti työpöytäsovelluksena ja se asennetaan usein asiakkaalle, vaikka järjestelmän käyttö myös pilvipalveluna on mahdollista. Verkkopohjaiseksi ERP-järjestelmäksi Oscar PRO:ta ei voida kuitenkaan lukea, sillä verkkokäyttö ei ole pääasiallinen tapa käyttää ohjelmistoa, eikä sitä myöskään myydä palveluna.

Toteutettava huoltoportaalin prototyyppi tuo vahvasti verkkopohjaisten ERP-järjestelmien piirteitä esimerkiksi Oscar PRO:hon ja muihin pääasiallisesti työpöytäsovelluksina toimiviin järjestelmiin. Prototyyppi on tarkoitettu käytettäväksi ainoastaan Internetin kautta, eikä siitä tehdä työpöytäsovellusta ollenkaan. Lisäksi prototyypistä kehitettyjä käyttöliittymiä on mahdollista myydä palveluna, sillä se rakennetaan niin, että portaalia voidaan ylläpitää Oscar Softwaren palvelimella, johon käyttäjä ottaa yhteyden verkkoselaimella. Lisäksi muun muassa roolipohjaisuuden myötä palvelu voidaan hinnoitella esimerkiksi tarvittujen roolien tai käyttäjien määrän mukaan. On kuitenkin muistettava, ettei prototyyppi tule olemaan oma järjestelmänsä, vaan se on pikemminkin liitännäinen olemassa olevaan ERP-järjestelmään, kuten Oscar PRO:hon. Se ei siis muuta millään tapaa käyttämänsä toiminnanohjausjärjestelmän olemusta.

5 WEB-SISÄLLÖNHALLINTA

5.1 Määritelmä

Kun lähdetään kehittämään järjestelmää, jonka tiedetään tulevan käyttöön ytimeltään samanlaisena useille asiakkaille, mutta sisällöltään jokaiselle erilaisena, tarvitaan jonkinlaiset työkalut sisällön hallintaan. Lisäksi kokonaisuuden hallintaan tarvitaan kehysohjelmisto, joka tarjoaa edellä mainitut työkalut. Tähän tarpeeseen voidaan vastata web-sisällönhallinnalla, joka tarjoaa keinot verkkopalvelun sisällön hallinnointiin ja ylläpitoon.

Web-sisällönhallinnalle ominaisina piirteinä voidaan Tolvasen (2007) mukaan pitää pieniä sisältöyksiköitä, älykkäitä sivupohjia, metatietojen hallinnan tärkeyttä, sisällön kustomointia käyttäjän mukaan ja sisällön, rakenteen ja ulkoasun erottamista toisistaan. Tästä listauksesta käy jo hyvin ilmi tavoitteena olevalle järjestelmälle tärkeitä piirteitä ja toisaalta myös kehitystä helpottavia ominaisuuksia. Pienillä sisältöyksiköillä tarkoitetaan sitä, miten käyttäjälle lopulta näkyvä verkkosivu voidaan koostaa erillisistä pienistä palasista, joita voidaan tehokkaasti hallita yksittäin. Tällöin esimerkiksi sivuston yleiset rakenteet, kuten header- tai footer-osat voidaan eriyttää omiin tiedostoihinsa ja näin helpottaa ylläpitoa. Älykkäät sivupohjat puolestaan voivat sisältää esimerkiksi muuttujia, jotka puolestaan mahdollistavat erilaiset ohjelmallisesti luodut sivut. Metatiedot ovat tietoa tiedosta, joilla esimerkiksi voidaan kertoa, minkä tyyppistä sisältöä jokin yksittäinen sivu sisältää. Sisällön kustomointia varten on kerättävä jonkinlaista tietoa käyttäjästä, jotta yksittäinen käyttäjä voidaan tunnistaa ja näin tarjota oikeanlainen sisältö kullekin käyttäjälle. Tähän liittyy vahvasti vaatimus roolitiedon tallennukseen, koska yksi toteutettavan järjestelmän keskeisistä vaatimuksista on roolipohjaisuus. Sisällön, rakenteen ja ulkoasun erottaminen toisistaan toteutuu osittain jo sivupohjia hyödyntämällä, kun suoritettava ohjelmakoodi voi päätellä rakenteen, avata ja täyttää oikeat sivupohjat sisällöllä, johon esimerkiksi CSS-tiedostoilla kustomoidaan ulkoasu.

5.2 Sisällönhallintajärjestelmät

Sisällönhallintajärjestelmä (CMS, Content management system) on laaja ja monia erityyppisiä sovelluksia kattava käsite. Määritelmän mukaan se on tietojärjestelmä, joka mahdollistaa sisällön julkaisun, muokkauksen ja ylläpidon yhdestä keskitetystä käyttöliittymästä. Tyypillisesti sisällönhallintajärjestelmät ovat dokumenttienhallintajärjestel-

mistä kehittyneempiä versioita ja ne tarjoavat muun muassa julkaisu- ja aineistonhallintajärjestelmien piirteitä, esimerkiksi tiedostopankkien muodossa. Erityisen tärkeä vaatimus sisällönhallintajärjestelmälle on sisällön hallittavuus sen koko elinkaaren aikana. Tyypillisesti verkkosivut elävät jonkin verran luontinsa jälkeen, jolloin sivuja pitää pystyä joustavasti päivittämään. Tavalliseen painettuun mediaan verrattuna sisällönhallintajärjestelmien on myös huolehdittava sisällön poistamisesta järjestelmästä, jotta käyttäjien nähtäväksi ei jää vanhentunutta sisältöä tai toimimattomia sivuja (Tolvanen 2007.)

Sisällönhallintajärjestelmät saatetaan nähdä myös hieman yksinkertaistettuina toiminnanohjausjärjestelminä, sillä niihin voi liittyä tuotantoprosessien hallintaa (Tolvanen 2007). Toiminnanohjausjärjestelmiä ei kuitenkaan pitäisi samaistaa sisällönhallintajärjestelmiin, sillä lähtökohdat molemmille järjestelmille ovat tyystin erilaiset. Siinä missä sisällönhallintajärjestelmät ovat nimensä mukaisesti sisältöpainotteisia, toiminnanohjausjärjestelmät keskittyvät ennemmin ennen kaikkea prosessin toiseen päähän, eli varsinaiseen tehtävien kulkuun ja tuotteiden hallintaan.

5.2.1 Sisällönhallintajärjestelmän valinta

Valittaessa sisällönhallintajärjestelmää, on erilaisia vaihtoehtoja käytännössä kolme: valmis ohjelmisto, räätälöity järjestelmä tai itse kehitetty järjestelmä (Tolvanen 2007). Valmiilla ohjelmistolla tarkoitetaan niin sanotusti suoraan paketista käyttövalmista sovellusta, jolloin muun muassa ohjelmointiosaaminen ei olisi välttämätöntä järjestelmän käytön kannalta. Räätälöidyllä järjestelmällä tarkoitetaan sovellusta, joka kustomoitaisiin vastaamaan asiakkaan tarvetta, esimerkiksi kehittämällä rajapinta toivotun mukaiseksi. Itse kehitettäessä järjestelmä rakennettaisiin alusta pitäen itse vastaamaan täsmälleen asetettuihin vaatimuksiin. Käytännössä ainoat realistiset vaihtoehdot prototyypin kannalta olivat siis räätälöity järjestelmä tai itse kehitetty järjestelmä, koska valmista sopivaa järjestelmää ei markkinoilta suoraan löytynyt.

Keskeisenä perusteena valinnassa Oscar Softwaren puolelta oli vaatimus joustavaan ja helposti kustomoitavaan järjestelmään, jotta jokaisen asiakkaan erityisvaatimukset saadaan kohtuullisella vaivalla tyydytettyä. Tämän kannalta oleellista olisi, että sisällönhallintajärjestelmän ydin voitaisiin eriyttää kustomoidusta sisällöstä ja rajapinnasta toiminnanohjausjärjestelmään, eli toisin sanoen tarvitaan portaalijärjestelmä. Oscar Softwarella on itse kehitetty sisällönhallintajärjestelmä Oscar CMS, jota ei kuitenkaan ollut suunniteltu portaalityyppiseen toimintaan. Järjestelmää oltaisiin kuitenkin projektin puitteissa voitu laajentaa tähän tarpeeseen. Räätälöitävistä järjestelmistä kandidaattina oli Liferay portaalijärjestelmä, josta Oscar Softwarella oli jo kuultu lupaavia käyttäjäkokemuksia. Prototyyppiä varten päädyttiin lopulta valitsemaan Liferay, koska kehityksessä haluttiin saada nopeasti näkyviä tuloksia aikaan, jolloin uuden järjestelmän käyttöönotto ja räätälöinti nähtiin oman vanhan järjestelmän jatkokehitystä nopeampana ratkaisuna.

5.2.2 Liferay

Liferay[7] on Javalla toteutettu avoimen lähdekoodin portaalijärjestelmä. Se toteuttaa web-sisällönhallinnalle ja portaalisovelluksille tyypilliset piirteet ja useita muita tärkeitä ominaisuuksia, kuten käyttäjätunnuksien ja -oikeuksien hallinnan ja kieliversioinnin. Koska Liferay on portaalijärjestelmä, perustuu sen käyttö vahvasti portaalisovelmien hyödyntämiseen. Jokaisen käyttäjälle näytettävän sivun sisältö voidaan siis koostaa useista portaalisovelmista, jotka puolestaan voidaan kustomoida tarpeen mukaan. Näitä portaalisovelmia on myös tarjolla valmiina lukuisia Liferay'n omasta palvelusta Liferay Marketplacesta[8], joka sisältää sekä ilmaisia että maksullisia portaalisovelmia. Erityisen hyödyllistä prototyypin kannalta on suora valmis tuki roolipohjaiselle sisällön esittämiselle. Liferay'n avulla jokainen portaalisovelma voi siis näyttää kirjautuneelle käyttäjälle tämän roolin mukaiset toiminnot, jolloin jokaiselle roolille ei tarvitse luoda omaa sovelmaa, vaan sovelmien logiikkaa voidaan kustomoida roolikohtaisesti ja näin säästää ylläpito- ja ohjelmointityössä. Liferay tarjoaa myös ulkoasun kustomointiin työkaluja, jolloin sivusto voidaan muokata helposti asiakkaan näköiseksi.

Pelkkä hyvä soveltuvuus prototyypin vaatimuksiin ei kuitenkaan ole ainoa syy uuden järjestelmän käyttöönottoon. Aloittamalla niin sanotusti puhtaalta pöydältä voidaan luopua menneisyyden huonoksi todetuista ratkaisuista, joita on kuitenkin ollut pakko käyttää, koska näiden muuttaminen olisi aiheuttanut kohtuutonta vaivaa. Lisäksi samalla voidaan ottaa huoletta käyttöön uusimpia tekniikoita, kun niiden yhteensopivuutta vanhojen versioiden kanssa ei tarvitse ottaa huomioon. Oscar Softwarella on myös jo vahvaa osaamista Java-ohjelmoinnista, jolloin Liferay'n ohjelmakoodia on helpompi lähteä muokkaamaan. Avoimen lähdekoodin sovellus tuo lisähyötynä myös periaatteessa ilmaisen osittaisen testauksen ja kehityksen, kun Liferay'n ydintä kehitetään Oscar Softwaren ulkopuolella jatkuvasti eteenpäin.

Liferay on myös pärjännyt hyvin aikaisemmissa eri portaalijärjestelmien vertailuissa. Akram et al. (2005) toteutti yhden tällaisen vertailun eri portaalijärjestelmien välillä ja Liferay saavuttikin siinä kärkisijan. Osa-alueet, joista Liferay sai täydet pisteet olivat JSR 168 -standardin noudattaminen, asennuksen helppous, kustomointi, hyödyllisten ilmaisten portaalisovelmien määrä ja palvelinriippuvaisuuden vähyys. Näistä erityisesti asennuksen helppous ja kustomointi olivat tärkeitä kriteereitä, jotka huoltoportaalille asetettujen vaatimusten mukaan valitun portaalijärjestelmän tulisi toteuttaa. Millään osa-alueella Liferay ei myöskään saanut kovin matalia pisteitä ollen vähintään keskiverto tasolla huonoimmillaankin. Vertailun tulokset vapaasti suomennettuna ja Liferay'n tulokset korostettuna on esitetty taulukossa 2, jossa nolla on huonoin ja viisi paras.

7 Liferay'n viralliset kotisivut [WWW] [Luettu 15.2.2013] http://www.liferay.com/
8 Liferay Marketplace [WWW] [Luettu: 1.5.2013]: http://www.liferay.com/marketplace

Taulukko 2: Portaalijärjestelmien vertailun tulokset (Akram et al. 2005).

	Sakai 1.5	uPortal	Gridsphere	Exo	**Liferay**	Stringbeans
JSR168 -standardin noudattaminen	0	5	5	5	**5**	5
Asennuksen helppous	3	5	5	5	**5**	5
Käytön helppous	3	5	4	5	**4**	5
Dokumentaatio	2	2	4	3	**3**	5
Tukipalvelut	3	3	4	4	**3**	5
Portaalin ylläpito	3	5	4	5	**4**	5
Kustomointi	4	3	4	3	**5**	4
Ilmaiset, hyödylliset portaalisovelmat	4	3	4	3	**5**	3
Suorituskyky	2	4	3	4	**3**	3
Tietoturva	3	4	3	4	**4**	4
Teknologian käyttö	3	3	4	5	**4**	3
Portaalin ominaisuudet	2	2	3	5	**4**	2
Palvelinriippu-vaisuus	3	3	3	4	**5**	3
WSRP:n noudattaminen	0	3	0	3	**3**	0
Yhteensä	35	49	51	57	**58**	51

Vaikka Liferay'n käyttöönotolla saadaan paljon hyvää aikaan, ei uusi järjestelmä ole kuitenkaan ilman huonoja puolia. Ensinnäkin Oscar Softwarella ei ole ennestään kokemusta Liferay'n käytöstä, joten käyttöönotto ja opettelu vie heti jonkin verran aikaa. Tähän voidaan vaikuttaa esimerkiksi tekemällä kumppanisopimus jonkun jo Liferay'ta pitempään käyttäneen yrityksen kanssa, jolloin kehitysapua ja koulutusta voidaan saada prototyypin toteutuksen alkua helpottamaan. Toiseen järjestelmään vaihtaminen voi myöskin osoittautua ongelmaksi, jos Liferay'hin ollaan sitouduttu, mutta siihen ei ollakaan enää syystä tai toisesta tyytyväisiä. Kehitysvaiheessa täytyikin pyrkiä löytämään ratkaisuja, jotka mahdollistivat Liferay'n ytimen eriytyksen itsetuotetusta koodista, jotta vastaavilta ongelmilta vältyttäisiin.

5.2.3 Oscar CMS

Oscar CMS on PHP:llä toteutettu Oscar Softwaren kehittämä sisällönhallintajärjestelmä, joka mahdollistaa perinteisten verkkosivujen luonnin lisäksi esimerkiksi verkkokaupan toteutuksen lisäosien avulla. Tämän lisäksi Oscar CMS voidaan kytkeä Oscar Softwaren toiminnanohjausjärjestelmiin käyttäen XML-sanomaliikennettä. Kuten Liferay'ssa. Myös Oscar CMS:ssä on web-sisällönhallinnalle tyypilliset piirteet toteutettuna, mutta portaalimahdollisuutta tai käyttäjien roolitukseen perustuvaa sisällön esitystä ei ole valmiina toteutuksessa.

Oscar CMS:n suurin vahvuus on järjestelmän tuttuus. Toteutukseen voitaisiin siis käydä heti ilman uuden järjestelmän tuomaa opetteluvaihetta. Toisaalta muutoksia perusrakenteeseen jouduttaisiin tehdä, jotta portaalitoiminnallisuus saataisiin aikaan, mikä osaltaan kumoaisi saavutetun edun Liferay'hin nähden. Portaalikehityksen myötä Oscar CMS:ää tulisi kuitenkin kehitettyä samalla eteenpäin, jolloin voitaisiin parannella myös muita Oscar Softwaren järjestelmiä.

Prototyypin kehitykseen Oscar CMS:ää ei kuitenkaan haluttu valita, koska hyvä tilaisuus uusien tekniikoiden opetteluun haluttiin Oscar Softwaren puolelta hyödyntää. Lisäksi rajapintatoteutus, joka nojaa XML-sanomiin, on todettu käytännössä virhealttiiksi ja virhetilanteissa testaus ja ongelman jäljitys onkin usein hankalaa. Oscar CMS vaatii myös oman tietokannan, jonne kahdennetaan toiminnanohjausjärjestelmästä esimerkiksi tuotetiedot. Vaikka tällä saavutetaan se, että yhteyden toiminnanohjausjärjestelmiin ei tarvitse jatkuvasti olla käytössä, tuo tiedon kahdennus kuitenkin omat haasteensa muun muassa ylläpidon ja samanaikaisuuden hallinnan kannalta.

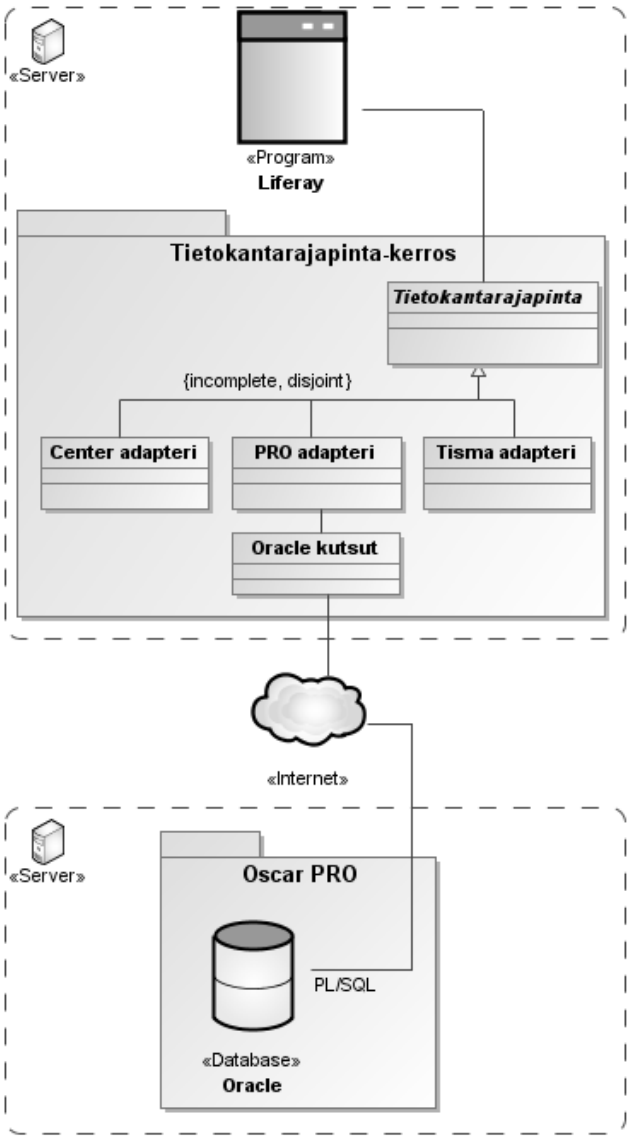
6 KÄYTTÖLIITTYMÄN PROTOTYYPPI

6.1 Esittely

Oscar Softwarella on kehitteillä huoltoportaali (Oscar Service Portal, OSP), jonka tehtävänä on tarjota huolto- ja kunnossapitotoimialalle kohdennettu ratkaisu asiakaspalvelun organisointiin. Tutkimusta varten toteutetaan huoltoportaalin prototyyppiversio, jonka avulla pyritään käytännössä selvittämään, kuinka roolipohjainen, web-portaalikäyttöliittymä voidaan toteuttaa toiminnanohjausjärjestelmään – tässä tapauksessa Oscar Softwaren Oscar PRO -järjestelmään. Prototyyppi koostuu käytännössä kolmesta osasta: huoltoportaalipuolta hallitsevasta sisällönhallintajärjestelmästä (Content Management System, CMS), Oscar PRO:n kanssa keskustelevasta toiminnallisuudesta sekä rajapinnasta PRO:n ja prototyypin välillä.

Sisällönhallintajärjestelmäksi tarjolla oli kaksi varteenotettavaa vaihtoehtoa: Oscar CMS ja Liferay. Näistä päädyttiin käyttämään Liferay'ta, joka on avoimen lähdekoodin projekti ja vaikutti ominaisuuksiltaan erinomaiselta prototyypin kehitystä varten, sillä se sisälsi valmiina sisällönhallintapuolen, joka tuki muun muassa roolipohjaisuutta ja portaalisovelmien käyttöä. Tarkemmin Liferay'ta ja muita projektia varten harkittuja sisällönhallintajärjestelmiä on tarkasteltu luvussa 5.2.

Oscar PRO käyttää tietokantanaan Oraclen kantaa, joten siihen ollaan yhteydessä PL/SQL:llä (Procedual Language/Structured Query Language), joka on pääasiallinen väline tehdä monimutkaisempia tietokantakutsuja Oracle kantoihin. Suuri osa Oscar PRO:n toimintalogiikkaa rakentuu PL/SQL-kutsujen varaan, joten sen avulla tietokantaa voidaan käyttää suoraan ja jo olemassa olevia kutsuja voidaan osin suoraan hyödyntää prototyypin kehityksessä. On kuitenkin huomioitava, että toiminnanohjausjärjestelmän ja siihen liitettävän huoltoportaalin välinen rajapinnan toteuttava luokka on tehtävä aina uudelleen mikäli tietokanta tai toiminnanohjausjärjestelmä vaihdetaan. Toteutuksesta on tältä osin täysin mahdotonta tehdä täysin geneeristä, sillä eri järjestelmien rakenteet, toimintalogiikka ja tietomallit eroavat huomattavan paljon toisistaan.



Kuva 6: Huoltoportaalin korkean tason arkkitehtuurikuvaus ja tarkemmin tietokantarajapinta.

Juuri edellä mainituista syistä huoltoportaalin rajapinnan suunnittelu onkin erityisen tärkeässä osassa. Kuvasta 6 huomataan, kuinka huoltoportaalin ja toiminnanohjausjärjestelmän välinen rajapinta on kapseloitu ja piilotettu funktiokutsujen taakse. Liferay'n käyttämälle Java-koodille rajapinta näyttää täysin samalta, riippumatta mikä tietokanta tai toiminnanohjausjärjestelmä taustalla on. Tällöin uutta järjestelmää liitettäessä, kehittäjien vastuulle jää ainoastaan rajapinnan toteuttavan adapterin luonti, niin että uuden järjestelmän tietorakenne huomioidaan ja käytetään oikeita tietokantakutsuja. Mikäli portaaliin puolestaan halutaan uusia ominaisuuksia, tulee näistä tehdä rajapintaan uusi määrittely ja sille vastaava toteutus jokaiseen adapteriin. PL/SQL-toiminnallisuus puolestaan rakentuu Oracle-tietokannan yhteyteen.

6.2 Kehityksen tavoitteet

Huoltoportaalin kehityksen keskeisimpänä tavoitteena voidaan pitää toiseen tämän tutkimuksen tutkimuskysymykseen vastaamista – eli soveltuuko Liferay alustana roolipohjaisen, web-portaalikäyttöliittymän toteutukseen. Tähän varsin laajaan kysymykseen pyritään löytämään vastausta useamman pienemmän tavoitteen kautta.

Liiketoiminnalliselta kannalta huoltoportaalin päätavoite asiakasyrityksiä ajatellen on, että sen kautta tehtyjä huoltotöitä voitaisiin laskuttaa toteuman perusteella, eikä esimerkiksi arvion. Huoltoportaalin kautta huoltohenkilöt voivat kirjata työvaiheisiin käyttämänsä ajan ja materiaalit suoraan taustalla toimivaan Oscar PRO järjestelmään heti työvaiheen suoritettuaan. Toiminta aikaisemmin on hyvin vaihtelevaa riippuen asiakasyrityksestä, mutta kirjauksia saatetaan tehdä esimerkiksi aina työpäivän päätteeksi, kun huoltohenkilö pääsee taas työpäätteen äärelle. Tästä seuraa lisäksi samalla se, että huoltotöistä saadaan koottua massadataa, joka kuvastaa suoraan oikeaa työmäärää. Tällöin huoltopäällikkö näkisi heti jokaisen huoltohenkilön työtilanteen ja voisi entistä paremmin suunnitella ja kohdentaa yrityksen toimintaa. Loppuasiakkaankin käyttöä huoltoportaali voisi helpottaa, sillä sen avulla huoltopyyntöjä voidaan kirjata suoraan järjestelmään, mikä nopeuttaa prosessia, kun järjestelmä osaa automaattisesti tuottaa oikeita tietorivejä ERP-järjestelmään.

Pelkkä valitun toimialan tarpeiden tyydyttäminen ja tavoitteiden saavuttaminen liikekannallisilta osin ei kuitenkaan johda vielä päätavoitteen täyttymiseen. Jo järjestelmää ideoitaessa Oscar Softwarella haluttiin varmistaa, että luotu web-portaalijärjestelmä mahdollistaisi mahdollisimman helpon kustomoinnin sekä asiakasyritysten yksilöllisiin tarpeisiin, että Oscar Softwaren muiden toiminnanohjausjärjestelmien liittämisen mahdollistamiseksi. Kustomointi ja joustavuus näiltä osin voidaankin nähdä myös liiketoiminnallisesti erittäin tärkeänä näkökohtana, sillä monien kyselyiden mukaan nämä ominaisuudet ovat asiakastyytyväisyyden kannalta ensiarvoisen tärkeässä osassa (What makes an ERP user satisfied 2012, What really matters for ERP users 2010). Lisäksi Oscarin toimintafilosofiaan kuuluu järjestelmien muokkaus asiakkaiden toiveiden mukaan, niin että asiakkaalle voidaan toimittaa juuri heidän yksilölliseen tarpeeseen parhaiten vastaava kokonaisuus. Tämä luo haasteita etenkin portaalin ja Oscarin toiminnanohjausjärjestelmien väliselle rajapinnalle, jotta esimerkiksi tietorakenteeseen tehtävät asiakaskohtaiset muutokset eivät aiheuttaisi kohtuutonta muutostarvetta portaalin puolelle.

Huoltoportaalin kehityksen aikana sekä uusien toimintojen luonnin että visuaalisen ilmeen kustomoinnin helppouteen tullaan ottamaan tältä osin kantaa, sillä nämä ovat myös liiketoiminnalliselta näkökulmalta kriittisiä asioita. Jotta huoltoportaali ja muut jatkossa kehitettävät portaalisovellukset olisivat mahdollisia, on järjestelmän oltava helposti tuotteistettava. Tämä tarkoittaa, että portaalijärjestelmässä uusien toimintojen luonti ei saisi vaatia suhteettoman paljon työtä saavutettavaan hyötyyn nähden ja että ylläpito ja asennus voitaisiin suorittaa helposti. Ylläpito on pitkällä tähtäimellä erityisesti tärkeä aspekti, sillä ohjelmistokehityksessä painopisteen tulisi pysyä järjestelmän kehi-

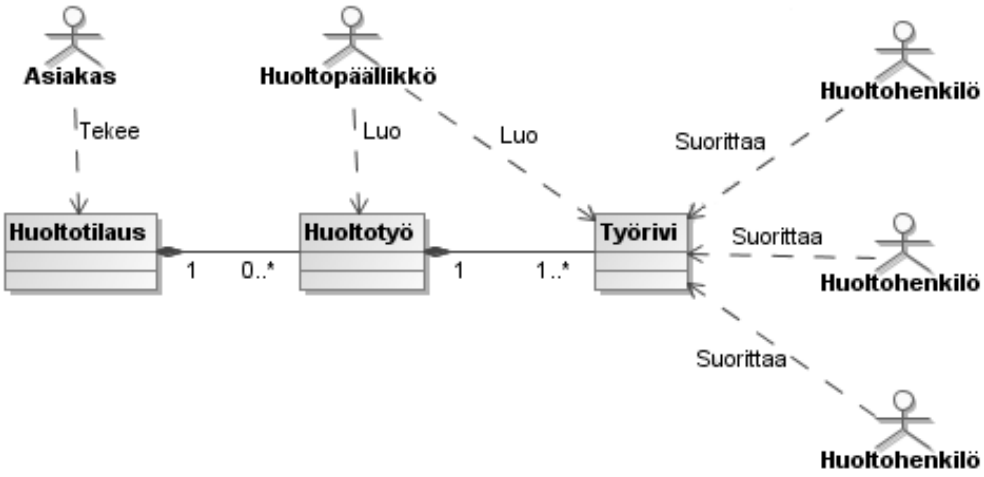
tyksen puolella, eikä ylläpitotoiminnoissa. Ylläpidon kannalta järjestelmä tulisi siis saada helposti konfiguroitavaksi ja toisaalta myös taustalla toimivan Liferay-portaalijärjestelmän päivitys täytyisi tarvittaessa onnistui niin, ettei portaalisovelmiin tarvitsisi tämän takia tehdä muutoksia.

Viimeisenä tavoitteena huoltoportaalin tulisi pystyä näyttämään, sopeutuuko valittu portaalijärjestelmä hyvin eri päätelaitteilla käytettäväksi. Koska nykyisin jo yli kolmannes työvoimasta käyttää työssään mobiililaitteita (Mobile Worker Population 2012), hyvä tuki näille halutaan varmistaa jo prototyyppivaiheessa. Mobiililaitteina tässä yhteydessä voidaan pitää sekä matkapuhelimia, että taulutietokoneita (eng. tablet). Koska laitteiden kirjo mobiilipuolella on valtava, on prototyypin tavoitteena tukea sujuvasti matkapuhelimista älypuhelimia, joissa on käyttöjärjestelmänä joko kosketusnäyttöä hyödyntävä Android tai iOS. Taulutietokoneiden osalta prototyypille riittää, että se toimii hyvin ainakin Applen iPad-laitteilla näiden versiosta riippumatta. Tulevaisuudessa tärkeää olisi varmistaa prototyypin toiminta myös Windows Phone 8 -käyttöjärjestelmällä, mutta prototyypin puitteissa se ei kuulu välittömiin tavoitteisiin.

6.3 Keskeisimmät roolit ja toiminnot

Huoltoportaalin keskeisimmät toiminnot jakaantuvat neljälle eri roolille: huoltopäällikkö, huoltohenkilö, myyjä ja asiakas. Huoltopäällikkö johtaa huoltotoimintaa ja hänen vastuunaan on muun muassa käsitellä saatuja huoltotilauksia, luoda uusia huoltotöitä ja seurata huollon kokonaistilannetta esimerkiksi erilaisten raporttien avulla. Huoltohenkilön tehtävänä on suorittaa varsinainen huolto yrityksen tiloissa tai asiakkaalla. Myyjät puolestaan tyypillisesti kiertävät asiakkailla ja tarvitsevat ajankohtaista tietoa esimerkiksi huoltodatasta ja huoltosopimuksista. Asiakkaat ovat Oscar Softwaren asiakasyrityksen asiakkaita, jotka voivat olla joko yrityksiä tai yksityishenkilöitä, hieman toimialasta riippuen.

Koska prototyypin kehityksen aikataulu on kuitenkin varsin tiukka, toteutetaan siihen ainoastaan huoltohenkilö- ja asiakasrooleihin liittyvät toiminnot ja näistäkin vain kaikkein tärkeimmät. Lisäksi näihin toimintoihin tehdään huoltopäällikkö-roolille muutama toiminto eri roolien vaikutusten demonstroimiseksi. Nämä kaksi roolia valittiin, koska huoltopäällikön pääasiallinen työväline on useimmiten nykyisellään yrityksen toiminnanohjausjärjestelmä ja myyjä puolestaan voi käyttää Oscar Softwaren muita myyjäliittymiä. Huoltohenkilöiden ja asiakkaiden saaminen Oscar Softwaren tuotteiden käyttäjien piiriin huoltotoiminnassakin on kuitenkin liiketoiminnallisesti tärkeää, joten nämä roolit valikoituivat mukaan prototyyppiin.



Kuva 7: Huoltotilauksen muodostuminen ja roolien tehtävät.

Huoltohenkilölle keskeisimmäksi toiminnoksi prototyyppiin valittiin huoltotyön kirjaaminen ja asiakkaalle huoltopyynnön tekeminen (Oscar Service Portal työmääräarviot 2013). Huoltopyyntö on ylätason käsite, joka sisältää yhden työkokonaisuuden. Käytännössä tällaisesta kokonaisuudesta huoltopäällikkö luo yhden tai useamman huoltotyön, joka puolestaan sisältää yhden tai useamman työrivin. Työrivit voidaan jakaa eri huoltohenkilöille, jotka kaikki siis näkevät saman huoltotyön, mutta vain heihin itseensä liitetyt työrivit ovat kirjattavissa. Rakennetta on havainnollistettu kuvassa 7.

6.3.1 Huoltotyön kirjaaminen huoltohenkilönä

Huoltotyö on usein useita työvaiheita sisältävä työkokonaisuus, jonka vaiheet on jaettu yhdelle tai useammalle huoltohenkilölle. Siihen sisältyy muun muassa tietoa työhön liittyvästä asiakkaasta ja laitteesta, työn vaiheita, tarvittuja materiaaleja, ohjeistusta, joita huoltohenkilö voi omalta osaltaan kirjata.

Huoltoportaalin prototyypillä huoltohenkilön täytyy voida ensinnäkin tarkastella listaa huoltotöistä, joissa on hänelle osoitettu työrivi, ja valita näistä jokin lähempään tarkasteluun. Huoltotyön täytyy sisältää tarvittava informaatio työstä, kuten asiakkaan ja laitteen tiedot, sekä mahdolliset ohjeet huoltopäälliköltä. Lisäksi työhön täytyy voida kirjata siihen merkatut työrivit, joita voivat olla esimerkiksi työmatka, laitteen kunnon tarkastus tai muu työn yksittäinen vaihe. Työlle saattaa olla tiedossa myös suunnitellut materiaalit, joita työn tekemiseen tarvitaan varastosta. Näiden osalta huoltohenkilön on voitava kirjata materiaalikäytön toteuma (Oscar Service Portal työmääräarviot 2013.)

6.3.2 Huoltopyynnön tekeminen asiakkaana

Huoltopyyntö on huoltotyön esivaihe. Tavallisesti huoltopyynnön kirjaa yrityksessä nykyisellään esimerkiksi asiakaspalveluhenkilö, mutta prototyypin avulla loppuasiakkaan olisi voitavat tehdä sama kirjaus yksinkertaisessa muodossa. Työlle täytetään tarvittavat tiedot, kuten asiakkaan ja laitteen tiedot sekä kuvaus viasta. Huoltopyynnön perusteella yhdestä pyynnöstä luodaan huoltopäällikön toimesta useampia huoltotöitä ja näihin työrivejä, jotka sitten jaetaan huoltohenkilöille.

Huoltoportaalin prototyypin kautta asiakkaan tulee voida valita haluamansa laite ja kirjoittaa sanallinen kuvaus laitteessa ilmenneestä viasta. Lisäksi pyyntöön on voitava lisätä tiedostoja, kuten kuvia hajonneesta kohdasta tai vaikkapa laitteelta tulostettu virheraportti. Huoltoportaalin kautta tehtävän huoltopyynnön ei tarvitse olla täydellinen, eli riittää kunhan se sisältää vähintään edellä mainitut tiedot, joiden jälkeen pyyntöön voidaan täydentää tarvittava informaatio sekä toiminnanohjausjärjestelmän automatiikalla, että myöhemmin huoltopäällikön toimesta (Oscar Service Portal työmääräarviot 2013.)

6.3.3 Huoltopäällikön toiminnot prototyyppivaiheessa

Jotta eri roolien toimintaa voidaan demonstroida portaalissa, toteutetaan huoltopäällikölle toimintoja huoltotyön kirjaamiseen ja huoltopyynnön tekemiseen. Huoltopäällikkö voi siis suorittaa vastaavat toiminnot, kuin asiakas tai huoltohenkilö prototyypin kautta, mutta kummassakin on vain huoltopäällikölle näkyviä kenttiä. Lisäksi huoltopäällikkö saa esimerkiksi asiakas- ja työlistauksiin näkyviin kaikki järjestelmästä löytyvät tiedot.

Huoltotyötä kirjattaessa huoltopäällikkö näkee listauksestaan ensinnäkin kaikki järjestelmässä olevat huoltotyöt. Työn tarkastelussa näkymä on muuten vastaava, kuin huoltohenkilöllä, mutta huoltopäällikkö voi lisäksi täydentää työohjetta, poistaa työrivejä tai osoittaa työrivejä haluamalleen huoltohenkilölle.

7 PROTOTYYPIN KUVAUS JA ARVIOINTI

7.1 Prototyypin arkkitehtuuri

Arkkitehtuuriltaan prototyyppi voidaan jakaa neljään kokonaisuuteen:

- Liferay ja sen portaalisovelmille tarjoama rajapinta.
- Portaalisovelmat ja niiden käyttämät näkymät.
- Rajapinta toiminnanohjausjärjestelmiin.
- Toiminnanohjausjärjestelmä.

Kokonaisuuksien jakaantuminen eri palvelinten välille on kuvattu luvussa 6.1 kuvassa 6. Arkkitehtuurin keskeisenä tavoitteena on ollut eriyttää Liferay'n ydin mahdollisimman tehokkaasti muusta järjestelmästä ja samalla myös mahdollistaa erilaisiin tietomalleihin perustuvien toiminnanohjausjärjestelmien liittämisen osaksi portaalia. Liferay'n arkkitehtuuriin ei tässä tarkemmin paneuduta, mutta todettakoon sen tarjoavan palvelurajapinnan, jota portaalisovelmat voivat kutsua tai jota voidaan käyttää Web service -kutsuilla esimerkiksi JSON (JavaScript Object Notation) tai SOAP (Simple Object Access Protocol) -muodossa (Ferrer 2012). Käytännössä Liferay'n rajapinnan käyttö on helpointa hyödyntämällä sen omaa ohjelmointiympäristöä (eng. integrated development environment, IDE) ja sitten periyttämällä luotavat portaalisovelmat Liferay'n omasta GenericPortlet-luokasta (luku 7.2.3).

Portaalisovelmien kerros on jaettu kahteen osaan: näkymä ja logiikka. Näkymän tehtävänä on näyttää portaalisovelman sisältö portaalisivustolla ja se kutsuu prototyypin tapauksessa JavaScriptillä JSON-muodossa toimintalogiikkapuoltaan. Toimintalogiikan toteuttava Java-luokka on siis periytetty vähintään GenericPortlet-luokasta, mutta prototyypin tapauksessa varsinaisen portaalisovelman ja Liferay'n väliin on luotu vielä OSPBasePortlet-luokka, josta jokainen prototyypin portaalisovelma on periytetty. Jos Liferay vaihdettaisiin johonkin muuhun taustajärjestelmään, voidaan tällä ratkaisulla helpottaa konversiota hieman, kun jokaiseen portaalisovelmaan ei tarvitsisi tällöin tehdä samoja muokkauksia. Lisäksi jokainen toiminnanohjausjärjestelmää käyttävä portaalisovelma tarvitsee AdapterFactory-luokkaa, joka tarjoaa oikean rajapintaluokan toiminnanohjausjärjestelmästä tietojen hakemiseen. Kaikki prototyypin toteuttamat toiminnot toiminnanohjausjärjestelmästä luotiin portaalisovelmina.

Toiminnanohjausjärjestelmään yhteydestä huolehtiva rajapintakerros koostuu kolmesta osasta: rajapintaluokka, rajapinnan toteuttava luokka ja kutsujen apuluokka. Rajapintaluokka on oleellinen, jotta rajapintakerros voidaan abstrahoida portaalisovelmakerrokselta, jolloin rajapinta näyttää portaalisovelmille samalta riippumatta toiminnanohjausjärjestelmästä. Apuluokat ja niiden tarve riippuvat täysin rajapinnan toteuttavasta luokasta ja siitä, millaista tiedonsiirtotapaa luokka käyttää toiminnanohjausjärjestelmän suuntaan. Rajapintakerros on kuvattu vielä tarkemmin luvussa 7.2.4.

Toiminnanohjausjärjestelmän rajapinta riippuu luonnollisesti täysin käytetystä järjestelmästä, mutta Oscar PRO:n tapauksessa prototyypin puoleinen rajapinta voi kutsua Oscar PRO:n PL/SQL-proseduureja. Oleellista on, että Oscar PRO:n toimintalogiikkaa ei jouduttaisi kirjoittamaan uudelleen prototyypin koodiin, vaan tarkoituksena on hyödyntää mahdollisimman vahvasti olemassa olevia PL/SQL-kutsuja. Toisaalta prototyyppi tarjoaa myös Oscar PRO:n kehitykselle hyvän tilaisuuden kehittää rakennettaan ja siirtää toimintalogiikkaa yhä enemmän PL/SQL-kutsujen puolelle. Parhaassa tapauksessa jatkossa prototyyppi voisi aina kutsua valmiita PL/SQL-kutsuja, eikä uusia tarvitsisi kirjoittaa sitä varten.

7.2 Prototyypin käyttö

7.2.1 Asennus

Prototyyppiä varten tarvitaan Liferay ja tietokanta Liferay'n käyttäjätietojen tallennukseen. Palvelinkoneena voi toimia Linux-, Mac- tai Windows-kone, kunhan käytössä on vähintään 1GB RAM-muistia ja Java-ympäristö. Tässä tutkimuksessa tehty prototyyppi käyttää Liferay'n versiota 6.1 ja sitä ajetaan Apache Tomcatilla ja Windows-koneella. Liferay'sta hankittiin siis Liferay/Tomcat-paketti, joka sisältää kaiken tarvittavan Liferay'n pystytystä varten. Liferay'n asennus sujuu helposti, kunhan Java-ympäristö on käytössä, suorittamalla paketin mukana tulleen *startup.bat*-tiedoston, joka käynnistää ja suorittaa Liferay'n asennusrutiinit automaattisesti. Tämän jälkeen Liferay on käytössä lokaalisti selaimen kautta. Mikäli Liferay asennettaisiin palvelimelle, täytyy vielä ennen käynnistystä lisätä valitun tietokannan JDBC-ajuri Tomcatin hakemistoon ja poistaa esimerkkisivun sisältö paketista siistimmän lähtötilanteet luomiseksi.

Kun tiedostot on saatu paikoilleen ja Liferay käyntiin, avautuu ensimmäisellä käyttökerralla Liferay'n kolmiosainen asennussivu. Asennustyökalulla nimetään portaali ja asetetaan sen oletuskieli, luodaan ylläpitäjätunnus ja tarvittaessa määritellään tietokannaksi jokin oletuksesta poikkeava kanta. Jos tietokantasovellus halutaan vaihtaa, tietokanta tulee olla luotuna ennen asennuksen suorittamista loppuun. Liferay vaatii tietokannaltaan UTF-8 yhteensopivuutta, jotta kaikki tarvittavat erikoismerkit voidaan tarvittaessa tallentaa kantaan. Lisäksi tietokantaan on luotava Liferay'ta varten käyttäjä, jolla on lukuoikeuksien lisäksi oikeus esimerkiksi luoda ja tuhota tauluja kannasta.

Huoltoportaalin prototyyppi voidaan Liferay'n asennuksen jälkeen kääntää palvelimelle. Kääntämisen tekeminen käytännössä riippuu valitusta kehitysympäristöstä, mutta ainakin portaalisovelmat, teema ja lokalisointitiedostot on kaikki käännettävä palvelimelle. Kun kaikki prototyyppiin liittyvät tiedostot on saatu käännettyä, täytyy Liferay'n puolella ottaa uusi teema käyttöön (luku 7.2.5) sekä luoda uudet sivut halutuille toiminnoilla ja liittää tarvittavat portaalisovelmat näihin. Lisäksi prototyypin käyttämät roolit ja käyttäjätiedot on lisättävä Liferay'n hallintapaneelin kautta (luku 7.2.2). Mikäli prototyypistä siirrytään tuotantokäyttöön, tulisi nämä toiminnot automatisoida mahdollisimman pitkälle, jotta asennus saataisiin nopeammaksi ja yksinkertaisemmaksi.

7.2.2 Uuden roolin luonti

Liferay sisältää oletuksena jo useita rooleja, joita käyttäjille voi antaa. Nämä valmisroolit kertovat lähinnä, mitä oikeuksia käyttäjällä on portaalin muokkauksen kannalta, joten ne eivät suoraan sovellu prototyypin vaatimiksi käyttäjärooleiksi. Siispä tarvitaan uusi rooli jokaiselle prototyypin kuvauksessa esitellyille käyttäjäryhmälle: huoltopäällikölle, huoltohenkilölle ja asiakkaalle. Uuden roolin lisääminen tapahtuu Liferay'n puolella portaalin asetuksista roolit-toiminnosta, jossa tarvitsee kertoa vähintään uuden roolin nimi. Kun rooli on saatu luotua, voidaan siihen liittää käyttäjiä samasta toiminnosta. Käyttäjä voi tarvittaessa kerrallaan kuulua useisiin rooleihin, kunhan rooleja ei suunnitella toisiaan poissulkeviksi.

Prototyypille tärkeää on käyttäjän tunnistaminen paitsi Liferay'ssä, myös Oscar PRO:ssa. Tunnistamista varten tarvitaan siis lisää tietoa Liferay'n käyttäjätietojen yhteyteen. Liferay'n portaalin asetuksista kustomoidut kentät -toiminnosta (eng. custom fields) voi lisätä esimerkiksi käyttäjille uusia datakenttiä. Prototyypin tapauksessa käyttäjille on lisätty *Loginid*-kenttä, joka kuvautuu roolista riippuen tiettyyn tunnistetietoon OscarPRO:ssa. Esimerkiksi loppuasiakaan tapauksessa *Loginid* kertoo käyttäjän asiakastunnuksen, kun taas huoltohenkilön tapauksessa kentästä selviää työntekijän tunniste. Prototyypissä nämä vaiheet tehtiin manuaalisesti, koska rooleja ja käyttäjiä oli vielä hyvin pieni määrä. Jatkokehityksenä perusroolit voitaisiin luoda Liferay'n tietokantaan automaattisesti jollain skriptillä ja käyttäjät puolestaan voitaisiin tuoda Oscar PRO:n kannasta suoraan Liferay'n puolelle, jolloin jokaista asiakasta ei tarvitsisi käsin liittää Oscar PRO:n tietomalliin.

7.2.3 Uuden portaalisovelman luonti

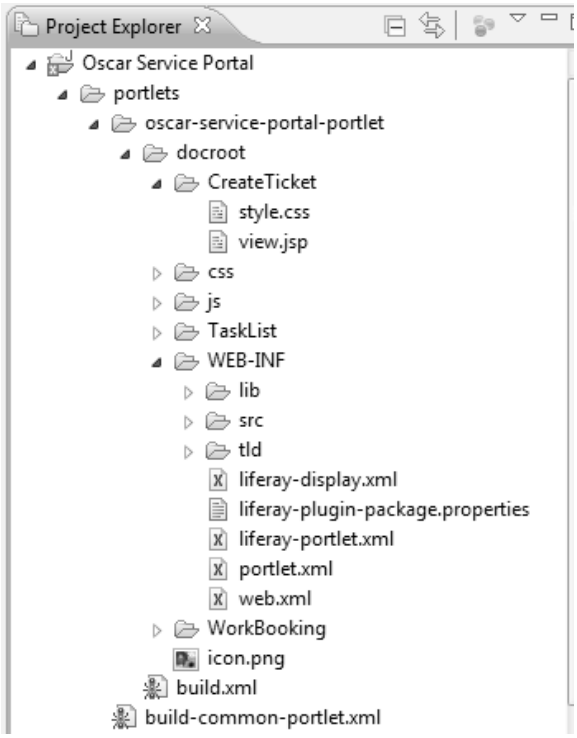
Portaalisovelmien luontiin voidaan käyttää lähes mitä vain Javaa ymmärtävää editoria ja kääntäjää. Tässä projektissa prototyypin teossa käytettiin Eclipsen Java-editoria[9] ja koodin kääntämiseen Apache Ant -ohjelmaa[10], sillä Eclipseen on saatavilla Liferay'n oma

9 Eclipse IDE for Java EE Developers [WWW] [Luettu 18.4.2013]: http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/junosr2

10 Apache Ant [WWW] [Luettu 18.4.2013]: http://ant.apache.org/

ohjelmointiympäristö Liferay IDE[11]. Tässä yhteydessä käydään tarkemmin läpi vain CreateTicket-portaalisovelman luonnin vaiheita ja siitä erityisesti asiakaslistan hakeminen toiminnanohjausjärjestelmän tietokannasta. Muut prototyypin käyttämät sovelmat luodaan vastaavalla periaatteella, vaikka niiden sisäinen logiikka luonnollisesti eroaakin esimerkistä.

Käyttämällä Liferay IDE:ä, luodaan tarvittavat tiedostot ja alkuasetukset automaattisesti, jolloin esimerkiksi Eclipsen projektiselaimella saadaan kuvan 8 mukainen näkymä. Kuvassa näkyvät myös prototyypin kolmelle esimerkkitoiminnolle lisättyjen sivujen omat kansiot (CreateTicket, TaskList ja WorkBooking), jotka sisältävät kunkin toiminnan CSS-tyylimääreet ja sirpaleen luovan JavaServerPages-tiedoston (JSP). Kansioista WEB-INF on erityisen tärkeä, sillä se sisältää sekä portaalisovelman konfiguraation, että näkymien Java-koodin (src-hakemistossa). Konfiguraatiotiedostoista *portlet.xml* on JSR-286 -standardin mukainen kuvaus portaalisovelmasta. Tämän lisäksi Liferay'n kanssa voi käyttää kolmea muutakin konfiguraatiotiedostoa: *liferay-portlet.xml*, *liferay-display.xml* ja *liferay-plugin-package.properties*. Näistä *liferay-portlet.xml* sisältää Liferay'n omia laajennoksia JSR-286 -standardiin, *liferay-display.xml* määrittelee portaalisovelmien esitystavan Liferay'n portaalisovelmien lisäystyökalussa ja *liferay-plugin-package.properties* kuvaa portaalisovelman riippuvuuksia muihin JAR-tiedostoihin. Konfiguraatiotiedostoille muodostetaan Liferay IDE:llä riittävä perussisältö automaattisesti, eikä niitä näin ollen välttämättä tarvitse muuttaa. Polku käytettyyn JSP-tiedostoon haetaan kutsumalla sovelman Java-lähdekoodista *getJSPPath*-metodia.



Kuva 8: Prototyypin portaalisovelmien tiedostorakenne Eclipsessä.

11 Liferay IDE[WWW][Luettu 18.4.2013]:http://www.liferay.com/downloads/liferay-projects/liferay-ide

Luodut konfiguraatiotiedostot sisältävät tiedon jokaisesta nimiavaruuden alle luodusta portaalisovelmasta. Ohjelmassa 1 on esitetty *portlet.xml*-tiedoston sisältö, josta on karsittu muiden kuin CreateTicket portaalisovelmien määritykset. Konfiguraatiossa riviltä kuusi alkaa portaalisovelman esittely, jossa *portlet-name* määrittelee sovelman yksilöllisen tunnukset, jonka perusteella sovelmat voidaan erotella toisistaan. *Display-name* määrittelee sovelman käyttäjälle näkyvän nimen ja *portlet-class* puolestaan Java-tiedoston, joka sisältää sovelman toiminnallisuuden. *Init-param* sisältää tiedot alustusparametreista ja *expiration-cache* vaikuttaa välimuistin vanhentumiseen. *Supports* määrittelee sovelman tukemat tietotyypit, joita käytetään muun muassa navigoinnin luontiin portaalissa. *Portlet-info* kertoo tietoja sovelmasta itsestään, eli se on käytännössä metadataa. *Security-role-ref* kuvaa, millä rooleilla on oikeus käyttää sovelmaa.

```
1  <?xml version="1.0"?>
2  <portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
3  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
5  http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd" version="2.0">
6      <portlet>
7          <portlet-name>oscar-service-portal-create-ticket</portlet-name>
8          <display-name>Oscar Service Portal Create Ticket</display-name>
9          <portlet-class>
10             fi.oscar.osp.portlets.createticket.CreateTicketPortlet
11         </portlet-class>
12         <init-param>
13             <name>view-template</name>
14             <value>/view.jsp</value>
15         </init-param>
16         <expiration-cache>0</expiration-cache>
17         <supports>
18             <mime-type>text/html</mime-type>
19         </supports>
20         <portlet-info>
21             <title>Oscar Service Portal Create Ticket</title>
22             <short-title>Oscar Service Portal Create Ticket</short-title>
23             <keywords>Oscar Service Portal Create Ticket</keywords>
24         </portlet-info>
25         <security-role-ref>
26             <role-name>administrator</role-name>
27         </security-role-ref>
28         <security-role-ref>
29             <role-name>guest</role-name>
30         </security-role-ref>
31         <security-role-ref>
32             <role-name>power-user</role-name>
33         </security-role-ref>
34         <security-role-ref>
35             <role-name>user</role-name>
36         </security-role-ref>
37     </portlet>
38 </portlet-app>
```

Ohjelma 1: CreateTicket-portaalisovelman konfiguraatio portlet.xml-tiedostossa.

Edellä mainituista kohdista *portlet-class* rivi on lisättävä sovelman konfiguraation silloin kun tarvitaan varsinaista toiminnallisuutta pelkän tiedon esittämisen sijasta. Tiedoston *liferay-portlet.xml* sisältö laajentaa *portlet.xml*:n sisältöä kertomalla, mitä kuvaketta portaalisovelma käyttää, voiko samalla sivulla olla sovelmasta useampaa instanssia yhdellä kertaa sekä mitä CSS- ja JavaScript-tiedostoja tarvitaan.

Kun konfiguraatio on valmis, voi varsinaisen portaalisovelman luonti alkaa. Kokonaisuuteen kuuluu siis toiminnallisuuden sisältävä Java-luokka, sisällön näyttävä JSP-tiedosto ja tarvittavat CSS-tyylitiedostot ja JavaScript-tiedostot. Ohjelmassa 2 on esitetty CreateTicket-portaalisovelman *view.jsp*-tiedoston sisältö siltä osin, mitä asiakaslistan hakeminen vaatii. Ensin on tarkistettava, saako portaalisovelman näkevä käyttäjä edes nähtäväkseen asiakaslistausta. Tätä varten JSP-koodissa voidaan käyttää sille ominaista skripti-elementtiä, scripletiä, jossa *TicketPermissionUtil*-luokka tarkistaa Liferay'ltä kyseisen käyttäjän oikeudet sovelman toiminnallisuuteen ja palauttaa vastauksen takaisin skriptille. Tarkistuksen tekee Liferay'n mukana tuleva *PermissionChecker*-luokka, jota kutsutaan *TicketPermissionUtil*-luokan kautta. Välivaihe on tarpeellinen, jotta tarkistuksen vaatimat tiedot muun muassa toiminnosta ja käyttäjästä saadaan kapseloitua pois näkymän koodilta. Tämän tarkastelun tekeminen kertaalleen sivunlatauksen yhteydessä riittää, sillä tulos voidaan pitää tallessa muuttujassa. Rivit 7-16 sisältävät asiakaslistan näyttämiseen tarvittavan HTML-merkkauksen, sekä siihen liitetyt Liferay'n käyttämät omat merkkaukset. Rivillä yhdeksän Liferay'n käyttöliittymäkerrokselta pyydetään lokalisoitu teksti avaimelle *osp-create-ticket-select-customer*, joka suomeksi portaalia käyttävälle palauttaisi arvon ”Valitse asiakas”. Rivillä 14 asiakaslistalle luodaan oma yksilöllinen tunniste-ID, jotta se voidaan tunnistaa muualla koodissa. ID:tä luotaessa on käytettävä portaalisovelman omaa nimiavaruutta, jotta vältytään mahdollisilta ristiriidoilta muiden portaalisovelmien määrittelemien tunnisteiden kanssa. Nimiavaruus voidaan pyytää Liferay'lta <portlet:namespace/>-merkkauksella. Lista puolestaan täytyy luoda dynaamisesti tietokannasta haetusta tiedosta, joten tarvitaan Java-koodia ja sen kutsumiseen puolestaan JSP-tiedoston JavaScript-koodia riveiltä 19-48. JavaScript-koodia kirjoitettaessa on käytetty apuna jQuery-kirjastoa[12]. Ready-funktio on osa jQueryä ja se suoritetaan aina sivunlatauksen yhteydessä, jolloin käyttäjän voidessa valita asiakkaan, kutsutaan *loadCustomerList*-funktiota. Roolipohjaisuus ilmenee jälleen else-haarassa, jonne esimerkiksi loppuasiakas päätyisi, sillä asiakaskoodi saadaan suoraan selville jo tämän kirjautuessa portaaliin. Riveillä 30-32 on esimerkkejä muiden funktioiden kutsuista, joilla muun muassa voidaan näyttää latausanimaatio, kunnes lista on saatu haettua.

12 jQuery-kirjasto [WWW] [Luettu: 18.4.2013]: http://jquery.com/

```
1  <%
2  boolean canSelectCustomer =
3          TicketPermissionUtil.hasPermissionToSelectCustomer(renderRequest);
4  %>
5
6  <% if ( canSelectCustomer ) { %>
7      <div class="osp-create-ticket-page-element">
8          <div class="osp-create-ticket-page-title">
9              <liferay-ui:message key="osp-create-ticket-select-customer" />
10         </div>
11
12         <select size="10"
13         class="osp-create-ticket-list osp-create-ticket-customer-list"
14             id="<portlet:namespace/>customerList">
15         </select>
16     </div>
17 <% } %>
18
19 <script type="text/javascript">
20 $(document).ready( function() {
21     <% if ( canSelectCustomer ) { %>
22         <portlet:namespace/>loadCustomerList();
23     <% } else { %>
24         <portlet:namespace/>loadDeviceList();
25     <% } %>
26 } );
27
28 <% if ( canSelectCustomer ) { %>
29     function <portlet:namespace/>loadCustomerList() {
30         <portlet:namespace/>showLoadingAnimation();
31         <portlet:namespace/>disable("<portlet:namespace/>page1NextButton");
32         <portlet:namespace/>setEmptyListInfoVisible("customer", false);
33         <portlet:namespace/>makeActionRequest( "getCustomerList", {},
34             function(data) {
35             var listElement = $("#<portlet:namespace/>customerList");
36             <portlet:namespace/>populateList( data.customers, listElement );
37
38             if ( data.customers.length == 0 ) {
39             <portlet:namespace/>setEmptyListInfoVisible("customer", true);
40             }
41
42             <portlet:namespace/>hideLoadingAnimation();
43             <portlet:namespace/>adjustContainerHeight(true);
44             }
45         );
46     }
47 <% } %>
48 </script>
```

Ohjelma 2: CreateTicket-portaalisovelman view.jsp tiedoston sisältö asiakaslistan lataukseen tarvittavalta osalta.

Riviltä 33 alkava *makeActionRequest*-funktiosta ilmenee hyvin Liferay'n tapa jakaa portaalisovelman suoritus kahteen vaiheeseen: toiminta (eng. action) ja renderöinti (eng. render). Sivun latauksen yhteydessä suoritetaan jokaiselle portaalisovelmalle renderöintivaihe, jossa tarkoituksena on näyttää sovelman sen hetkinen tila käyttäjälle. Toiminta-

vaihe puolestaan voidaan suorittaa vain yhdelle portaalisovelmalle kerrallaan, ja sen aikana suoritetaan kaikki portaalisovelman Java-puolelle lähettämät toimintapyynnöt. Vaihejako on tehty, jotta pyynnöt suoritettaisiin vain kertaalleen kutakin sivunlatausta kohden. Esimerkkitapauksessa *makeActionRequest*-funktio tekee Ajax-kutsun *CreateTicketPortlet.java*-tiedostolle ja pyytää *getCustomerList*-nimiseltä prosessilta vastausta. Kutsun suorituksen jälkeen vastaus saadaan JSON-muodossa *data*-muuttujaan ja se voidaan käsitellä. Esimerkin tapauksessa aiemmin HTML-merkkauksella luotu asiakaslista, jolle annettiin yksilöllinen ID, täytetään nyt *populateList*-funktiossa.

Kun näkymä on saatu luotua, on vuorossa Java-puolen tekeminen. Java-tiedostossa on huomattava, että portaalisovelman toiminnallinen luokka tulee periyttää joko Liferay'n MVCPortlet-luokasta tai GenericPortlet-luokasta. Prototyypin tapauksessa GenericPortlet-luokasta on periytetty vielä prototyypin sovelmille yhteisiä ominaisuuksia sisältävä OSPBasePortlet-luokka, josta CreateTicketPortlet on puolestaan periytetty. Lisäksi konfiguraatiotiedostoon *portlet.xml* on vielä lisättävä *portlet-class*-rivi, ellei sitä jo aiemmin tehnyt.

```
1  @ProcessAction(name="getCustomerList")
2  public void handleGetCustomerList(ActionRequest request,
3                                    ActionResponse response) {
4
5      JSONObject result = JSONFactoryUtil.createJSONObject();
6      JSONArray customerArray = JSONFactoryUtil.createJSONArray();
7
8      if ( TicketPermissionUtil.hasPermissionToSelectCustomer(request) ) {
9          String loginId = OspUtils.getCurrentLoginId(request);
10         OscarBackendCommon backend = AdapterFactory.getBackendCommon();
11         List<Customer> customers = null;
12
13         if ( TicketPermissionUtil.hasPermissionToViewAllCustomers(request) ) {
14             customers = backend.getAllCustomers();
15         } else {
16             String technicianId = backend.getTechnicianIdByLoginId(loginId);
17             customers = backend.getCustomersByTechnician(technicianId);
18         }
19
20         if ( customers == null ) {
21             customers = new ArrayList<Customer>();
22         }
23
24         for ( Customer customer : customers ) {
25             customerArray.put( customer.toJSON() );
26         }
27     }
28
29     result.put("customers", customerArray);
30     setJSONResponse(result, response);
31 }
```

Ohjelma 3: getCustomerList-prosessi CreateTicketPortlet.java-tiedostosta.

Ohjelmassa 3 on esitetty rivillä yksi *handleGetCustomerList*-funktio, joka toteuttaa annotaatiolla merkatun *getCustomerList*-prosessikutsun. Koska pyyntö käsitellään JSON-muodossa, luodaan aluksi riveillä viisi ja kuusi JSON-muotoiset muuttujat tuloksen käsittelyyn. Rivillä kahdeksan varmistetaan vielä uudelleen väärinkäytösten estämiseksi, saako kutsun suorittaja todella nähdä asiakaslistan. Rivillä yhdeksän pyydetään käyttäjän yksilöllinen tunniste, jonka avulla tuotannonohjausjärjestelmässä voidaan hakea oikeaan käyttäjään liittyvää tietoa tietokannasta. Rivillä kymmenen pyydetään staattiselta AdapterFactory-luokalta tietokantakutsun tekevä olio. Riippuen siitä, saatiinko asiakkaan yksilöllinen tunniste selville (esimerkiksi huoltopäälliköllä ei tällaista ole), voidaan palauttaa joko täydellinen lista tietokannassa olevista asiakkaista rivillä 14 tai vain tietylle tunnisteelle kuuluvat asiakkaat rivillä 17. Lopuksi vastaus muutetaan JSON-muotoon rivillä 25 ja palautetaan takaisin näkymälle rivillä 30.

Ohjelmassa 4 on esitetty Oscar PRO -tietokantarajapinnan *getCustomersByTechnician*-funktio. Koska prototyypin koodista pyrittiin tekemään mahdollisimman yksinkertaista ja toisaalta suuri osa Oscar PRO:n toiminnallisuutta on PL/SQL-kutsuissa, ovat rajapintafunktiot varsin yksinkertaisia ja niissä lähinnä vain parsitaan rajapinnalta saatu vastaus. Rivillä kuusi suoritetaan kutsu *get_customers_by_technician*-nimiselle PL/SQL-proseduurille, joka joko palauttaa rajapintakuvauksen mukaisen vastauksen tai heittää poikkeuksen. Mikäli pyyntö onnistuu, parsitaan vastaus riveillä 8-13 rivillä kaksi luotuun listaan. Mikäli kutsusta aiheutui poikkeus, otetaan se kiinni rivillä 16 ja kirjoitetaan seuraavalla rivillä lokiin tästä merkintä.

```
1  public List<Customer> getCustomersByTechnician(String technicianId) {
2      List<Customer> result = new ArrayList<Customer>();
3
4      try {
5          ResultSet resultSet =
6              SQLUtil.callProsedure("get_customers_by_technician",technicianId);
7
8          while ( resultSet != null && resultSet.next() ) {
9              String customerId = resultSet.getString(1);
10             String name = resultSet.getString(2);
11             Customer customer = new Customer(name, customerId);
12             result.add(customer);
13         }
14
15         SQLUtil.closeResultSet(resultSet);
16     } catch ( Exception e ) {
17         log.error("Failed to load customer list", e);
18     }
19
20     return result;
21 }
```

Ohjelma 4: Oscar PRO -tietokantarajapinnan getCustomersByTechician-funktio.

Varsinaisen tiedon toiminnanohjausjärjestelmästä hakeva PL/SQL-proseduuri on esitetty ohjelmassa 5. Proseduuri saa parametrikseen huoltohenkilön tunnuksen, jonka avulla toiminnanohjausjärjestelmästä osataan antaa vain kyseiselle huoltohenkilölle kuuluvat asiakkaat. Paluuarvona proseduuri palauttaa kursorin tulosjoukkoonsa, joka sitten Java-puolella voidaan parsia halutulla tavalla. Rivit kahdeksasta kolmeentoista sisältävät varsinaisen SQL-lauseen, jonka mukaan tieto haetaan.

```
1  FUNCTION get_customers_by_technician(
2      a_technician_id             IN VARCHAR2
3  ) RETURN ofw_types_pkg.ref_cursor IS
4
5  v_ret ofw_types_pkg.ref_cursor;
6
7  BEGIN
8
9      OPEN v_ret FOR
10         SELECT asiakas.nimi1
11         FROM asiakas, jtyo, jtrivi
12         WHERE asiakas.tunnus = jtyo.astunnus
13         AND jtyo.tyonro = jtrivi.tyonro
14         AND jtrivi.suhenktunnus = a_technician_id;
15     RETURN v_ret;
16 END get_customers_by_technician;
```

Ohjelma 5: Asiakaslistan huoltohenkilön mukaan hakeva PL/SQL-proseduuri.

Tämän esimerkkitoiminnon PL/SQL-proseduuri on varsin yksinkertainen, mutta vastaavien proseduurien avulla voidaan tehdä myös huomattavasti monimutkaisempiakin toimintoja. Prototyypissä kaikki toiminnallisuus haluttiin kuitenkin pitää kohtuullisen yksinkertaisena ja moni toiminto sisältääkin vain tämän esimerkin kaltaisia tietojen hakuja joidenkin tiettyjen ehtojen perusteella.

7.2.4 Rajapinta toiminnanohjausjärjestelmiin

Prototyypin rajapinta toiminnanohjausjärjestelmiin on toteutettu luomalla ohjelman 6 mukainen rajapintaluokka, jossa esitellään kaikki prototyypillä käytössä olevat funktiot. Oscar PRO:ta varten tämä rajapinta toteutettiin omassa luokassaan ja vastaavasti muihin toiminnanohjausjärjestelmiin prototyyppi liitettäisiin vastaavasti toteuttamalla rajapinta omassa luokassaan. Käytännössä Oscar PRO:lle rajapinnan toteuttava *CommonProAdapter*-luokka ainoastaan ottaa käyttöön Oraclen JDBC-ajurin, kutsuu PL/SQL-proseduureja ja parsii vastauksista rajapinnan mukaisen paluuarvon tai kertoo virheestä, kuten ohjelman 4 esimerkissä.

```
1  package fi.oscar.osp.service;
2
3  import java.util.List;
4  import fi.oscar.osp.service.beans.Customer;
5  import fi.oscar.osp.service.beans.Device;
6  import fi.oscar.osp.service.beans.Technician;
7
8  public interface OscarBackendCommon {
9      public void initializeBackend();
10     public void disconnectFromBackend();
11     public String getCustomerIdByLoginId( String loginId );
12     public String getTechnicianIdByLoginId( String loginId );
13     public List<Device> getDevicesByCustomer( String customerId );
14     public List<Customer> getCustomersByTechnician( String technicianId );
15     public List<Customer> getAllCustomers();
16     public List<Technician> getAllTechnicians();
17 }
```

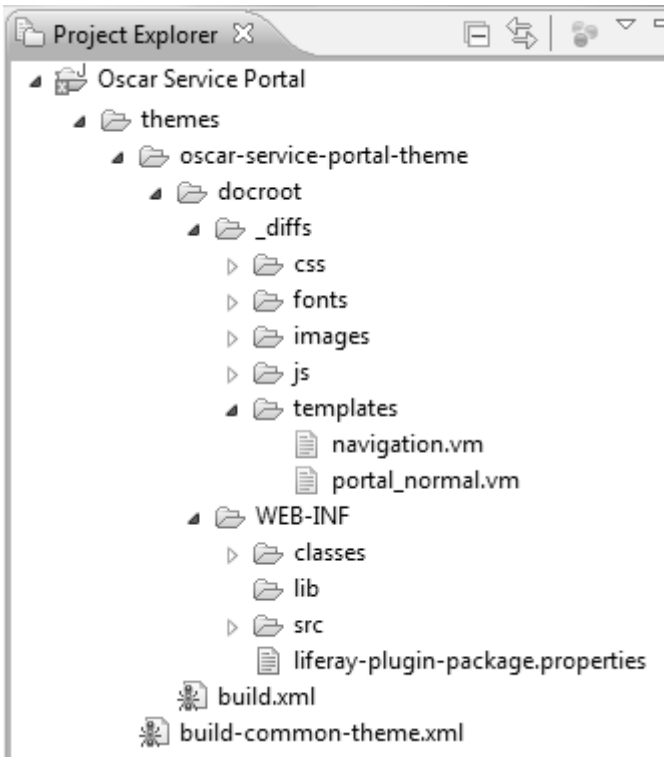
Ohjelma 6: Rajapintaluokka toiminnanohjausjärjestelmiin.

Rajapinnan monimutkaisuus riippuu siis vahvasti taustalle olevasta järjestelmästä ja sen käyttämästä tietomallista ja datansiirtotavasta. Rajapinnan toteuttavassa luokassa voidaan tarvittaessa ottaa yhteys esimerkiksi MySql-tietokantaan ja suorittaa tämän mukaisia kutsuja, tai vaikkapa lukea XML-sanomia jostain määritellystä hakemistosta. Koska rajapinnan toteutus on abstrahoitu portaalisovelmilta, täytyy sopiva rajapintaluokka olla käytössä, ilman että sitä joudutaan erikseen valita portaalisovelmien toteutusta kirjoitettaessa. Rajapintaluokan valinnasta pitää huolen tehdas-suunnittelumallin (Factory pattern) mukaan toteutettu *AdapterFactory*-luokka, jossa oikea toteutus valitaan ja palautetaan pyydettäessä portaalisovelmalle.

7.2.5 Ulkoasun kustomointi

Portaalin ulkoasun kustomointi voidaan tehdä Liferay'n teemoilla, jotka perustuvat eroavaisuuksiin alkuperäisiin teemoihin nähden. Teemojen luonti tapahtuu samaan tapaan, kuin portaalisovelmien luonti. Liferay IDE:llä voidaan luoda uuden teeman vaatima rakenne automaattisesti, jolloin saadaan suunnilleen kuvan 9 mukainen hakemistopuu. Teemojen osalta oleellinen on *diffs*-hakemisto, jonne kootaan alkuperäisestä teemasta (määritelty *build.xml*-tiedostossa) eroavat osat. Prototyypin teemassa on esimerkiksi muokattu oletussivupohjista yleistä sivupohjaa (*portal_normal.vm*) ja navigaation sivupohjaa (*navigation.vm*). Tällöin *_diffs*-hakemistoon luodaan *template*-hakemisto, jonne alkuperäiset sivupohjat kopioidaan. Kopioituja sivupohjia voi sitten muokata haluamakseen, jolloin käytettäessä kustomoitua teemaa, käytetään ensisijaisesti muokattuja sivupohjia. Sivupohjien vm-tiedostopääte viittaa Liferay'n käyttämään template-kieleen, Apache Velocityyn[13]. Sivupohjien lisäksi prototyypin teemassa on kustomoitu myös tyylimäärityksiä, fontteja, kuvia ja JavaScript-koodia.

13 Apache Velocity [WWW] [Luettu 19.4.2013]: http://velocity.apache.org/



Kuva 9: Prototyypin teeman tiedostorakenne Eclipsessä.

Kun teema on luotu, se täytyy portaalisovelman tavoin kääntää Liferay'lle, jotta uusi teema tunnistetaan. Käyttöönotto puolestaan tapahtuu Liferay'n hallintapaneelista portaalin sivuista (eng. site pages). Teema määritellään erikseen julkisille ja yksityisille sivuille ja määritykset kummassakin noudattavat sivuhierarkiaa, jolloin jokaiselle yksityiselle sivulle ei tarvitse erikseen määritellä uutta teemaa, ellei niin erikseen haluta.

Uusien teemojen lisäksi, portaalin ulkoasua voi kustomoida myös väriskeemoilla (eng. color scheme). Väriskeemat ovat osa teemaa ja näin ollen ne ovat myös teemoja rajoittuneempia kustomointimahdollisuuksiltaan. Skeemoilla otetaan käyttöön ylimääräisiä tyylitiedostoja, joilla teeman tyylimäärityksiä voidaan ylikirjoittaa värien lisäksi esimerkiksi käytettyjen kuvien osalta. Tuotantokäytössä voitaisiin esimerkiksi käyttää prototyypille luotua teemaa, josta sitten tehtäisiin jokaiselle asiakkaalle erikseen kustomoitu väriskeema, mistä ilmenisi asiakasyritykselle tunnusomaiset värit, logo ja muut kuvat.

7.3 Prototyypin arviointi

7.3.1 Liferay'n soveltuminen prototyypin taustajärjestelmäksi

Prototyypin taustajärjestelmäksi haluttiin jokin suoraan portaalit mahdollistava taustajärjestelmä, jossa järjestelmän oma koodi olisi helppo erottaa prototyyppiä varten tehtävästä koodista. Liferay on alunperin portaalijärjestelmäksi kehitetty ja se tarjoaa lisäksi sisällönhallintajärjestelmille tyypillisiä ominaisuuksia, joten se oli selkeä vaihtoehto taustajärjestelmäksi. Liferay'tä jaetaan LGPL-lisenssin[14] (Lesser General Public License) alla, joten sen kokeilu käytännössä on myös helppoa, kun lisenssimaksuja tai sopimuksia ei tarvita. Lisäksi Liferay on ohjelmoitu Javalla, joka on ohjelmointikielenä käytössä useissa muissa Oscar Softwaren järjestelmissä ja näin ollen osaamista löytyy jo valmiiksi. Tarvittaessa Liferay'lle voi kehittää portaalisovelmia myös PHP:lla, joka on toinen vahvasti tuettu ohjelmointikieli Oscar Softwarella.

Liferay'n arkkitehtuuri sallii hyvin laajan kirjon palvelin- ja käyttöjärjestelmäalustojen valintaan, ja tuettuna onkin kevyiden Java Servlet -rajapintojen lisäksi suuri osa sekä avoimen lähdekoodin palvelimista että kaupallisista Java-palvelimista (Akram et al. 2005). Oscar Softwarella on käytetty avoimen lähdekoodin Apache Tomcat[15] Java Servletiä, jonka kanssa Liferay'n saa suoraan yhteen paketoituna, joten palvelinten kannalta järjestelmän käyttöönotto oli helppoa. Useiden tuettujen palvelinten lisäksi Liferay tukee useita eri tietokantoja (SQL ja NoSQL), mikä osaltaan laajentaa mahdollisuuksia käyttää järjestelmää erilaisilla konfiguraatioilla. Asennuksen suhteen järjestelmän käyttöönotto olikin hyvin helppoa.

Yksi vaatimus huoltoportaalille oli, että sen ulkoasua olisi voitava helposti kustomoida. Liferay'llä tämä tapahtuu sekä muuttamalla sivujen sommittelua (eng. layout), että muokkaamalla teemoja (eng. themes). Sommittelun muokkaamiseen tarjolla on graafinen verkkokäyttöliittymä ja joukko valmiita vaihtoehtoja, joista uusi sommittelu voidaan valita. Teemat puolestaan luodaan hieman portaalisovelmien tapaan ja ne perustuvat aina johonkin Liferay'n sisäänrakennettuun teemaan. Uusissa teemoissa siis käytännössä kerrotaan, kuinka se eroaa vanhasta muokkaamalla tyyli- ja sivupohjatiedostoja. Uusia teemoja ei siis tarvitse alkaa rakentamaan tyhjästä, vaan ne pohjustetaan aina valmiilla teemoilla. Järjestelmän kehityksessä on myös panostettu vahvasti jQuery UI:n[16] tukemiseen[17] ja tätä hyödynnettiin vahvasti jo prototyypin kehityksessä.

Portaalisovelmien kehityksessä Liferay hyödyntää JSR 168 (JSR-000168 2003) -standardia, joten tarvittaessa kehitettyjä portaalisovelmia voidaan siirtää muillekin standardia tukeville alustoille. Toisaalta näiden alustojen portaalisovelmia voidaan myös

14 LGPL-lisenssi [WWW] [Luettu 12.4.2013]: http://www.gnu.org/copyleft/lesser.html
15 Apache Tomcat [WWW] [Luettu: 12.4.2013]: http://tomcat.apache.org/
16 Jquery UI [WWW] [Luettu: 12.4.2013]: http://jqueryui.com/
17 Jquery UI:n kehittäjä siirtyi kehittämään Liferay'tä [WWW] [Luettu 12.4.2013]: http://blog.jquery.com/2008/01/23/jquery-ui-and-beyond-the-jquery-liferay-partnership/

tuoda pienillä muokkauksilla Liferay'n puolelle. Portaalikehityksen avuksi on tarjolla lisäksi joukko apuluokkia, mutta näiden käyttö vie pois JSR 168 -standardista ja näin ollen haittaa tai estää jopa kokonaan apuluokkien avulla kehitettyjen portaalisovelmien siirron (Akram et al. 2005.)

Liferay soveltuu siis kaikin puolin hyvin toteutetun prototyypin kaltaisen järjestelmän taustajärjestelmäksi. Se tarjosi kaikki vaaditut perusominaisuudet ilmaiseksi suoraan perusversiossaan, jolloin vältyttiin suurelta määrältä pohjustavaa työtä, jota prototyypin tekemiseksi oltaisiin tarvittu. Prototyypin kehittämiseen kului aikaa noin kaksi kuukautta määrittelyineen, mikä on melko pieni aika kokonaan uuden järjestelmän suunnitteluun ja toteutukseen. Kehitys uudelle alustalle oli siis varsin helppoa, vaikka prototyyppi sisälsikin vain pienen määrän yksinkertaisia toimintoja.

7.3.2 Käytettävyyden arviointi

Tämän työn puitteissa käytettävyyden arviointia ei suoriteta, mutta muutamia huomioita käydään läpi, jotta arviointi voidaan toteuttaa myöhemmin. Lähtökohtia käytettävyyden arviointiin on kaksi: heuristinen arviointi ja käyttäjätestaus. Jos prototyyppi ja sen luontiprosessi samaistetaan käyttäjäkeskeisen suunnittelun iteratiiviseen prosessimalliin, voidaan prototyypin valmistumista pitää ensimmäisen iteraation suunnitteluratkaisujen tuloksena. Prototyypille täytyy siis tulevaisuudessa suorittaa ainakin jonkinlainen arvio, jos sitä halutaan kehittää eteenpäin käytettävyyden saralla.

Kun käytettävyystestausta sitten päädytään tekemään, soveltuu heuristinen arviointi todennäköisesti paremmin prototyypin arviointiin, sillä prototyypillä ei ole varsinaista asiakasta, jolloin käyttäjätestauksen vaatimien loppukäyttäjien löytäminen voi olla haasteellista. Heuristinen arviointi prototyypille voitaisiin suorittaa asiantuntija-arviona, joka Kuutin (2003) mukaan voidaan toteuttaa jopa ilman varsinaisia käytettävyysasiantuntijoita. Ideanin käytettävyysasiantuntijan, Harri Klemettin (2013), mukaan todellisen käytettävyyden asiantuntijan käyttö auttaa kuitenkin selvästi arvioinnissa, sillä asiantuntija osaa eläytyä erilaisten potentiaalisten käyttäjäryhmien asemaan satunnaista testaajaa paremmin, jolloin saadaan luotettavampia tuloksia. Lisäksi arviointia suorittavan henkilön perusosaaminen on tärkeämmässä roolissa kuin erilaiset tarkistuslistat tai sääntökokoelmat, joiden avulla testausta voidaan tehdä (Klemetti 2013.)

Parempi lähtökohta käytettävyyden arviointiin saattaakin olla jakaa sovellus visuaaliseen kerrokseen, käyttöliittymäkerrokseen ja prosessitason kerrokseen (Klemetti 2013). Näistä visuaalinen kerros käsittää erilaiset informaatiografiikat, kuten ikonit ja toimintapainikkeet. Käyttöliittymäkerroksen tarkastelu puolestaan keskittyy sivukohtaisesti käyttöliittymän komponenttien toimintaan ja käytettävyyteen. Viimeinen kerros, eli prosessitaso, on mahdollisesti kerroksista tärkein. Vaikka aiemmin olisikin todettu joidenkin yksittäisten näyttöjen olevan käytettävyydeltään huippuluokkaa, ovat niissä saavutetut onnistumiset täysin turhia, mikäli käyttäjän suorittamat työprosessit kokonaisuudessaan eivät ole käytettäviä. Prosessin vaiheiden tulisi olla loogisia ja mielellään käyttäjän

käsitteelliseen malliin sopivia. Käsitteellisellä mallilla tarkoitetaan käyttäjällä mielessään olevaa mallia, joka on muodostettua joko aiemmista kokemuksista vastaavista toiminnoista tai uuden prosessin tapauksessa aluksi nähtävistä näytöistä. Tämä malli puolestaan vaikuttaa siihen, kuinka käyttäjä olettaa prosessin etenevän.

Klemetti (2013) pitää käytettävyyden testausta loppukäyttäjän käyttökokemuksen parantamisen lisäksi myös yrityksen kannalta riskienhallintana. Suunnitteluprosessin aikana olisi hyvä miettiä, kenellä loppukädessä on vastuu hyvästä käytettävyydestä. Mikäli käytettävyyttä testataan yrityksen sisäisesti, on vastuu tietenkin yrityksellä itsellään, mutta testauksen ulkoistamisen tapauksessa voidaan vastuutakin osittain siirtää. Toisaalta on muistettava, että mikäli testaus jätetään liian myöhäiseen vaiheeseen toteutusta, ei hyväkään käytettävyystestaus auta enää, kun löydettyjä ongelmakohtia ei voida välttämättä kohtuullisella vaivalla todella poistaa. Usein käytettävyystestauksessa löydetyistä ongelmista vain hyvin pienet tai hyvin suuret ongelmat puretaan ja ratkaistaan, sillä aika ja raha ei useimmiten riitä kaikkeen (Klemetti 2013.)

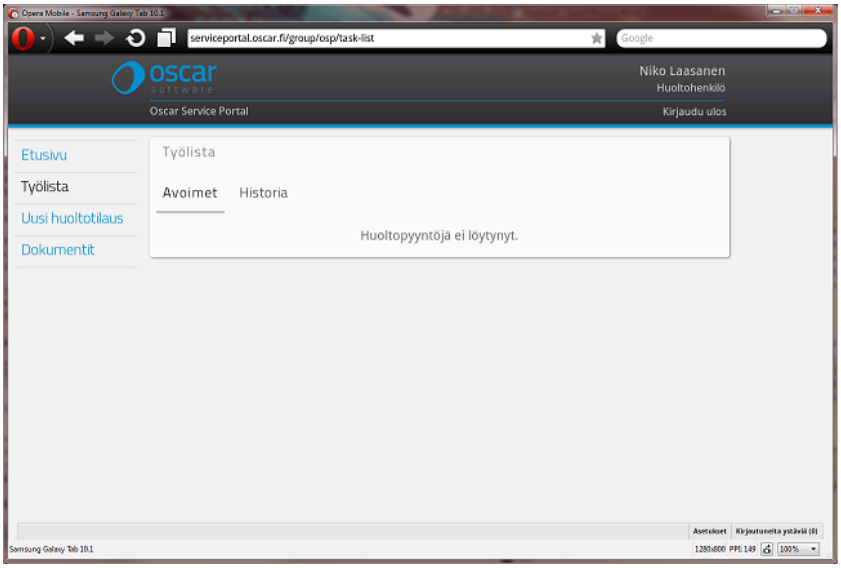
7.3.3 Hyvien mobiilikäytäntöjen huomioiminen

Koska käytettävyyttä ei arvioida heuristisilla menetelmillä, käydään tässä läpi osa W3C:n kokoamasta listasta parhaista käytännöistä Internetissä mobiilipuolella (Mobile Web Best Practices 2008). Prototyypin kehitykselle asetetuissa tavoitteissa (luku 6.2) mainittiin toiminnan mobiiliympäristössä olevan tärkeää, joten W3C:n listan kohtien täyttämistä ja huomiointia voidaan pitää tärkeänä tavoitteen täyttymisen kannalta. Listan sisältämästä 60 kohdasta käydään tässä kuitenkin läpi vain kymmenen tärkeimmiksi arvioitua niiden esiintymisjärjestyksessä alkuperäisessä listassa. Listan otsikot ovat vapaasti suomennettuja.

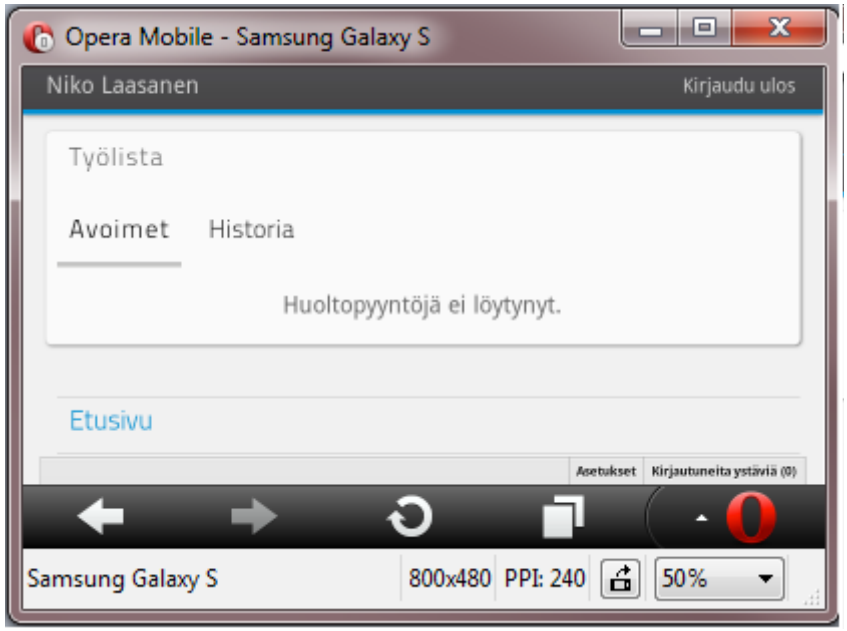
Temaattinen johdonmukaisuus ja tyylitiedostojen käyttö

Prototyypin kehityksessä on huomioitu käyttö erilaisilla päätelaitteilla, kuten pöytäkoneilla, matkapuhelimilla ja taulutietokoneilla. Päätelaitteesta riippumatta sivustosta esitetään sama versio käyttäjälle, mutta esimerkiksi asettelua on voitu muuttaa. Sivuston muokkautuminen on toteutettu responsiivisella CSS:llä, joka ikkunan koon mukaan asettelee sisällön sopivasti.

Kuvassa 10 on esimerkki siitä, miltä prototyyppi voisi näyttää taulutietokoneen ruudulla. Esimerkkikuva on otettu Opera Mobile Emulatorilla, jota on ajettu Samsung Galaxy Tab 10.0 -tilassa, käyttöjärjestelmänä Android ja resoluutiona 1280x800. Tässä tilassa prototyyppi näyttää samalta kuin ilman tarkempia responsiivisia CSS-tyylimääreitä. Kuvassa 11 on esitetty sama näkymä, mutta vaihdettu emulaattori Samsung Galaxy S -tilaan, joka käyttää myös Androidia, mutta resoluutiona on 800x480. Tällä pienemmällä resoluutiolla sivuston yläpalkkia karsitaan niin, että siitä tulee kapeampi ja näkyvissä on ainoastaan käyttäjän kirjautumistieto ja linkki uloskirjautumiseen. Lisäksi sivuston eri osien esittämistä vierekkäin vältetään pienillä resoluutioilla.



Kuva 10: Prototyyppi Opera Mobile Emulatorilla Samsung Galaxy Tab 10.1 -tilassa.



Kuva 11: Prototyyppi Opera Mobile Emulatorilla Samsung Galaxy S -tilassa.

URL-osoitteet ja uudelleenohjaukset

Sivuston URL-rakenne on toteutettu Liferay'n tarjoamille toiminnoilla URL-osoitteiden muokkaukseen. Kaikille portaalin sivuille on oma kustomoitu url-osoite, joissa ei ole käytetty tiedostopäätteitä. Uudelleenohjaukset eri sivuille on tehty Apachen mod_rewriten avulla, mutta näissä voidaan käyttää myös Liferay'n urlrewrite.xml-tiedostoa, jossa uudelleenohjauksia voi tarvittaessa määritellä.

Navigaatio

Navigaatio sivujen eri vaiheiden välillä tapahtuu sisällön reunoilla olevilla painikkeilla, jotka vaihtavat JavaScriptillä ruudun sisällön. Navigaatiovalikko puolestaan on sijoitettu suurilla resoluutioilla keskeisen sisällön vasemmalle puolelle tai pienillä resoluutioilla sisällön alle. Tällä on pyritty parantamaan käytettävyyttä erityisesti mobiilipuolella, jotta sisältö olisi käyttäjällä heti näkyvissä pelkän otsikkopalkin ja navigaatiovalikon sijaan. Tilasiirtymissä käytetyt napit on myös tehty koko sisällän korkuisiksi, jotta niiden käyttäminen erityisesti kosketusnäyttöihin pohjautuvilla käyttöliittymillä olisi mahdollisimman helppoa.

Automaattinen sivun uudelleenlataus

Mikäli käyttäjän istunto vanhenee käyttäjän vielä ollessa sivustolla, näytetään ilmoitus istunnon vanhenemisesta ja pyydetään tallentamaan tiedot ennen uudelleenlatausta. Käyttäjän on siis manuaalisesta ladattava sivun sisältö uudelleen, eikä uudelleenlatausta tehdä ohjelmallisesti esimerkiksi meta refresh -tagilla. Uudelleenlatauksesta voisi olla hyötyä monissa listauksissa, jolloin portaalia voisi pitää päätelaitteella auki ja päivitykset ilmestyisivät ruudulle automaattisesti. Mobiilipuolella sivuston automaattisesta uudelleenlatauksesta voi kuitenkin koitua käyttäjälle turhia kuluja, joten tätä mahdollisuutta ei ole käytetty.

Keskeinen sisältö

Monet prototyypin toiminnoista koostuvat useista vaiheista, jotka käyttäjän on suoritettava järjestyksessä. Toiminnot onkin jaettu vaiheisiin, kuten esimerkiksi uuden huoltotilauksen tapauksessa, jossa vaiheet ovat esiintymisjärjestyksessä asiakkaan valinta, laitteen valinta, ongelman kuvaus ja tietojen lähetys. Koko toiminto on toteutettu portaalisovelmaksi. Tämän portaalisovelman näkymä on kuitenkin rakennettu JavaScriptin avulla niin, että vain yksi vaihe on kerralla näkyvillä ja näin ollen käyttäjälle tarjotaan aina sen hetken tärkein sisältö samassa osiossa ruutua.

Jakamalla toiminnot vaiheisiin ja näyttämällä näistä kerralla vain yksi, voidaan poistaa tarve sivun vierittämiselle tai ainakin vähentää tätä tarvetta. Lisäksi vaiheistamisella voidaan vähentää päätelaitteen kuormitusta, kun myöhemmin tulevien vaiheiden sisältö voidaan ladata palvelimelta vasta siinä vaiheessa, kun käyttäjä on siirtymässä seuraavaan vaiheeseen. Lisäksi koko sivua ei tarvitse ladata uudelleen, vaan ainoastaan portaalisovelman seuraavassa vaiheessa tarvitsema tietoa.

Värikontrasti

Koska prototyyppiä voidaan olettaa käytettävän vaihtelevissa käyttökonteksteissa, joiden valaistus voi poiketa paljonkin toisistaan, on värien kontrastin erityisesti tekstin ja sen taustan välillä oltava hyvä. W3C:n saavutettavuuden ohjelistan (Web Content Accessibility Guidelines 2008) mukaan hyvä arvo kontrastisuhteelle olisi vähintään 4,5:1 ja se voidaan laskea luminositeettien suhteena. Esimerkiksi toimintojen vaiheissa käy-

tetty lähes valkoinen taustaväri on RGB arvoltaan 250,250,250 ja harmaa teksti 102,102,102. Näillä väriarvoilla saadaan luminositeettien suhteeksi noin 5,328 eli yhden desimaalin tarkkuudella yleisemmin esitetyssä muodossa 5,3:1, joka ylittää selkeästi suositellun alarajan.

Rakenne

Prototyypin HTML-merkkauksessa on käytetty HTML5-merkkausta ja sen mukaisia tageja sisällön kuvaamiseen. Esimerkiksi navigaation merkkaus on tehty nav-tageilla ja sisältö content-tageilla. W3C:n validaattoria esimerkiksi portaalin kirjautumissivu ei kuitenkaan läpäise, vaan tulokseksi saadaan 2 virhettä ja 3 varoitusta, mikä on kuitenkin varsin kohtuullinen määrä (Prototyypin validaattoritulokset 2013). Yksi validaattorin antamista varoituksista liittyi HTML5:n menu-tagiin, jota validaattori kehottaa vielä välttämään huonon selaintuen vuoksi. Muut virheet ja varoitukset liittyvät lähinnä Liferay'n tulostamaan merkkaukseen. Taulukoita sivuston rakenteen luonnissa ei ole käytetty.

Virheilmoitukset

Prototyyppiä suunniteltaessa lähtöajatuksena on ollut luvun 2.4 Shneidermanin sääntöjen mukaisesti ajatus siitä, että käyttäjän ei pitäisi voida tehdä virheitä. Tähän on pyritty esimerkiksi mahdollistamalla vaiheiden välillä siirtyminen vasta, kun kaikki tarvittava tieto on annettu. Virheitä voi kuitenkin tapahtua yhteydessä toiminnanohjausjärjestelmään, jolloin käyttäjän suorittama toiminto voi epäonnistua. Tällöin käyttäjälle kerrotaan selkokielisellä virheilmoituksella sivuston sen hetkisellä kielellä, miksi toiminto epäonnistui. Käyttäjä voi myös ennen toiminnon tallentamista siirtyä vaiheissa myös taaksepäin ja joissain toiminnoissa, kuten huoltotyön kirjaamisessa työn voi tallentaa ennen kuin koko työ on valmis, jolloin ainakin osa tiedoista pysyy turvassa, vaikka tietokantayhteys myöhemmin katkeaisikin.

Evästeet

Prototyypin käyttäminen ilman evästeitä ei ole mahdollista, sillä sen käyttö perustuu vahvasti käyttäjän tunnistamiseen. Evästeissä on kuljetettava siis vähintään käyttäjän tunnistetietoja. Liferay mahdollistaa kuitenkin myös julkisten sivujen teon ja tällaisille sivuille voitaisiinkin toteuttaa jatkossa toimintoja, jotka eivät kirjautumista vaatisi. Esimerkiksi tuotelistan hakeminen toiminnanohjausjärjestelmästä voisi olla portaalille soveltuva julkinen toiminto.

Vapaiden tekstikenttien välttäminen

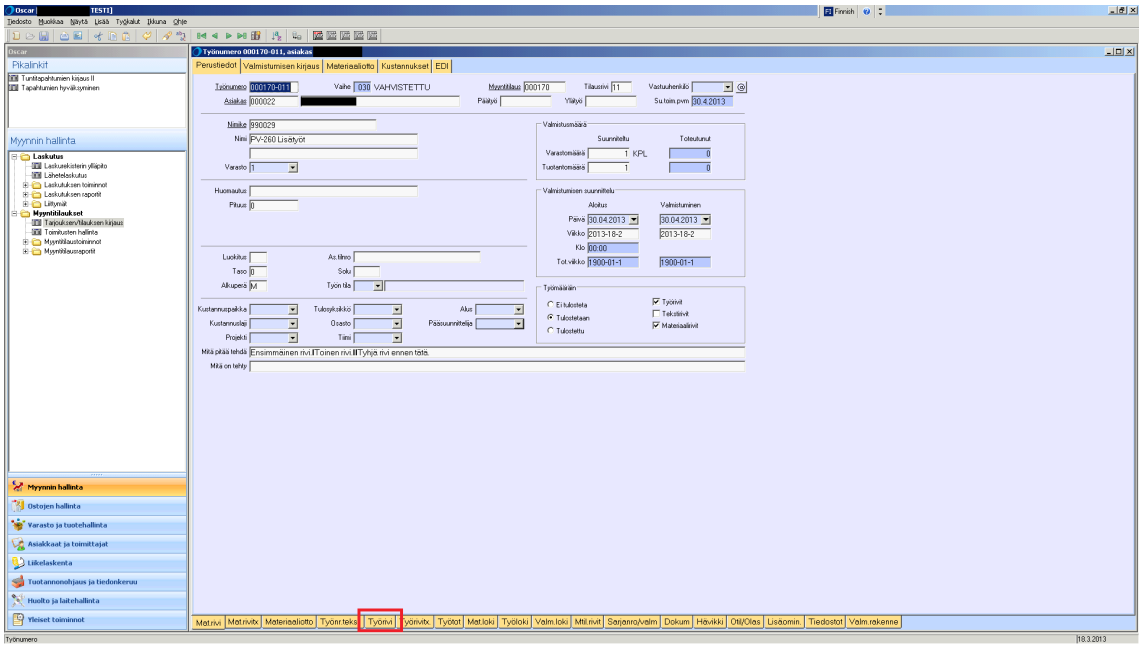
Prototyypin toimintojen käyttöliittymää suunniteltaessa vapaita tekstikenttiä on pyritty välttämään mahdollisimman paljon hyödyntämällä esimerkiksi pudotusvalikkoja ja listoja. Huoltopäällikön roolia lukuun ottamatta ainoat vapaat tekstikentät ovat loppuasiakkaalle uutta huoltotyötä kirjattaessa kysyttävä vapaamuotoinen kuvaus viasta ja huolto-

henkilölle huoltotyön kirjauksessa käytettyjen työtuntien tai materiaalimäärien lukumäärä. Esimerkiksi kaikki asiakas- tai laitelistat on esitetty rajattavina listoina ja huoltopäällikön näkemät vaihtoehdot huoltohenkilöistä pudotusvalikkoina.

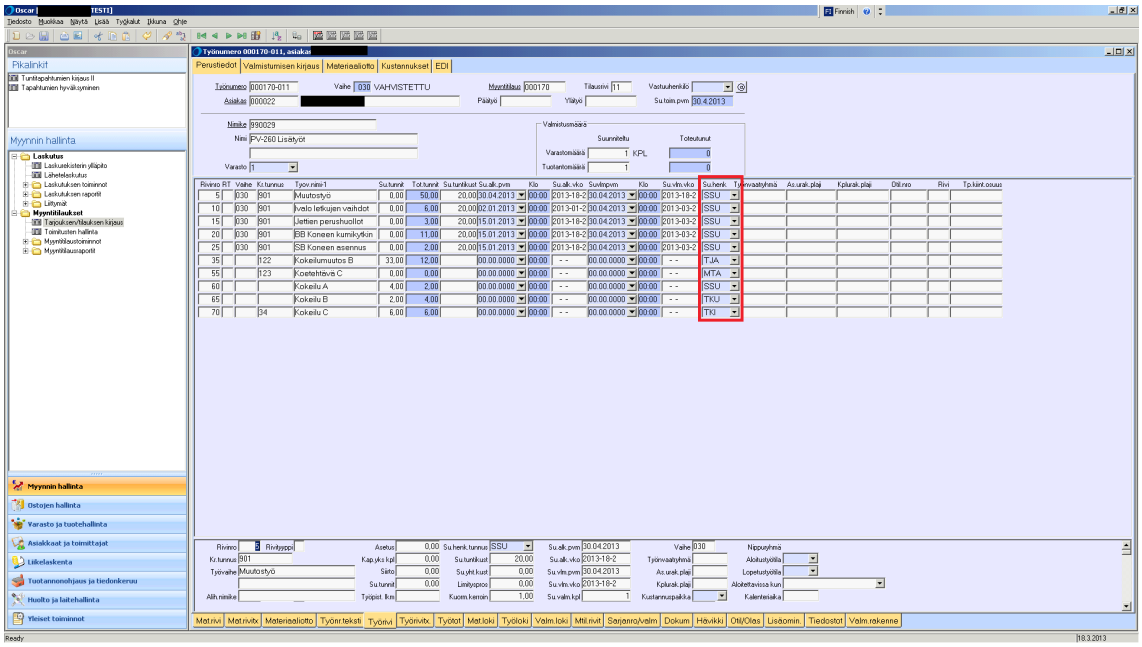
7.3.4 Huoltotyön osoitus huoltohenkilölle prototyypissä verrattuna Oscar PRO -järjestelmään

Prototyypin ja Oscar PRO:n toiminnallisuuden vertailu on aiheellista rajoittaa huoltopäällikön rooliin, sillä kuten luvussa 6.3 kerrottiin, Oscar PRO on pääasiallinen työväline tälle roolille. Huoltotyön osoitus tietylle huoltohenkilölle puolestaan on huoltopäällikön ainoa toiminto, joka prototyypissä selkeästi eroaa muiden roolien toiminnoista muutenkin kuin käyttäjälle näkyvän tiedon määrässä. Oletetaan tässä vertailussa, että huoltotilaus on jo kirjattu järjestelmään ja siihen on liitetty yksi huoltotyö (000170-011 Oscar Prossa ja Task 1 prototyypissä), johon on puolestaan tehty kymmenen työriviä. Huoltopäällikkö tuntee huoltotilauksen tunnuksen ja haluaa tarkistaa huoltohenkilöt, joille huoltotilauksen huoltotöiden työrivit on määrätty tehtäviksi ja tarvittaessa muuttaa ne. Kummassakin järjestelmässä lähtötilanne on näkymässä, joka listaa huoltotyöt, koska prototyyppi piilottaa käyttäjältä Oscar PRO:n monimutkaisemman tietorakenteen näyttäen kaikki huoltotyöt yhdessä listassa.

Oscar Prossa huoltopäällikkö avaa listasta huoltotyön 000170-011, jolloin ruudulle aukeaa kuvan 12 mukainen näkymä (esimerkkiasiakkaan tunnisteet peitetty). Koska Oscar PRO:n kautta on tarkoitus pystyä muokkaamaan kaikkea huoltotyöhön liittyvää tietoa, sisältää näkymä suuren määrän välilehtiä (23 kpl), joista jokaisella on jopa kymmeniä tietokenttiä. Näistä suoritettavan tehtävän kannalta oleellinen on työrivi-välilehti (korostettu kuvassa punaisella). Siirtymällä tälle välilehdelle, saadaan kuvan 13 mukainen listaus huoltotyöhön liittyvistä työriveistä. Välilehdeltä osan tilasta vie toisteinen tieto ensimmäisen näkymän kanssa, mutta loppuosa on varattu listalle töistä, sekä yhden valitun työn esittämiseen tarkemmin näkymän alalaidassa. Työrivit voidaan osoittaa halutulle huoltohenkilölle, jonka voi määritellä kuvassa punaisella korostetut sarakkeen alasvetovalikoista. Vaikka Oscar PRO:ssa työrivin osoittaminen tietylle huoltohenkilölle sujuukin varsin vähäisillä vaiheilla, sisältävät kummatkin näkymät todella paljon toiminnon kannalta turhaa tietoa. Mikäli käyttöliittymää ei tunne hyvin, on näiden oleellisten näkymien löytäminen vaikeaa ja aikaa vievää. Lisäksi tehtävän suoritus aloitettiin tässä esimerkissä näkymästä, jossa listataan huoltotilauksen huoltotyöt. Mikäli tähän vaiheeseen siirtyminen oltaisiin otettu mukaan tarkasteluun, olisi se lisännyt kaksi vastaavaa näkymää myös huoltotöiden listauksen hakemiseksi.



Kuva 12: Huoltotyö 000170-11 avattuna Oscar PRO:n käyttöliittymässä.

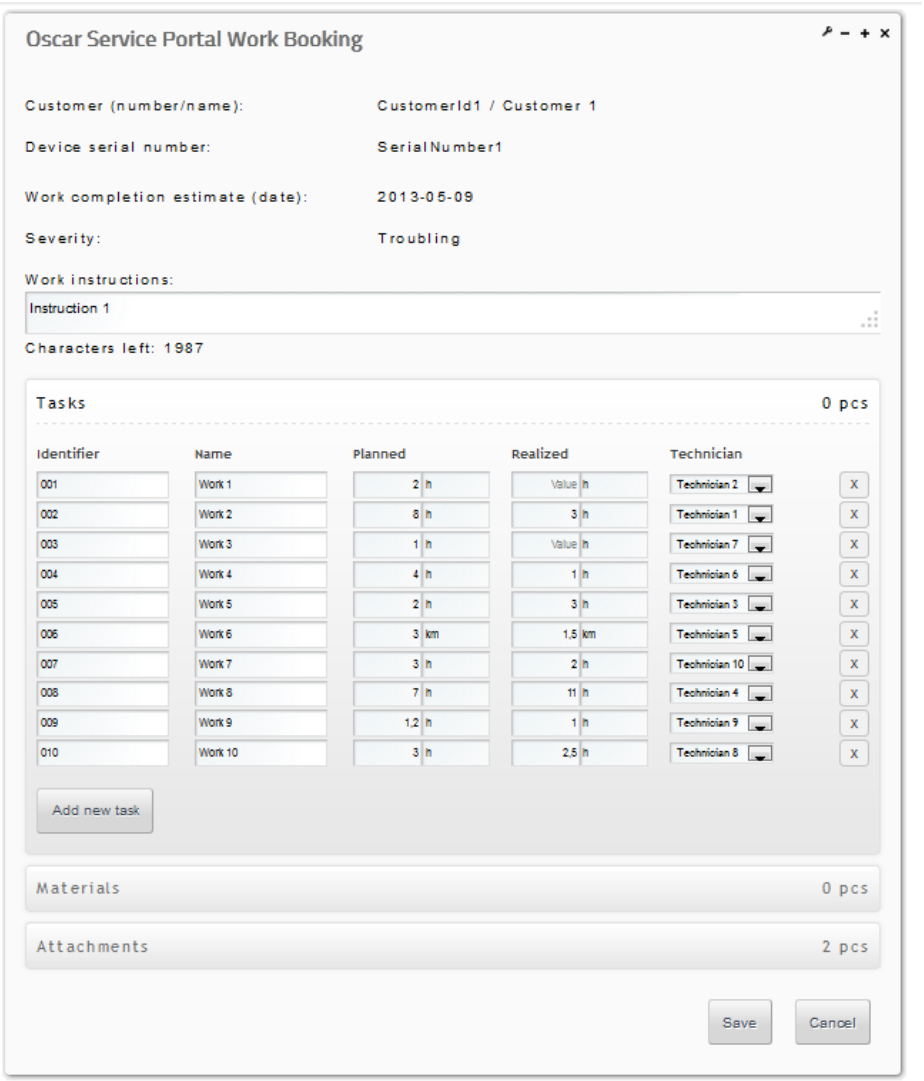


Kuva 13: Huoltotyön 000170-11 sisältämät työrivit Oscar PRO:n käyttöliittymässä tarkasteltuna.

Prototyypissä tehtävän suoritus alkaa siis vastaavasti huoltotöiden listauksesta, jossa Oscar PRO:sta poiketen näytetään kaikki järjestelmään kirjatut huoltotyöt. Kuten luvussa 6.3 jo todettiin, on huoltopäällikön pääasiallinen työväline jatkossakin yrityksen toiminnanohjausjärjestelmä. Tästä johtuen huoltotyölistausta ei ole varsinaisesti suunniteltu käytettävyydeltään parhaaksi huoltopäällikön kannalta, sillä lista voi pian kasvaa melko pitkäksi. Huoltohenkilölle vastaavassa listauksessa näytetään vain tämän omat

aktiiviset huoltotyöt, jolloin listan pituus on todennäköisesti huomattavasti lyhyempi. Listausta voi kuitenkin järjestellä eri kriteerien perusteella, jolloin työn löytäminen huoltopäälliköllekään ei pitäisi olla vaikeaa.

Löydettyään ja siirryttyään työhön Task 1, huoltopäällikkö saa nähtäväkseen kuvan 14 mukaisen työlistauksia hallitsevan portaalisovelman muodostaman sirpaleen. Muodostetun sirpaleen yläosa on varattu huoltotyön oleellisille yleistiedoille ja huoltopäällikön antamalle työohjeelle. Tasks-nimisessä haitarielementissä on listattuna huoltotyön työrivit, joille huoltopäällikkö voi technician-sarakkeen alasvetovalikoista määritellä haluamansa huoltohenkilöt. Roolipohjaisuus ilmenee tässä näkymässä muun maussa siten, että huoltohenkilö näkisi työriveistä ainoastaan hänelle osoitetut rivit.



Kuva 14: Huoltotyön Task 1 sisältämät työrivit prototyypin portaalisovelmassa huoltopäällikön roolissa.

Kuten näistä esimerkeistä huomataan, on välttämättömiä vaiheita tehtävän suorittamiseen kummallakin tavalla lähestulkoon yhtä monta. Selkein ero ja hyöty prototyypin käytöstä liittyykin näytettävään tiedon määrään ja oikeuksien rajoittamiseen. Siinä missä Oscar PRO:ssa huoltopäällikkö tai muu järjestelmää käyttävä henkilö voi hallita koko työkokonaisuutta, prototyypissä on otettu tästä kokonaisuudesta vain tarkkaan rajattu osio, jonka toimintojen määrä riippuu roolista. Jos toimintojen määrä Oscar PRO:ssa samaistetaan välilehtien lukumäärään, huomataan että jo tällä karkealla jaottelulla toimintoja on noin viisinkertainen määrä prototyyppiin nähden, kuten taulukosta 3 huomataan. Prototyypin toiminnoiksi käytetyssä portaalisovelmassa voidaan laskea työohjeen kirjoittaminen, työrivien hallinta, materiaalirivien hallinta ja liitteiden hallinta. Näin saadaan tehokkaasti rajattua ylimääräinen tieto pois käyttäjää häiritsemästä ja toisaalta myös piilotettua tieto, jota tietyn roolin käyttäjä ei saa, eikä tarvitse toiminnon suhteen nähdä.

Taulukko 3: Oscar PRO:n ja prototyypin eroavaisuudet työrivin osoitus huoltohenkilölle -toiminnon osalta.

	Oscar PRO	Prototyyppi
Siirtymien määrä	2	1
Toimintojen kokonaismäärä	23	4
Näytettävä tieto	Kaikki	Rajoitettu roolin mukaan
Mahdollisuus luoda uusi huoltotyö	Kyllä	Ei
Mahdollisuus luoda uusi työrivi	Kyllä	Kyllä

Toimintojen karsiminen rajoittaa toisaalta prototyypin mahdollisuuksia. Mikäli huoltopäällikkö haluaisikin siirtää työrivin huoltotehtävästä johonkin uuteen työhön, ei tätä ole mahdollista tehdä prototyypin avulla. Työrivejä voi kuitenkin huoltopäällikön roolissa luoda lisää myös prototyypin kautta. Huoltopäällikön rooliin ei kuitenkaan keskitytty kovin vahvasti prototyypin kehityksessä ja prototyyppiin tuotiinkin tarkoituksella lähinnä muiden roolien kanssa yhteisiin näkymiin sisällytettäviä toimintoja. Jatkokehityksessä myös huoltotöiden luonti voitaisiin haluttaessa lisätä prototyypin puolelle.

8 YHTEENVETO JA JOHTOPÄÄTÖKSET

Toiminnanohjausjärjestelmät ovat perinteisesti varsin laajoja järjestelmiä, joilla hallitaan koko yrityksen liiketoiminnallista kokonaisuutta. Suuri määrä toimintoja ja mahdollisuuksia vaikuttaa asioihin hyvinkin yksityiskohtaisella tasolla tuo paljon haasteita käyttäjille ja näin ollen myös sovelluksen käytettävyydelle ja siitä seuraavalle käyttökokemukselle. Toiminnanohjausjärjestelmillä saattaa myös olla takanaan pitkä, jopa vuosien mittainen historia, eikä sellaista järjestelmää voida korvata tai suunnitella uudelleen hetkessä.

Ongelman ratkaisemiseksi työn aiheen antajalla, Oscar Softwarella, alettiin ideoimaan helppokäyttöistä, verkkopohjaista käyttöliittymää heidän Oscar PRO -toiminnanohjausjärjestelmäänsä. Käyttöliittymän haluttiin sisältävän pienen määrän yksinkertaisia roolipohjaisia toimintoja, jotka toteuttavat kohdealueen tärkeimmät toiminnot tärkeimpien käyttäjäroolien osalta. Oscar Softwarella päätettiin lähteä hakemaan ongelmaan ratkaisua hyvin toteutuspainotteisesti prototyypillä käyttöliittymästä huollon toimialalle. Prototyypin taustajärjestelmäksi valittiin Liferay-portaalijärjestelmä.

Tutkimuskysymys koostui kahdesta osasta: kuinka tällainen käyttöliittymä tulisi toteuttaa ja mitä se sisältää, sekä soveltuuko Liferay portaalijärjestelmäksi käyttöliittymälle. Olennaisia ongelmia näihin kysymyksiin liittyen olivat, kuinka rajapinta toiminnanohjausjärjestelmään saadaan toteutettua, kuinka helppoa ja nopeaa uusien portaalisovelmien luonti on, miten roolitieto tallennetaan ja kuinka portaalikäyttöliittymän ulkoasua voidaan kustomoida. Vastauksia edellä mainittuihin kysymyksiin pyrittiin hakemaan prototyypin avulla.

Prototyyppi jakautui käytännössä neljään eri osaan: Liferay ja sen rajapinta portaalisovelmille, portaalisovelmat, rajapinta toiminnanohjausjärjestelmiin ja toiminnanohjausjärjestelmä ja niiden toimintalogiikka. Jokainen osio pyrittiin toteuttamaan niin, että tarvittaessa se voitaisiin korvata täysin tekemällä vain minimaalisia muutoksia muihin kerroksiin. Osioiden vaihtoa ei tämän työn puitteissa testattu, mutta ratkaisua pidettiin järkevänä, jotta Oscar Softwaren muitakin toiminnanohjausjärjestelmiä voitaisiin tulevaisuudessa liittää osaksi käyttöliittymiä ja toisaalta Liferay voitiin näin irrottaa muusta järjestelmästä tehokkaammin.

Kun Liferay irrotetaan arkkitehtuurin kannalta muusta prototyypistä, voidaan Liferay tarvittaessa päivittää uudempaan versioon, mikäli sen portaalisovelmille tarjoama rajapinta ei muutu. Liferay'n tuki JSR-000168 -standardille tarkoittaa myös sitä, että proto-

tyypin portaalisovelmat toimisivat muokkauksin myös muissa tätä standardia tukevissa järjestelmissä. Prototyypille luotujen portaalisovelmien jako näkymä ja logiikka -osiin selkeyttää puolestaan kehitystä ja helpottaa myös käyttöliittymän kustomointia asiakaskohtaisesti, kun toimintalogiikka voidaan pitää koskemattomana ja muuttaa vain näkymää, mikäli kyse ei ole olennaisesti toiminnallisesta muutostarpeesta.

Rajapintakerros tarjoaa portaalisovelmille yhden yhteisen rajapinnan, joka voidaan toteuttaa erikseen toiminnanohjausjärjestelmäkohtaisesti. Rajapinnan toteuttavat luokat riippuvat täysin taustalla olevasta toiminnanohjausjärjestelmästä. Oscar PRO:n tapauksessa suuri osa toiminnallisuudesta rakentuu PL/SQL-proseduurien varaan, jolloin prototyypin rajapinta voi kutsua näitä proseduureja. Keskeisenä tavoitteena onkin ollut kutsua mahdollisimman paljon valmiita proseduureja, jolloin toimintalogiikan kirjoittamisesta uudelleen vältyttäisiin. Toisaalta osa Oscar PRO:n toimintalogiikkaa sijaitsee sen arkkitehtuurin muissa kerroksissa, jolloin prototyyppi tarjoaa mahdollisuuden ja motivaation viedä toimintalogiikkaa yhä enemmän PL/SQL-proseduurien hallittavaksi ja tätä kautta keventää muita kerroksia. Yleisessä tapauksessa prototyypin kaltaisten käyttöliittymien kehityksen helppous ja nopeus riippuu siis vahvasti siitä, millaisen rajapinnan toiminnanohjausjärjestelmä tarjoaa. Mikäli logiikkaa joudutaan kirjoittamaan uudelleen paljon, on kehitykseen tarvittava työmäärä tällöin merkittävämpi.

Prototyypin kehityksen aikana huomattiin huollon toimialan olleen erinomainen prototyypin kaltaisen käyttöliittymän kohteeksi. Toimialueen tyypilliset roolit ovat pääosin asiakkaiden luona työtä tekeviä henkilöitä, kuten huoltohenkilöitä tai myyjiä, jolloin selaimen kautta toimiva järjestelmä on helppo ottaa käyttöön, eikä vaadi asiakasyritykseltä välttämättä investointeja esimerkiksi uusiin laitteisiin. Prototyypin toimintoja suunniteltaessa huomattiin myös pian yksinkertaisten kirjausten tekemisen ja listausten hakemisen olleen tärkeimpiä käyttöliittymältä kaivattuja ominaisuuksia ja tällaiset olivatkin toteutuksen kannalta myös suoraviivaisia. Lisäksi esimerkiksi asiakaslistauksen hakeminen prototyypin kautta on selkeästi nopeampaa, kuin Oscar PRO:n käynnistäminen ja oikeaan listaukseen navigoiminen sen käyttöliittymän kautta.

Jatkokehityksen haasteena voi olla roolikohtaisesti oleellisten toimintojen tunnistaminen, jotta käyttöliittymä ei muutu epäselväksi ja käytettävyydeltään hankalaksi, vaan pysyy alkuperäisessä ajatuksessa helppokäyttöisestä ja yksinkertaisesta liittymästä toiminnanohjausjärjestelmään. Mikäli toimintoja toiminnanohjausjärjestelmästä aletaan tuoda kovin paljon käyttöliittymän puolelle, palataan helposti vanhaan toimintojen paljouteen, jossa merkittävä osa käyttäjän ajasta kuluu oikean toiminnon löytämiseen varsinaisen tuottavan tekemisen sijaan. Lisäksi Oscar Softwaren toimintamalliin kuuluu asiakkaan tarpeisiin mahdollisimman hyvin vastaaminen, jolloin asiakaskohtaisilta räätälöinneiltä ei voida täysin välttyä Oscar PRO:n puolella. Tästä seuraa todennäköisesti myös tarve tehdä jonkinlaisia asiakaskohtaisia muutoksia myös rajapinnan toteuttaviin luokkiin.

Erityistä hyötyä prototyypistä on Oscar Softwaren puolella koettu olevan siksi, että se tarjoaa mahdollisuuden siirtyä vaiheittain nykyaikaisempaan arkkitehtuuriin myös toiminnanohjausjärjestelmän puolella ja toisaalta samalla laajentaa käyttäjäkuntaa aikaisemmin tavoittamattomina olleisiin käyttäjäryhmiin. Esimerkiksi Oscar PRO:sta voidaan eriyttää erilaisia moduuleja, joista sitten toteutettaisiin prototyypin kaltaisella käyttöliittymällä oleellisimmat toiminnot niille käyttäjille, joiden pääasiallisena työvälineenä ei ole Oscar PRO tai jokin muu toiminnanohjausjärjestelmä. Tälle moduulille toteutettaisiin sitten PL/SQL-proseduurit niiltä osin toimintoja, joiden logiikka sijaitsee yhä Oscar PRO:n käyttöliittymäkerroksessa. Samalla Oscar PRO:sta saataisiin vietyä pois niitä toimintoja, jotka sopivat paremmin muun tyyppisiin järjestelmiin, ja pitää toiminnanohjausjärjestelmän ydin yrityksen toimintaketjujen hallinnassa ja päätöksenteon tuen apuvälineenä. Prototyypin kaltaisista käyttöliittymistä on siis hyötyä sekä yrityksen järjestelmien kehityksen kannalta, että asiakkaiden näkökulmasta käytettävämpinä ja paremmin saatavilla olevina liittyminä oleelliseen tietoon.

LÄHTEET

Akram, A., Chohan, D., Dong Wang, X., Yang, X., Allan, R. A Service Oriented Architecture for Portals Using Portlets. [PDF] 2005, Science & Technology Facilities Council. [Viitattu: 12.4.2013] Saatavissa: http://epubs.cclrc.ac.uk/bitstream/785/406.pdf

Android User Interface Guidelines. [WWW] Android.com. [Viitattu 11.4.2013]
Saatavissa: http://developer.android.com/guide/practices/ui_guidelines/index.html

App Design Process for Windows Phone. [WWW] 2013, Microsoft.com. [Viitattu 11.4.2013] Saatavissa:
http://msdn.microsoft.com/en-us/library/windowsphone/design/hh202873%28v=vs.105%29.aspx

Babaian, T., Lucas, W., Topi, H. Identifying Usability Issues with an ERP Implementation. Waltham, USA 2005, Bentley College, Computer Information Systems Department. 6 p.

Calisir, F. Calisir, F. The relation of interface usability characteristics, perceived usefulness, and perceived ease of use to end-user satisfaction with enterprise resource planning (ERP) systems. Computers in Human Behavior 20(2004)4, pp. 505–515.

Davydov, M. Corporate portals and e-business integration. 2001. McGraw-Hill Professional. 280 p.

Factory Pattern. [WWW] Object Oriented Design. [Viitattu: 19.4.2013]
Saatavissa: http://www.oodesign.com/factory-pattern.html

Ferraioli, T. The types of ERP. [WWW] Demand Media. [Viitattu: 8.3.2013]
Saatavissa: http://smallbusiness.chron.com/types-erp-5161.html

Ferrer, J. Liferay's Architecture: The beginning of a blog series. [WWW] 2012, Liferay. [Viitattu: 25.4.2013]
Saatavissa: http://www.liferay.com/web/jorge.ferrer/blog/-/blogs/liferay-s-architecture-the-beginning-of-a-blog-series

Gable, G. G., Klaus, H. Rosemann, M. What is ERP? 2000, Kluwer Academic Publishers.

Immonen, J. Graphical User Interfaces: Luentomoniste. [WWW] 2001, Joensuun yliopisto. [Viitattu: 7.2.2013]
Saatavissa: http://cs.joensuu.fi/~jimmonen/gkl_moniste/gkl_v202.html

iOS Human Interface Guidelines. [PDF] 2012. Apple Inc. [Viitattu: 11.4.2013]
Saatavilla: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/MobileHIG.pdf

ISO 13407. Vuorovaikutteisten järjestelmien käyttäjäkeskeinen suunnitteluprosessi. 1999. Suomen Standardisoimisliitto. 59 s.

ISO 9241-11. Näyttöpäätteillä tehtävän toimistotyön ergonomiset vaatimukset. Osa 11: Käyttävyyden määrittely ja arviointi. 1998. Suomen Standardisoimisliitto. 44s.

Jacobs, F. R., Weston Jr. F. R. Enterprise resource planning (ERP) – A brief history. 2006, Journal of Operations Management. 7 p.

Jankowska, A., Kurbel, K., Nowakowski, K. A Mobile User Interface for an ERP System. [PDF] Issues on Information Systems 7(2006)2, pp. 146–151. [Viitattu: 17.1.2013]
Saatavissa: http://iacis.org/iis/2006/Kurbel_Jankowska_Nowakowski.pdf

JSR-000168. Portlet Specification. 2003, Java Community Process. 132 p.

Kalimo, A. Graafisen käyttöliittymän suunnittelu. Jyväskylä 1996, Gummerus Kirjapaino Oy. 230s.

Katz, R. N. It's a Bird! It's a Plane! It's a... Portal? In: Katz, R. N. and associates. Web Portals and Higher Education: Technologies to Make IT Personal. 2002, Jossey-Bass Inc. 14 p.

Klemetti, H. [Keskustelu] 21.3.2013

Kuoppala, H. Parkkinen, J. Sinkkonen, I. Vastamäki, R. Käytettävyyden psykologia. Helsinki 2002, Edita Oyj. 343 s.

Kuutti, W. Käytettävyys, suunnittelu ja arviointi. Saarijärvi 2003, Talentum Media.191s.

Leon, A. Enterprise Resource Planning. 2. painos. 2007, Tata McGraw-Hill Education. 370 p.

Miller, G. A. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. [PDF] Psychological Review 101(1956)2, American Psychological Association. pp. 343–352. [Viittattu: 11.4.2013]
Saatavissa: http://www.psych.utoronto.ca/users/peterson/psy430s2001/Miller%20GA%20Magical%20Seven%20Psych%20Review%201955.pdf

Mobile Web Best Practices 1.0 [WWW] 2010, World Wide Web Consortium. [Viitattu: 25.4.2013] Saatavissa: http://www.w3.org/TR/mobile-bp/

Mobile Worker Population to Reach 1.3 Billion by 2015 [WWW] 2012. International Data Corporation [Viitattu: 1.2.2013]
Saatavissa:http://www.idc.com/getdoc.jsp?containerId=prUS23251912#.UQudoGczJ8G

Molich, R. Nielsen, J. Heuristic evaluation of user interfaces. Seattle 1990. pp. 249–256.

Nielsen, J. Usability Engineering. 1993. Academic Press. 351 p.

Oscar Service Portal Työmääräarvio [XLSX]. 2013, Oscar Software Oy [Viitattu 15.2.2013] Saatavuus rajoitettu.

Prototyypin validaattoritulokset. 2013. [Viitattu: 25.4.2013] Saatavissa: http://validator.w3.org/check?uri=http%3A%2F%2Fserviceportal.oscar.fi%2Fweb%2Fosp%2Fwelcome&charset=%28detect+automatically%29&doctype=Inline&group=0&user-agent=W3C_Validator%2F1.3+http%3A%2F%2Fvalidator.w3.org%2Fservices

Sandhu, R. S. Coyne, E. J. Feinstein, H. L. Youman, C. E. Role-Based Access Control Models. [PDF] IEEE Computer 29(1996)2, pp. 38–47. [Viitattu: 4.4.2013]
Saatavissa: http://csrc.nist.gov/rbac/sandhu96.pdf

Shneiderman, B. Designing the User Interface: Strategies for Effective Human-Computer Interaction. 1987. Addison-Wesley Publ. Co. 448 p.

Tolvanen, P. Web-sisällönhallintajärjestelmä – ominaisuudet ja käyttöönotto. 2007, Jyväskylän yliopisto. 145 s.

Thexton, R. Shah, N. Gaur, H. A Role-Based Approach to Automated Provisioning and Personalized Portals [WWW] 2011. Oracle. [Viitattu: 4.4.2013] Saatavissa: http://www.oracle.com/technetwork/articles/role-based-automated-provisioning-213244.html

Vainio, T. Web-portaalit ja niiden hyödyntäminen liiketoiminnassa. 2012, Tampereen Teknillinen Yliopisto. 60 s.

Vestola, M. Essee käyttäjäkeskeisen tuotekehityksen prosessimalleista. [WWW] 2007. [Viitattu: 7.2.2013]
Saatavissa:
http://www.mvnet.fi/index.php?osio=Tutkielmat&luokka=Yliopisto&sivu=K%C3%A4ytt%C3%A4j%C3%A4keskeisen_tuotekehityksen_harjoitusty%C3%B6_2

Web Content Accessibility Guidelines (WCAG) 2.0 [WWW] 2008, World Wide Web Consortium. [Viitattu: 25.4.2013] Saatavissa: http://www.w3.org/TR/WCAG20/

What makes an ERP user satisfied? [WWW] 2012, LLP Dynamics. [Viitattu: 8.3.2013] Saatavissa: http://www.llpdynamics.com/what-makes-an-erp-user-satisfied/

What really matters for ERP users? [PDF] 2010, Abas Business Software. [Viitattu: 8.3.2013] Saatavissa: http://www.abas-software.com/en/download/marktstudien/erp-surveys_eng.pdf

Zykov, S. V. Integrating Enterprise Software Applications with Web Portal Technology. [PDF] Proceedings of 5th International Workshop on Computer Science and Information Technologies 1(2003) Ufa State Aviation Technical University [Viitattu: 1.2.2013] Saatavissa: http://arxiv.org/ftp/cs/papers/0607/0607127.pdf